A Classification of the Debugging Techniques of Wireless Sensor Networks

Sreedevi T.R
Dept. of Computer Science,
Rajagiri School of Engineering & Technology,
Kochi, India
sreedevi.prabin@gmail.com

Mary Priya Sebastian
Dept. of Computer Science,
Rajagiri School of Engineering & Technology,
Kochi, India
marypriyas@gmail.com

***Abstract*—Wireless sensor networks are meant for monitoring in remote and inaccessible locations where it is not feasible to install conventional wired infrastructure. But the health monitoring system of such networks is relatively intangible to network administrators, because of their complex and unpredictable behavior. In this paper a survey on debugging techniques of wireless sensor networks are presented. The debugging tools are divided on the basis of their usage in application life cycle as pre-deployment, post-deployment and deployment-time validation. Based on the implementation strategies, the debuggers are further classified as software-based, hardware-based and hybrid. A comparative study is made on the debugging tools and a summary is provided in Table I. This paper also discusses about the key characteristics required for implementing an efficient debugging tool for wireless sensor network.**

Keywords-classification; wireless sensor network; debugging; implementation strategy; simulator; deployment

I. INTRODUCTION

Wireless Sensor Networks (WSN) employs a large number of autonomous devices known as sensor nodes which are capable of sensing and sending environmental conditions to a gateway node that acts as an interface to the outside world. The development of Wireless Sensor Networks(WSN) is highly appreciated due to its monitoring capabilities in the remote areas and in human-unfriendly locations where a conventional wired infrastructure is found to have less use [2]. The purpose of a WSN debugger is to iteratively detect and figure out the root causes of the failures which are difficult to identify with limited resources and visibility of the WSN network. The debugging tools that aid in fault detection and diagnosis systems for WSNs have been emerging out enormously out of the dissatisfaction with the current state of art. An excellent visualization of the various WSN debugging techniques are well covered in [1].

This paper mainly focuses on a survey conducted on the various implementation techniques adopted in different debuggers. The classification of the debugging tools as software, hardware and hybrid is done by analyzing the tool and by identifying the predominant implementation technique adopted in it. A tool is categorized as hybrid if the contribution of software and hardware in its implementation is of the same proportion. A discussion on the pros and cons of implementation strategies of WSN debuggers are summarized which in turn aids in developing an efficient debugger with promising characteristics.

The survey also highlights a new extension to the generalized classification of debuggers based on the usage in application life cycle. In [1] a detailed study on pre-deployment and post-deployment debugging tools is done. Here, an analysis on tools which identifies early network failures at the time of deployment is also discussed.

The rest of the paper is organized as follows. Section II provides a general classification of debugging tools along with the tools that falls under deployment-time validation category. Section III gives further classification of WSN debuggers based on the implementation techniques. A discussion on the comparative study done on the various implementation techniques and its summary is presented in Section IV. The paper is concluded in Section V.

II. GENERAL CLASSIFICATION ON DEBUGGING TECHNIQUES

In general the debuggers of WSN can be classified into two groups as: Pre-deployment debuggers and Post-deployment debuggers. Deployment-time validation is an extension to this classification.

A. Pre-deployment Tools

These tools are used in the concept development, initial testing and evaluation stage of an application life cycle [3]. The pre-deployment tools that are generally used for debugging WSNs are listed below.

1) Simulators: Wireless Sensor Networks (WSN) are formed by a large number of autonomous sensing nodes. An analytical modeling of a WSN usually leads to over simplified analysis with limited confidence. Besides, deploying testbeds supposes a huge effort. Therefore, simulation is essential to study WSN. However, simulation results rely on a particular scenario under study (environment), hardware and physical layer assumptions, which are not usually accurate enough to capture the complex behavior of a WSN. This will minimize the accuracy of simulation results. A good simulator should have a proper balance between scalability and accuracy factors of WSN. Simulation environment can be classified into general simulation packages and specific WSN frameworks [3].

The simulators which come under general simulation package are discussed here. NS-2 [4] is one of the most

popular non-specific network simulators, and supports a wide range of protocols in all layers. OMNET++ [5] provides a powerful GUI library for animation, tracing and debugging support. Its major drawback is the lack of available protocols in its library, compared to other simulators. JiST/SWANS [6] is a discrete event simulation framework that embeds the simulation engine in the Java byte code. GloMoSim [7] is a simulation environment for wireless networks built with Parsec. Most of the popular simulation tools are listed in [8].

Some of the simulators that belong to specific wsn simulation package are discussed below. EmStar [9] is a software framework to develop WSN applications on special platforms called microservers. Mannasim [10], an extension of ns2 is a wireless sensor network simulation environment. TOSSIM [11] is a bit-level discrete event simulator and emulator of TinyOS. SENS [12] is a discrete event simulator implemented in C++. SENS utilizes a simplified sensor model with three layers (application, network and physical) plus an additional combined environment and radio layer.

