SeNeTs - Test and Validation Environment for Applications in Large-Scale Wireless Sensor Networks

Jan Blumenthal, Matthias Handy, and Dirk Timmermann, Member, IEEE

Institute of Applied Microelectronics and Computer Science, University of Rostock, Germany,
e-mail: (jan.blumenthal, matthias.handy, dirk.timmermann)@etechnik.uni-rostock.de

Abstract—**We present SeNeTs, a powerful software environment for test and validation of sensor network applications. SeNeTs is not tied to a specific hardware platform. It administrates large-scale sensor networks during execution without affecting communication among sensor nodes. Moreover, SeNeTs allows efficient debugging of sensor network applications with sophisticated update mechanisms.**

***Index Terms*—Test, Validation, Wireless Sensor Networks**

I. Introduction

Applications and protocols for wireless sensor networks require novel programming techniques and new approaches for validation and test of sensor network software. In practice, sensor nodes have to operate in an unattended manner. The key factor of this operation is to separate unnecessary from important information as early as possible in order to avoid communication overhead. In contrast, during implementation and test phases, developers need to obtain as much information as possible from the network. A test and validation environment for sensor network applications has to ensure this.

Suppose a sensor network with thousands of sensor nodes. Furthermore, suppose to develop a data fusion and aggregation algorithm that collects sensor information from nodes and transmits them to few base stations. During validation and test, developers often have to change application code, recompile, and upload a new image onto the nodes. We could "flood" the network using the wireless channel. However, this would dissipate a lot of time and energy. If we used a broadcast mechanism, how could we ensure that every node runs the most recent version of our application? If we had a second communication channel for signaling and update purposes only, application updates could be done without affecting sensor network operation.

Pure simulation can produce first and important insights. However, modeling the wireless channel is difficult. Simulation tools often employ simplified propagation models in order to reduce computational efforts for large-scale networks. Widely-used simulation tools such as NS2 [1] use simplified network protocol stacks and do not simulate at a bit level. Furthermore, code used in simulations often cannot be reused on real sensor node hardware; why should developers implement applications and protocols twice?

In contrast to simulation, implementation on the target platform is often complicated. The targeted hardware itself may be still in development stage. Perhaps there are a few prototypes, but developers need hundreds of them for realistic test conditions. Moreover, prototype hardware is very expensive and far away from the targeted "1 cent/node".

Consequently, we are in need of something "in-between" that combines the scaling power of simulations with real application behavior. Moreover, administration must not affect sensor network application. The best approach is to use different communication channels for administration and application communication.

We present SeNeTs [2], a powerful software environment for test and validation of sensor network applications. SeNeTs is not tied to a specific hardware platform and allows the administration of large-scale sensor networks during execution without affecting communication among sensor nodes. Moreover, SeNeTs allows efficient debugging of sensor network applications with sophisticated update mechanisms.

The paper is organized as follows. Section II reviews related work. We describe the SeNeTs architecture in Section III. In Section IV, we illustrate how SeNeTs improves application development. We conclude our paper and discuss future research activities in Section V.

II. Related Work

A. EmStar

In recent years, several research projects started with the aim to facilitate the development process of software for wireless sensor networks. EmStar, a Linux-based software framework, is one of those projects [3,4]. EmStar supports three operational modes: pure simulation, true in-situ deployment and a hybrid mode that combines both. SeNeTs and EmStar are similar in some aspects: both run the same code in simulation and on real sensor node hardware. Additionally, both incorporate an environment model. However, there are significant differences: In EmStar, simulations and the hybrid mode are controlled by a central server. SeNeTs runs on different distributed machines and therefore is more scalable. Furthermore, EmStar code is designed to run on the mote platform [7], a support of the Stargate-platform is intended. In contrast, SeNeTs code is not tied to a specific platform.

