Freemote Emulator: A Lightweight and Visual Java Emulator for WSN*

Timothée Maret[1], Raphaël Kummer[2], Peter Kropf[2], and Jean-Frédéric Wagen[1]

[1] TIC Institute, University of Applied Science of Fribourg,
Bd de Pérolles 80, CP 32, CH-1705 Fribourg, Switzerland
{timothee.maret,jean-frederic.wagen}@hefr.ch
[2] Computer Science Department, University of Neuchâtel,
Emile-Argand 11, CP 158, CH-2009 Neuchâtel, Switzerland
{raphael.kummer,peter.kropf}@unine.ch

Abstract. Research on Wireless Sensor Networks (*WSNs*) has developed highly optimized software environments fitting the limited hardware resource constraints of Motes. Unfortunately, these environments suffer from relatively complex programming models. Nowadays well known languages such as Java and optimized JVMs become available and simplify the application development for the Motes. Thus, we developed the Freemote Emulator which is a Java based emulator providing a lightweight emulation tool for emerging Java based Motes. It runs experiments in real time mixing real and emulated nodes. Its layered architecture and a set of predefined code templates allow developers to quickly produce runnable code for real and emulated nodes as well as predefined scenarios to help the newcomers to introduce into the system and WSNs. Our emulator provides as well a useful visualization tool based on a parametrizable slow down feature that helps to understand complex WSN behaviours and to debug tricky implementation problems. Finally, a single emulation can run on several computers, thus allowing programmers to conduct experiments with a pretty large number of emulated and real nodes.

Keywords: Wireless Sensor Networks, Lightweight Emulator, Java Based Motes, Freemote

1 Introduction

Research on software tools and applications for Wireless Sensor Networks (WSN) is much guided by the hardware constraints of Motes such as a small memory footprint, limited energy and computational power. Operating systems like TinyOS running on such components are specialized to work with these constraints but suffer from complex and hard to learn programming models and

* The work presented in this paper was partially funded by the HES-SO projects No 17223-ADS (development of the JMote) and No 16197-RASMAS (Freemote Emulator integration).

languages. Some research has been carried out to produce virtual machines that run on top of TinyOS as for example Maté[11] or SwissQM[10]. These virtual machines provide simpler and more accurate programming interfaces. They define user extensible bytecode supporting different specialized programming languages. Unfortunately, they mostly remain "application specific virtual machines" as defined in [12].

Further optimized virtual machines for well known, high level and general purpose languages like Java have recently emerged which decrease the development time and complexity. Squawk[13] and Sentilla Point[3] are two Java virtual machines optimized either for heavily limited devices such as Java Cards or more powerful devices such as the Sun SPOT[4] platform.

In this paper we present the Freemote Emulator, a novel lightweight and distributed Java based emulator which aims at providing an emulation tool for the emerging Java based Motes. Rather than on performance evaluation accuracy, this emulator focuses on behaviour credibility by mixing emulated nodes possibly distributed on many networked computers and real nodes reachable through a specialized bridge. The Freemote Emulator divides the software architecture of a Mote in three independent layers connected through well defined interfaces: *Application*, *Routing* and *Data Link and Physical*. A unique XML file modifiable thanks to a user friendly GUI permits to dynamically select and configure the proper layer implementation.

Moreover, the development of specialized code for the real nodes is not necessary while the code of *Application* and *Routing* layers can be directly run on them with no adaptation. Currently the code runs on the JMote, a Java programmable Mote prototype developed at University of Applied Science of Fribourg. These Motes are based on a IEEE 802.15.4 compliant radio chip and support the same message format (Active Message) as the TinyOS based Motes (eg. MICAz, TelosB). Consequently, they are compatible with well known Motes and can perform in a heterogeneous environment thus increasing the reality of the conducted experiments. Additionally, the emulator supports the nodes playing different roles in the network and thus providing high flexibility in the scenarios emulated.