B. TOSSIM / TinyOS

TOSSIM is a discrete event simulator for TinyOS-applications [5,6]. TinyOS is an operating system for wireless sensor networks that runs on 'motes', a hardware plat-

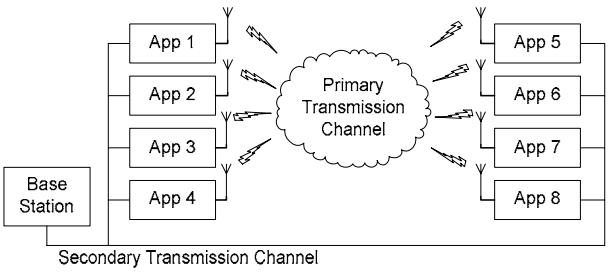

Fig. 1 Communication channels in SeNeTs

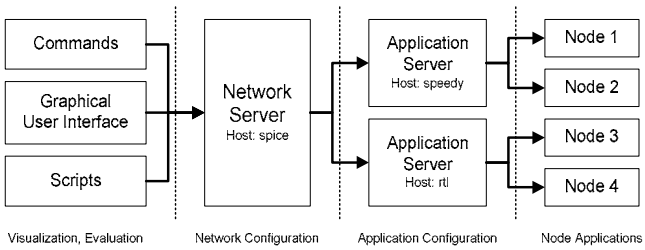

Fig. 2 SeNeTs components using the secondary transmission channel

form for sensor nodes. TinyOS applications can be compiled into TOSSIM without code changes. TOSSIM simulates the TinyOS network stack at a bit level and is strongly tied to the 'mote' platform. Thus, it cannot be used with different sensor node architectures. Furthermore, TOSSIM does not support a hybrid mode combining real and simulated nodes in one simulation.

III. SeNeTs Architecture

A. Definition of Terms

In this section, we describe the SeNeTs architecture. First, we define a few important technical terms in sensor networks. A *sensor node application* (SNA) combines all software components which are implemented on a real sensor node, e.g.:

- Sensor drivers,
- Sensor node operating system,
- Middleware such as modules for data aggregation, positioning, routing etc.

A *sensor network application* (SNWA) describes the global task of a sensor network. It includes the definitions of all sensor node applications and a collaboration description of all components. We define a *SeNeTs network* as a sensor network controlled by SeNeTs.

B. Components

Software for wireless sensor networks is often simulated due to lack of real sensor nodes and their high costs. The quasi-parallel and sequential processing of concurrently triggered events is disadvantageous compared to real-world programs.

In SeNeTs, sensor node applications run distributed on independent hosts such as PCs, PDAs, or evaluation boards of embedded devices. The parallel execution decouples applications and simulation environment. Further, it prevents a very unpleasant effect in simulation environments at concurrently triggered events as aforementioned. Without SeNeTs, this effect results in sequenced execution of created tasks and corrupted simulation output. To summarize, realistic simulations of sensor networks are complicated.

The development and particularly the validation of distributed applications are hard to realize. Especially systems with additional logging and controlling facilities affect the primary behavior of applications. Suppose, a logging message is transmitted, then an application message may be transmitted delayed. Exceptionally in wireless applications with limited channel capacity, the increased communication leads to a modified timing behavior and as a consequence to different results. Due to the degrading channel capacity of $1/\sqrt{n}$ per node in a network with n nodes, the transport medium acts as a bottleneck [8]. Thus, in wireless sensor networks with thousands of nodes, the bottleneck effect becomes the dominant part.

To eliminate the bottleneck effect, SeNeTs contains two independent communications channels as illustrated in Fig. 1. The primary communication channel is defined by the sensor network application. It is the communication method used by sensor node applications, e.g. Bluetooth or ZigBee.

The secondary communication channel is an administration channel only used by SeNeTs components. This channel transmits controlling and logging messages. It is independent of the primary communication channel and uses a different communication method, e.g. Ethernet or Ultrasound. The separation into two communication channels simplifies the decoupling of application modules and administration modules after testing.

The parallel execution of applications on different host systems requires a cascaded infrastructure to administrate the network. Fig. 2 displays important modules in SeNeTs: node applications, application servers, a network server, and optional evaluation or visualization modules. All these modules are connected via the secondary transmission channel.

C. Network Server

The network server (NS) administrates sensor networks and associated sensor nodes. The NS starts, stops or queries sensor node applications. In a SeNeTs network, exactly one network server exists. However, this NS can manage several sensor networks simultaneously. Usually, the network server runs as service of the operating system.

A network server opens additional communication ports. External programs such as scripts, websites or telnet clients can connect to these ports to send commands. These commands may be addressed and forwarded to groups or standalone components.

Furthermore, the network server receives logging messages from applications containing their current state. Optional components such as graphical user interfaces can install callbacks to receive this information.

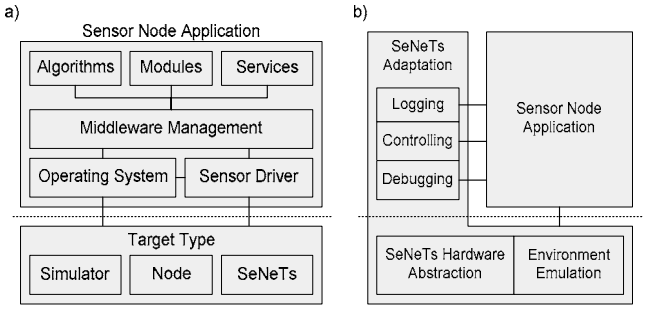

Fig. 3 a) Software layer model of a sensor node application (SNA), b) Software layer model of a SeNeTs application (SeA)

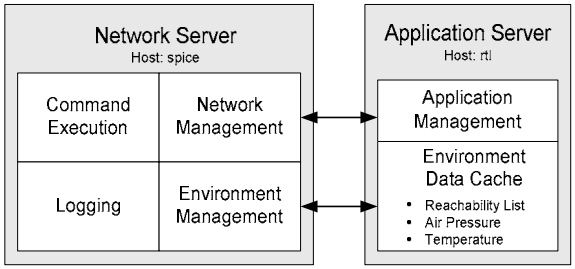

Fig. 4 Environment management in SeNeTs

D. Application Server

The application server (AS) manages instances of node applications on one host (Fig. 2). It acts as bridge between node applications and the network server. Usually, at least one application server exists within the SeNeTs network. Ideally, only one node application should be installed on an application server to prevent quasi-parallel effects during runtime.

The application server runs independent of the network server. It connects to the network server via a pipe to receive commands. Each command is multiplexed to one of the connected node applications. Moreover, if the pipe to the network server breaks, node applications will not be affected besides losing logging and controlling facilities. Later, the network server can establish the pipe again.

Generally, an application server starts as service together with the host's operating system. At startup, it requires configuration parameters of the node's hardware. With these parameters, the application server assigns hardware to node applications. Suppose a host system that comprises two devices representing sensor nodes as principally shown in Fig. 2. Then, the AS requires device number, physical position of the node, etc. to configure the dynamically installed node applications at runtime.

E. SeNeTs Application

Applications for wireless sensor nodes are usually designed based on a layered software model as depicted in Fig. 3 [9]. On top of the node's hardware, a specialized operating system (OS) is set up such as TinyOS [6]. A sensor driver contains software to initialize the measurement process and to obtain sensor data. Above the OS and the sensor driver, middleware components are located containing services to aggregate data or to determine the node's position. The aforementioned modular design allows:

- Abstraction of hardware, e.g. sensors, communication devices, memory, etc.,
- Adaptation of the node's operating system,
- Addition of optional components, e.g. logging and configuration.

The SeNeTs Adaptation is a set of components which are added or exchanged to wrap the sensor node application. Fig. 3b represents the SeNeTs Adaptation layer consisting of at least a logging component, a controlling unit, a hardware abstraction layer, and an optional environment encapsulation module. These additional components provide substantial and realistic test and controlling facilities.

An application composed of a sensor node application and SeNeTs Adaptation components is called SeNeTs Application (SeA). The sensor node application is not changed by added components. Generally, it is not necessary to adapt the sensor node application to SeNeTs interfaces. However, supplementary macros can be added to interact with the linked components.