In addition, an optional network visualization tool displays the emulated nodes, the bridge node as well as the content of the messages sent and the physical topology. The emulator provides an interesting and tunable slow down feature that allows the developer to reduce the emulation speed to easily analyse the behaviour of algorithms.

Finally, as our emulator has been developed in Java, it can be quickly and easily started from a website with a set of selectable and predefined scenarios using the Java Web Start technology. These features place the Freemote Emulator as an attractive educational tool for students and newcomers in the sensor network field. It is also an interesting scientific tool because it is highly configurable and related to the reality through the real node integration.

[3] http://www.sentilla.com/
[4] http://www.sunspotworld.com/

The remainder of this paper is organized as follows. Section 2 discusses related work in the field of sensor network evaluation tools. The Section 3 describes the design and implementation of the Freemote Emulator. Section 4 presents an evaluation of the Freemote Emulator based on the implementation of a simple scenario. Finally, Section 5 discusses the design choices, concludes and lists the possible evolution of the Freemote Emulator.

2 Related Work

Researchers in the field of Wireless Sensors Network can choose among a large set of environments and tools to test and validate their developments ranging from real testbeds to less accurate application oriented simulators. A survey of more than forty tools for wireless networks can be found in [16]. Each tool has advantages and drawbacks and should be chosen depending on the experiment to be conducted in order to provide adequate results. Most of them target Berkeley Motes platforms and TinyOS applications, if they do not target abstract Motes and use a code translation phase to provide a platform specific binary code.

Mobile Emulab[1] and Motelab[9] are testbeds that provide time shared remote access to dynamically programmable mobile or fixed network of Berkeley Motes. The experiments done with these tools are close to reality, but deal only with a relatively small number of nodes.

ATEMU[2], Avrora[3] and MSPsim[4] are "fine-grain simulators" that operate at instruction level by simulating the Berkeley Motes processor instruction sets such as the MSP430 or the AVR family. As they care about the hardware particularities, these simulators provide highly accurate evaluation results of experiments including timing or power consumption aspects. However, they require much computational power and provide poor visualization tools. TOSSIM[7] is a step by step discrete event simulator for TinyOS applications which can be coupled to TinyViz to provide an extensible visualization tool. It runs the applicative nesC code unmodified and simulates the TinyOS behavior of the components tied to hardware.

The MEADOWS[6], EMStar[8] and SENS[5] environments are more similar to our work than the previous ones, as they provide a less accurate low level and radio simulation model. Moreover, they focus on network behavior analysis more than on time based performance evaluations.

As our emulator, MEADOWS can be distributed over networked computers and simulates the radio channel using UDP sockets. It can emulate a large number of TinyOS based nodes, but does not provide any bridge to real nodes. Similar to our work, SENS is a layered and modular environment that runs applications (composed of interchangeable modules) written in C++, a high level and general purpose language. It differs from our environment as the code is not directly executable on real nodes but can be ported to Berkeley Motes. Moreover it does not support any bridge to real nodes. EMStar is an environment for testing applications for wireless networks which runs on Linux Microservers. EMStar provides the EmTOS facility that enables execution of TinyOS appli-

cations written in nesC on Microservers. This environment is highly versatile as it can mix Microservers and emulated Motes in the same experiment and can provide either simulated or real radio channels between emulated nodes based on an array of Berkeley Motes.

3 The Freemote Emulator

3.1 Heterogeneous Experiments

The Freemote Emulator performs experiments that involve both real nodes and emulated nodes connected through a bridge node in real time. Fig. 1 represents a typical deployment of an experiment. The real nodes could be any Java JME CLDC1.0 or more powerful Motes based on the IEEE 802.15.4 LR-WPAN radio standard. A new suitable Mote such as the Sun SPOT (CLDC1.1) must simply provide an implementation of the Data Link and Physical layer interface to be adapted to the Freemote Emulator. This work is relatively simple if the basic sending and receiving services are already provided.