A SeNeTs application runs as process of the host system. Due to the architecture of a sensor node application with its own operating system, the SeNeTs application runs autonomously without interaction to other processes of the host system. At startup, the SeNeTs application opens a pipe to communicate with the application server.

After the test phase, all SeNeTs components can be removed easily by recompilation of all node applications. SeNeTs specific components and logging calls are automatically deactivated due to compiler switches.

F. Environment Management

Sensor network applications require valid environment data such as temperature or air pressure. Under laboratory conditions, this information is not or partly not available. Therefore, environment data must be emulated. SeNeTs provides these environment data to the node application by the Environment Emulation module (Fig. 3b).

All Environment Emulation modules are controlled by the Environment Management of the network server which contains all pre-defined or configured data (Fig. 4). This data comprises positions of other nodes, distances to neighboring nodes, etc. If required, other data types may be added.

In the application server, the Environment Data Cache module stores all environment information required by each node application to reduce network traffic.

Optionally, position-based filtering is provided by the Environment Emulation component of SeNeTs. Especially, if large topologies of sensor nodes should be emulated under small-sized laboratory conditions, this filtering ap-

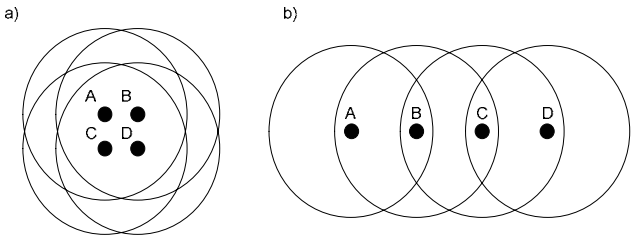

Fig. 5 a) Physically arranged sensor nodes (solid circles). All nodes are in transmission range (dotted line) of each other. b) Virtually arranged nodes with appropriated transmission ranges. Nodes are not longer able to communicate without routing.

proach is essential. Suppose real and virtual positions of nodes are known, a mapping from physical address to virtual address is feasible. A node application only receives messages from nodes that are virtually in transmission range. All other messages are rejected by the SeNeTs Adaptation components. This is accomplished by setting up a filter in the primary communication channel.

One application scenario that illustrates position-based filtering is flood prevention as described in [10]. Here, sensor nodes are deployed in sandbags, piled along a dike of hundreds of meters or even kilometers. These nodes measure the humidity and detect potential leakages. Testing this scenario under real world conditions is not practical and very expensive. However, the evaluation under real world conditions of software regarding communication effort, self organization of the network, routing, and data aggregation is most important.

Fig. 5 illustrates the difference between laboratory and real world. The left side represents laboratory conditions where all nodes are in transmission range of each other. The right side sketches the flood prevention scenario under real conditions. On the left side, the nodes A-D are in transmission range of each other. Thus in contrast to the real-world scenario, no routing is required. Next, data aggregation yields wrong results, because nodes are not grouped as they would in reality. Thus, if physical arranged nodes in test environment do not meet the requirements of the real world, the results are questionable.

Assume, node A sends a message to node D, then all nodes receive the message due to the physical vicinity in the test environment (Fig. 5a). Node C and D receive the message, but they are not in the virtual transmission range of node A. Thus, the environment emulation module rejects these messages. As a result, SeNeTs prevents a direct transmission from node A to node D. Messages can be transmitted only by using routing nodes B and C. In short, the emulation of the sensor network software becomes more realistic.