The present version of the emulator supports the JMote platform [18, 19] developed at University of Applied Science of Fribourg under the Ad hoc Design Studio Project [17]. The JMote is based on the CC2420 radio chip like other known Motes (e.g., MICAz, TelosB).than for flexibility as they have 2MB of memory and a microprocessor supporting direct Java bytecode execution without needs of software virtual machine.

The bridge node is composed of an emulated and a real part. The real part is based on a MICAz or TelosB Mote that executes a code simply forward the messages from the radio interface to the UART and vice versa. The emulated part uses the TinyOS Java facility to send and receive messages from the UART. Fig. 1 shows that many processes (JRE *Java Runtime Environment*) to emulate nodes can be involved in the same experiment . These processes are inter-connected by an IEEE 802.3 network and can either be distributed over multiple computers or run concurrently on the same machine. Each process emulates one or more nodes, each one in a separate thread. The code for all the nodes emulated in the same process is the same but can change between processes. This feature permits to involve nodes with different roles (eg. aggregator, execution node) in the same experiment. The physical topology as well as the motion of the emulated nodes is computed by a single *Topology Manager Server* instance which provides this information to emulated nodes through *Topology Manager Client* instances. Each process contains a unique instance of the *Topology Manager Client*.The first process started in an experiment will also automatically run the *Topology Manager Server* and the GUI which displays the emulated nodes and allows to manage the evaluation.

3.2 Layered Architecture

The Fig. 2 shows the layered architecture of the emulator for the three kinds of nodes that an experiment can involve. The proper implementation for the three

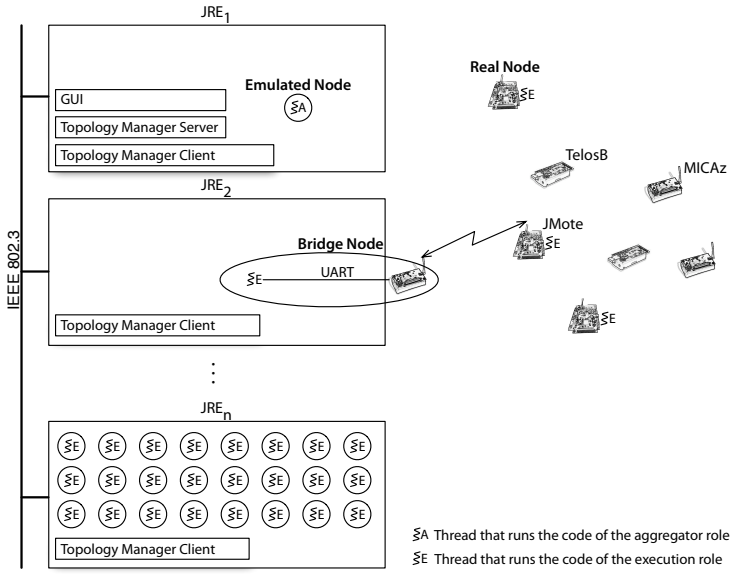

Fig. 1. Typical Deployment of an Experiment

layers is dynamically loaded with the settings defined in the XML configuration file. The multiplexer/demultiplexers define the interfaces between the layers as well as the methods to send and receive messages. Every messages sent at the *Data Link and Physical* layer or *Routing* layer is associated with an 8 bit label. This label is typically used upon reception to differentiate one type of message from another one. At the lowest layer, this information is directly mapped to the *type* field of the TinyOS Active Message, while a new field is added for the routing layer. An application that wants to receive messages must register as listener for the corresponding label on the multiplexer/demultiplexer. The lowest layer must implement an optional promiscuous mode which permits to receive every message sent by the physical neighbors, even if the destination of the message is not the current node.

The actual version provides some examples of applications such as the Ping like test, which is further detailed in Section 4, and a routing protocol based on AODV[20]. This implementation is optimized for the IEEE 802.15.4 standard as it uses the MAC level acknowledgements to detect the broken routes. The implementations of the lowest layer are tightly coupled to the kind of platform and are further detailed in the next section.

3.3 Data Link and Physical Layer Simulation

The *Data Link and Physical* layer is the only layer that has a different implementation for each kind of nodes. For the real nodes, most of the features are

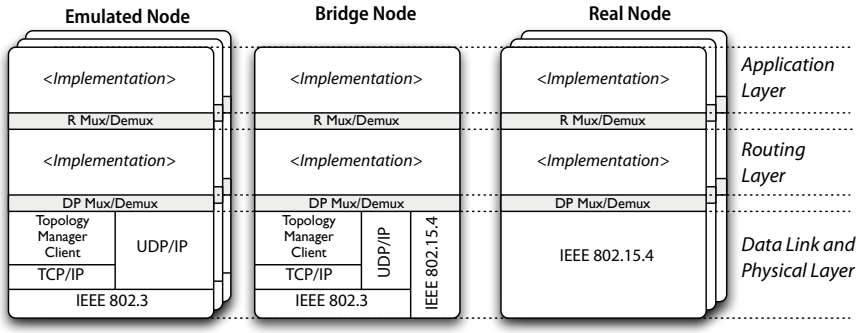

Fig. 2. Layered View of the Software Architecture

provided by the CC2420 radio chip, so the implementation simply contains a driver to this chip and a medium access control protocol similar to CSMA/CA. The emulated nodes and bridge node simulate the IEEE 802.15.4 behavior in a lightweight manner. In order to simulate the broadcast characteristic of the radio medium, a node must know its physical neighborhood before sending a message. This functionality is provided by the *Topology Manager* organised in a client/server manner. The clients act as caches and contain topological information only for the nodes emulated in the associated process. The server maintains a cartesian square map that contains all the emulated nodes and periodically updates their positions following the mobility scheme selected in the configuration file. Different mobility schema are provided from mostly random to totally pre-defined schema. The latter is useful to describe repeatable scenarios. Once this update has been completed, the server computes the new physical topology. The algorithm in the *Topology Manager* first sorts the nodes following the X axis and then creates all the physical bindings in one run through the list of nodes. It operates in $\mathbf{O}(N(\log N + C))$ time complexity, where C is the average connectivity of the network and N is the number of emulated nodes. The bindings are unidirectional as each emulated node can have its own circular radio range. During this computation, the server proactively sends the topological modifications (add or remove a neighbor) using TCP sockets to the clients. This process guarantees that the clients contain always up to date topological informations.

Every time an emulated node needs to send a message, it first picks a random value. If this value is lower than the message error rate specified in the configuration file, and if the destination address is not the broadcast address, the message is discarded. This allows to take into account the effect of poor radio transmissions. In the other cases, the node fetches its list of physical neighbors from the client and then sends a copy of the message to each of them using UDP sockets. The sending node also checks that the destination of the message is in its neighborhood and if an acknowledgement is requested, it checks that the destination node is able to send back an acknowledgement (if it contains the sending

node in its neighborhood). This simulates the IEEE 802.15.4 standard efficiently by avoiding the sending of acknowledgement messages and simply simulates the radio channel with a message error rate. However, the actual implementation of this layer does not provide a simulation of the channel collision detection which implies that two physical neighbors can send messages simultaneously without generating any error. Furthermore, it does not take into account the radio propagation delays, and considers neither the throughput nor the path loss and fading. The implementation of the *Bridge node* layer is similar except that it also sends the message to real nodes.

3.4 Simple Development Environment Based on Templates

The implementation of each layer can be easily extended as the Freemote Emulator provides templates of codes. The configuration GUI dynamically lists the selectable implementations by looking up specific package for each layer. If a developer wants to implement his own routing protocol or physical layer he can simply create a new class that implements the abstract classes provided for each layer. These classes already provide an access to the lowest layer implementations selected in the configuration file. We present below the process of building a new routing protocol implementation. This process is similar for the upper layer.