G. Message Passing

All SeNeTs components communicate via pipes with each other. Only one pipe is used between two components even between the network server and an application server. Otherwise, the number of pipes and initialization time

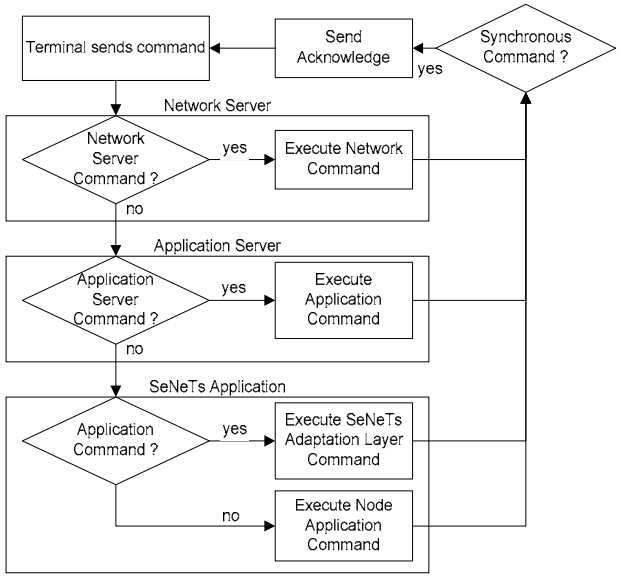

Fig. 6 Command flow in SeNeTs

would increase and would exhaust the available resources such as file descriptors and temporary buffers. To minimize the number of pipes, SeNeTs employs a cascading communication hierarchy between network server, application server, and applications as sketched in Fig. 2.

SeNeTs distinguishes three types of commands which are sent in 3 byte messages between the components:

- Network server commands
- Application server commands
- Application commands

Network server commands and application server commands are sent to configure the network server and the application server, respectively. Application commands set up the SeNeTs Adaptation modules. Developers can add optional node-specific application commands.

Fig. 6 illustrates the command flow within SeNeTs. An external application connects to the network server and sends a command. First, the command passes the network server. In case of a network server command, it is executed immediately. Otherwise, the network server forwards the command to the specified application server. If the command is an application server command, it is executed here. Otherwise it is forwarded to the appropriate SeNeTs application. Finally, the application distinguishes between SeNeTs adaptation commands and node-specific commands. The former handles control requests such as start and stop of the application. The latter is application-dependent and has to be implemented by the software developer, if necessary.

Additionally, commands are divided into synchronous and asynchronous ones. Synchronous commands demand components to be blocked until they receive an acknowledgment. Therefore, synchronous messages are critical due to their blocking behavior at source and target components. Thus, requests to sensor node applications have to be asynchronous. Synchronous commands are suitable for administration tasks.

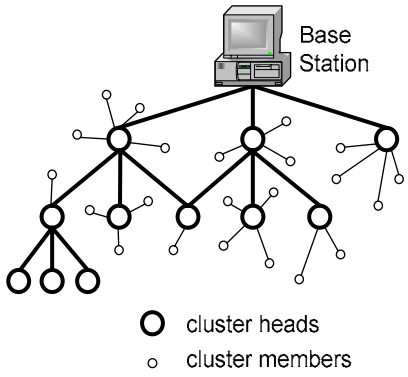

Fig. 7 Tree-like structure of a DCP sensor network

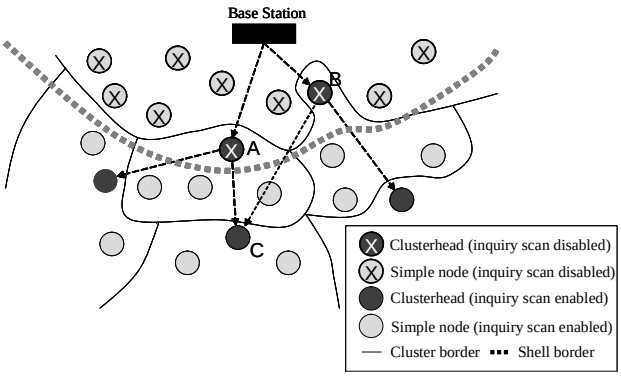

Fig. 8 Cutting from a DCP network during set-up phase illustrating the mutual discovery problem.When cluster heads that are one hop away from the base station perform an inquiry, they do not discover other one-hop-away cluster heads due to their disabled inquiry scan mode (marked by an X).

IV. Example Application

The first application we tested with SeNeTs was a Data Collection Protocol (DCP) for Bluetooth-based sensor networks [10]. The main task of DCP is to collect data from a sensor network at a central point called base station.

DCP distinguishes sensor nodes according to their role in the network. Each node holds one of the following two roles. Cluster members gather sensor data and forward the data to a cluster head. Cluster heads collect sensor data from cluster members and forward it to the base station. The whole sensor networks then consists of a set of clusters; each cluster consists of a cluster head and at least one cluster member.

DCP comprises two phases. During set-up phase, the network is formed. Base station and sensor nodes explore their vicinity and search for neighboring nodes. At the end of the set-up phase each sensor node knows at least one Packet Forward Address (PFA). If a sensor node is not able to reach the base station directly, it transmits sensor data to a PFA instead. The PFA then is responsible for data forwarding. Basically, the set-up phase produces a tree with the base station as root (Fig. 7).

When the set-up phase has finished, the network is formed and steady-state phase starts. During steady-state, sensor information is transmitted to the base station. After a certain time, the protocol enters set-up phase again and reorganizes the network. The reorganization interval depends on various network parameters, such as type and frequency of topology changes or the energy level of nodes.

We tested DCP at a computer pool with 10 Linux PCs. Each of these PCs runs one application server that acts as host for two node applications each. As communication modules we use USB-Bluetooth sticks controlled by BlueZ Bluetooth Linux driver. The network server runs on one of these PCs administrating all application servers.

We obtained essential insights from testing DCP with SeNeTs. One of them was the handling of Bluetooth page scan and inquiry scan modes during set-up phase. Fig. 8 illustrates this. Set-up phase starts with an inquiry procedure of the base station. Thereby the base station discovers all cluster heads and cluster members in range (one hop away from base station). Thereupon, all discovered cluster members themselves start an inquiry in order to discover sensor nodes that are two hops away from the base station, etc. However, how can we prevent cluster heads from mutual discovery? This can be accomplished by temporary disabling the inquiry scan mode of cluster heads until set-up phase is finished. Testing DCP within SeNeTs environment helped us to improve DCP's scan mode management.

V. Conclusion

We presented SeNeTs, a test and validation environment for wireless sensor networks. SeNeTs substantially enhances the development process of sensor network applications. SeNeTs is independent of a specific node platform. It introduces a secondary communication channel to prevent overload of the wireless transmission medium with logging and administration data. SeNeTs allows test and validation of large-scale sensor networks in small-size laboratories.

VI. References

[1] The Network Simulator - ns-2, http://www.isi.edu/nsnam/ns.
[2] The SeNeTs Project: http:/www.senets.org.
[3] J. Elson, S. Bien, N. Busek, V. Bychkovskiy, A. Cerpa, D. Ganesan, L. Girod, B. Greenstein, T. Schoellhammer, T. Stathopoulos, and D. Estrin: “EMStar – An Environment for Developing Wireless Embedded Systems Software”, Technical Report CENS-TR-9, University of California, Los Angeles, Center for Embedded Networked Computing, March 2003.
[4] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin: “Em*: a software environment for developing and deploying wireless sensor networks”, to appear in the Proceedings of USENIX 04, also as CENS Technical Report 34.
[5] P. Levis: “TOSSIM system description”, http://www.cs.berkeley.edu/~pal/research/tossim.html, January 2002.
[6] Berkeley WEBS: “TinyOS”, http://today.cs.berkeley.edu/tos/, 2004.
[7] Crossbow Technology: “http://www.xbow.com/Products/Wireless_Sensor_Networks.htm“, 2004.
[8] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee, R. Morris: “Capacity of Ad Hoc Wireless Networks, Proc. of MobiCom, 2001.
[9] J. Blumenthal, M. Handy, F. Golatowski, M. Haase, and D. Timmermann: Wireless Sensor Networks - New Challenges in Software Engineering, Proceedings of 9th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA), Lisbon, Portugal, 2003.
[10] M. Handy, J. Blumenthal, D. Timmermann: “Energy-Efficient Data Collection for Bluetooth-Based Sensor Networks”, IEEE Instrumentation and Measurement Technology Conference, Como, Italy, 2004.