First, the `run` method (from the `Runnable` interface) must be implemented with the desired business logic. This method is automatically called by the environment at the end of the launching phase. The implementation should start by calling the following method in order to subscribe to the labeled messages exchanged at this layer: `dlpLayerSubscribeToLabel (byte label)`
Then the implementation should call the following method in order to build the protocol behavior: `dlpLayerSendMsg (byte label, I802_15_4_MPDU msg)`
Finally the following method must be implemented with the business logic code that processes the incoming messages. This method is automatically called by the environment upon reception: `dlpLayerProcessIncomingMsg (byte label, I802_15_4_MPDU msg)`

3.5 Powerfull Visualization Tool

As shown in Fig. 3, the Freemote Emulator provides a powerful visualization tool that displays in real time a map containing the *Emulated nodes* and the *Bridge node*. The position of the nodes in the map points to the position simulated by the *Topology Manager* and is updated to visualize the motion of the nodes. The physical neighbor bindings are displayed by a grey line ending with a small circle that shows the binding direction. When a node sends a message, the developer can implicitly call a method that displays the content of the message and the radio coverage on the map. Moreover, this method will toggle one of the 8 LEDs associated to the node. This is visible on the JMotes and on the emulated Motes. The latter also display a small text near the LEDs that help to understand their

meaning. This method slows down the execution speed by putting the current thread to sleep to give enough time to understand the network behavior. This is a key feature of the Freemote Emulator. The slow down factor is parametrizable in the configuration file. The standard output is redirected to the console in the GUI which logs the messages from every layer. The logging mechanism is organized with different levels identical to those defined by the framework Log4J[5] which permits to provide fine grained and context aware logs. Each layer can be parametrized independently from the others concerning the levels of displayed logs. This console helps to understand the behavior of the network as it maintains the history of the events and can display a large quantity of details.

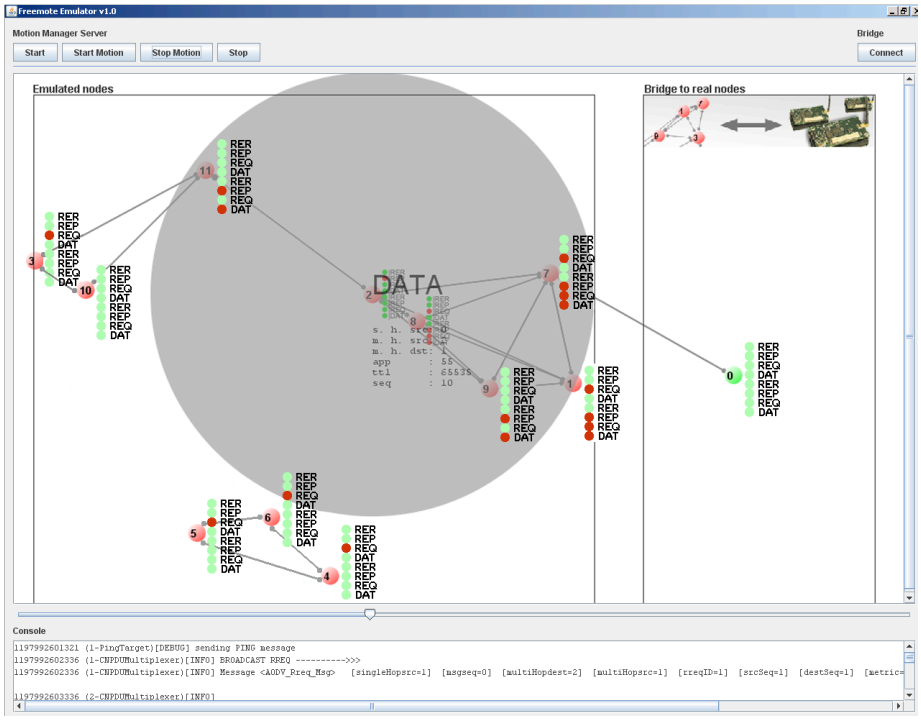

Fig. 3. The GUI with the emulated nodes visualization tool

4 Evaluation

As an evaluation example of the Freemote Emulator we describe an experiment that runs a simple Ping like application on a highly heterogeneous network com-

[5] http://logging.apache.org/log4j/

posed of emulated nodes, JMotes, MICAz and TelosB Motes. One emulated node has the role of an aggregator and runs the `PingAggregator` application that logs the evaluation data received from the execution nodes. The latter run the `PingEvaluation` application and some of them (listed in the configuration file) generate ping request. The Berkeley Motes are used only as routing nodes. They execute a modified version of NST-AODV[21], a nesC implementation of AODV for TinyOS.

This scenario does not focus on the percentage of successfull ping request nor on timing aspects such as the round trip time. Indeed, these kind of results depend heavily on many parameters settings at each layer and on the motion of the nodes. However this scenario should demonstrate that the messages will make a successfully trip through every kind of node and that the reasons some messages are not conveyed are understandable with the visual information provided by the Freemote Emulator.

4.1 Simple Ping Scenario

In this scenario, only the emulated node with the address 1 periodically generates ping requests for the real node with the address 9 (see also Fig. 4). The requests must cover the network from emulated to real nodes. The emulated nodes move following a random walk mobility model for which the nodes periodically update their position by choosing a new random angle in $[0; 2\pi[$ and a new random speed in [0.5ms; 2.0ms]. The real nodes are positionned in a linear topology and will not move during the scenario. In order to run this experiment on a desk, the power of the Motes is set to the minimal value. Fig. 4 shows a topology that could come up during the experiment. The average connectivity for the emulated nodes is set to 7, the message error rate is fixed to 3% and every emulated node has the same radio range. The logging mechanism is set in order to display all the logs of the routing layer. The slow down feature is set to 5 sec and we inserted a display instruction in the AODV implementation in order to visualize every messages sent or received by the protocol. As this protocol uses four types of messages, we have associated each LED to a couple composed of the type of message and its direction (reception or sending). A LED toggles only if the corresponding message is received or successfully sent (reception of an acknowledgment, if requested). The real part of the bridge is a Crossbow Ethernet programming base MIB600 associated to a MICAz Mote. The main parameters of AODV are set as follows : the route life time is set to a value greater than the scenario execution in order for the routes to never expire; and the feature that lets intermediate nodes to repare broken routes is disabled.

4.2 Discussion and Results

Fig. 5 shows the environment in action during the execution of the Ping scenario. After ten minutes of observation of the network behavior based on the visualization facilities, the following results can be observed. First of all, the large majority of the requests reach the destination node 9 and successfully return to

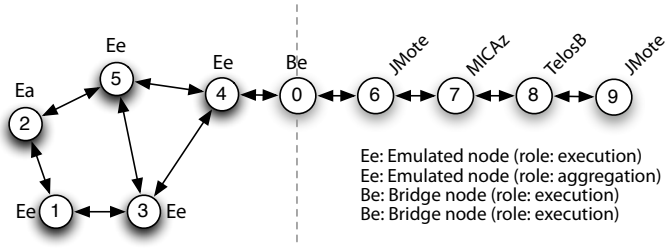

Fig. 4. Topology for the Ping scenario

node 1. As the real nodes are linearly arranged and the destination is at the end, the messages must pass through each kind of real nodes. This demonstrates that the JMote messages and those supported by the emulated nodes are compatibles with the Active Messages of TinyOS. Secondly, the visual information given by the 8 LEDs on every node is sufficient to easily understand the basic behavior of the AODV protocol such as the route construction by flooding, the construction of the direct and reverse path, and the generation of error messages to alert the originator node when a route is broken. However, to understand more complex behaviors, the set of LEDs is not expressive enough, but the emulated nodes provide more information. By looking at the content of the messages exchanged, the experimenter can easily understand the management of the sequence number or the method to choose the shortest route. The console contains all the information provided by displaying the messages as it keeps the whole history of events. However, the console has the drawback for the experimenter that he must look for the information in a huge quantity of data.

We have presented two visual and a textual feature provided by the Freemote Emulator that permit to understand the behavior of a routing algorithm. These features are complementary as they provide three distinct levels of expressiveness. Moreover, these features can be combined in a sequential manner to understand tricky implementations. Indeed, the less expressive feature will quickly point the finger at the problem without providing a clear understanding and speed up the lookup of the same problem with the more expressive feature.

5 Conclusion and Future Work

Software development in the field of wireless sensors networks is evolving towards easier programming models by adopting widely used, general purpose languages such as Java. We developed the Freemote Emulator, a highly heterogeneous and visual emulator that aims at providing a general tool for the emerging Java based Motes. The main features of this emulator provide a useful visualization tool for understanding the behavior of a WSN at different levels from simple application flows to more tricky routing algorithm concerns. We think that this tool will help developers to quickly produce applications in this field as the Freemote

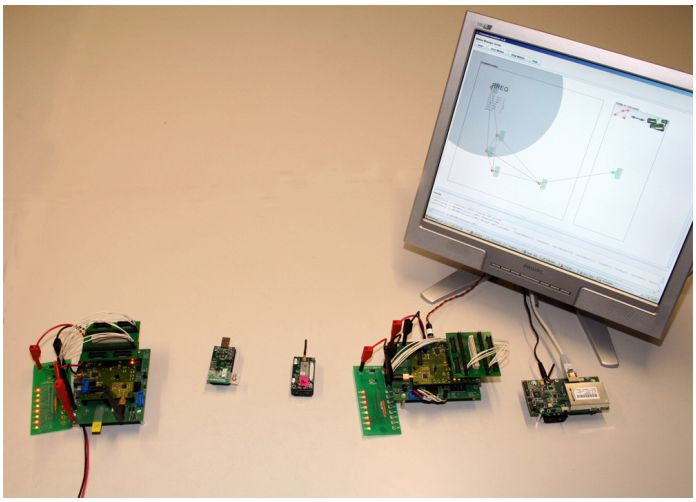

Fig. 5. The environment during the experiment. From left to right:
JMote 9; TelosB 8; MICAz 7; JMote 6; Bridge 0 (real part); Emulated nodes 1 to 5 on the screen

Emulator provides an easy to learn programming environment and assures that the code that runs in emulation mode will also run on real nodes (although currently limited to the JMote nodes). The emulator does not focus on timing aspects but gives accurate results for evaluations that do not depend on it. In [14] this emulator has been used by our research group to evaluate the performances in terms of number of messages exchanged for a DHT based lookup algorithm described in [15] over networks ranging from 100 to 10'000 nodes.

As future work, we want to add some features to the emulator. First, we are going to improve the low layer simulation by implementing a medium access control protocol in order to increase the credibility of the results from the experiments. Moreover we are going to extend the number of Java based Motes supported by the platform such as the Sun SPOT and evaluate the adaptability with the emerging Sentilla Point Java virtual machine. Finally, we want to extend the number of bridges in an experiment in order to provide more complex topologies involving both, real and emulated nodes.

The documentation, the source code and a web startable distribution with predefined scenarios are available at http://mote.tic.eia-fr.ch.

References

1. David Johnson, Tim Stack, Russ Fish, Daniel Montrallo Flickinger, Leigh Stoller, Robert Ricci and Jay Lepreau.: Mobile Emulab: A Robotic Wireless and Sensor Network Testbed. Proceedings of INFOCOM (2006) 1–12

2. Jonathan Polley, Dionysys Blazakis, Jonathan McGee, Dan Rusk and John S. Baras.: ATEMU: a fine-grained sensor network simulator. Proceedings of SECON (2004) 145–152
3. Ben L. Titzer, Daniel K. Lee and Jens Palsberg.: Avrora: scalable sensor network simulation with precise timing. Proceedings of IPSN (2005) 477–482
4. Joakim Eriksson, Adam Dunkels, Niclas Finne, Fredrik Österlind and Thiemo Voigt.: Mspsim - an extensible simulator for msp430-equipped sensor boards. Proceedings of EWSN (2007)
5. Sameer Sundresh, Kim Wooyoung and Gul Agha.: SENS: A Sensor, Environment and Network Simulator. Proceedings of the Simulation Symposium (2004) 221–228
6. Qiong Luo, Lionel M. Ni, Bingsheng He, Hejun Wu and Wenwei Xue.: MEADOWS: Modeling, Emulation, and Analysis of Data of Wireless Sensor Networks. Proceeedings of DMSN **72** (2004) 58–67
7. Philip Levis, Nelson Lee, Matt Welsh and David Culler.: TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. Proceedings of Sensys (2003) 126–137
8. Lewis Girod, Nithya Ramanathan, Jeremy Elson, Thanos Stathopoulos, Martin Lukac and Deborah Estrin.: Emstar: A software environment for developing and deploying heterogeneous sensor-actuator networks. TOSN **3** (2007)
9. Geoffrey Werner-Allen, Patrick Swieskowski and Matt Welsh.: MoteLab: a wireless sensor network testbed. Processing of IPSN (2005) 483–488
10. René Müller, Gustavo Alonso and Donald Kossmann.: A Virtual Machine for Sensor Networks. Proceedings of EuroSys (2007)
11. Philip Levis, David Culler.: Maté: a tiny virtual machine for sensor networks. Proceedings of ASPLOS (2002) 85–95
12. Philip Levis, David Gay and David Culler.: Active sensor networks. Proceedings of NSDI (2005) 343–356
13. Nik Shaylor, Douglas N. Simon and William R. Bush.: A java virtual machine architecture for very small devices. Proceedings of SIGPLAN (2003) 34–41
14. Timothée Maret, Peter Kropf and Béat Hirsbrunner.: Un environnement d'émulation hybride pour un algorithme de DHT adapté au contexte des réseaux ad hoc. Master Thesis. Universities of Fribourg & Neuchâtel (2007)
15. Raphaël Kummer, Peter Kropf and Pascal Felber.: Distributed Lookup in Structured Peer-to-Peer Ad-Hoc Networks. Proceedings of DOA (2006)
16. Erek Göktürk.: A Stance on Emulation and Testbeds, and A Survey of Network Emulators and Testbeds. Proceedings of ECMS (2007)
17. Médar Rieder, Philippe Joye, Alain Clerc, Timothée Maret, Rico Steiner and Thomas Sterren.: ADS Project. Technical Report, HES-SO RCSO.TIC (2007)
18. Alain Clerc, Philippe Joye, Médar Rieder, Nicolas Schroeter.: BlueBee: A ZigBee and Bluetooth extension board for PDA Java. Technical report of ADS Project, University of Applied Science of Fribourg, (2007)
19. Timothée Maret, Nicolas Martenet, Philippe Joye, Nicolas Schroeter.: PDA Java: A Java Based Embedded System. Technical report, University of Applied Science of Fribourg (2005)
20. Charles Perkins and Elizabeth Royer.: Ad-hoc on-demand distance vector routing. Proceedings of WMCSA (1999) 90–100
21. C. Gomez, P. Salvatella, O. Alonso and J. Paradells.: Adapting AODV for IEEE 802.15.4 mesh sensor networks: theoretical discussion and performance evaluation in a real environment. WoWMoM (2006) 159–170