2) Emulators: As a networked embedded system, a WSN application involves sensor node hardware, its drivers, operating systems, and networking protocols. As a result, the performance of the WSN application depends on all of these factors in addition to its implementation. An emulator is a special type of simulator whose aims is to enable realistic performance evaluation for WSN applications. Emulation environment or emulators are good choice, in which WSN applications can be directly run for testing, debugging, and performance evaluation. Additionally, studies on the lower layers (e.g., hardware drivers, OS, and networking) as well as cross-layer techniques can also be done in this environment by plugging the target modules into the emulator. For example, emulators can compute the power of a particular sensor's hardware platform in wireless sensor networks.

ATEMU [19] at its core is a software emulator for AVR processor based systems such as the MICA2. ATEMU is intended to bridge the gap between actual sensor network deployments and sensor network simulations. MSPSim [20] [21] is a Java-based instruction level emulator of the MSP430 series microprocessor and emulation of some sensor networking platforms. MEADOWS [22] is a software framework for modeling, emulation, and analysis of data of wireless sensor networks. The JTAG ICE[23] is a complete tool for On-chip Debugging on all AVR 8-bit microcontrollers with the JTAG interface. The JTAG interface is a 4-wire Test Access Port (TAP) controller that is compliant with the IEEE 1149.1 standard. C)

3) Testbuds: Although reasonable tools exist for evaluating large sensor networks in simulation and emulation, only a real sensor network testbed can provide the realism exigent to understand resource limitations, communication loss, and energy constraints at scale. Testbeds are an environment that provides support to measure number of physical parameters in controlled and reliable environment. This environment contains the hardware, instrumentations, simulators, various software and other support elements needed to conduct a test. Generally, testbeds allow for rigorous, transparent and replicable testing.

Motelab [24] is a public testbed using MICA2 platforms, which allows users to upload executables and receive execution results via the Internet. MoteLab is a set of software tools for managing a testbed of ethernet-connected sensor network nodes. Kansei [25] is another testbed which employs XSM, MICA2, and Stargate platforms. To address high levels of testbed contention Intel Research Berkeley has developed Mirage [26] which applies microeconomic approaches to arbitrate among competing users.

B . Post-deployment Tools

These are the tools which are used for evaluation and parameter tuning in the field .The ongoing maintenance of the WSN can also be done with post-deployment tools. In the following, the pre-deployment tools that are generally used for debugging WSNs are listed.

1) Record and Replay: WSN sensing events are normally asynchronous and non-repeatable which will make it particularly difficult to statistically evaluate the performance of sensor network applications, realistic protocol comparison and parameter tuning. Hence, it is essential to have the capability to capture and replay sensing events, providing a basis not only for system. Using this technique data stream generated by a module will be logged during the occurrence of an environmental event, and regenerate the same data stream later, from the perspective of the modules that consume the data. This will make the environmental event repeating itself.

EnviroLog [27] is a distributed service which uses asynchronous event recording and replay. CoreSight Trace Macrocells [28] provides hardware cores that can be added on as peripherals to an ARM-based system-on-chip to produce a cycle-accurate trace of execution. Hardware designed to interface an OCDM to a host computer via the JTAG standard is often referred to as an In-Circuit Emulator (ICE) or In-Circuit Debugger (ICD), or more correctly, a JTAG adapter. Many ICE tools are available for the MSP430 processor family. An example is the open source GoodFET [29] tool.

2) Logging and Symptom Mining: Symptom mining technique can be used for uncovering bugs due to interactive complexity in networked sensing applications. Such bugs are not caused by a single faulty component, but rather result from complex and unexpected interactions between multiple non-faulty components. Moreover, the manifestations of these bugs are often not repeatable, making them particularly hard to find, as the particular sequence of events that invokes the bug may not be easy to

reconstruct. Because of the distributed nature of failure scenarios, this technique looks for sequences of events that may be responsible for faulty behavior, as opposed to localized bugs such as a bad pointer in a module.

DustMiner [30] is a symptom mining debugger for uncovering root causes of failures and performance anomalies in wireless sensor network applications in an automated way. Sentomist (Sensor application anatomist) [31] is a novel tool for identifying potential transient bugs in WSN applications. Transient bugs, are buggy logics that may only be triggered by some occasionally interleaved event procedures that bear implicit dependency. LiteOS [32] provides the required functionality to log kernel events on MicaZ platforms. Specifically, the kernel logs events including system calls, radio activities, context switches and so on.

3) Live Monitoring and Control: In a WSN users should monitor the network and respond to network state changes continuously because of the high reliability of network state on the physical environment in which the network is deployed. So it is desirable to combine the functionalities of sensor network monitoring and control in a single monitoring system.

Octopus [33] is a protocol-independent open-source tool developed in two languages, nesC [34], for the embedded application, and Java, for the user application, specifically for TinyOS Version 2. We can also remotely reprogram them in batches over the air, such as with Deluge [35]. MARWIS [36] is another example for a live monitoring and control system for heterogeneous Wireless Sensor Networks, which supports common management tasks such as visualization, monitoring, (re)configuration, updating and reprogramming.

4) Passive Network Analysis: In a WSN, network monitoring can be divided into active monitoring and passive monitoring. In active monitoring, a monitoring protocol needs to be deployed on the network protocol stack in the nodes to obtain detailed information of the network and node parameters. In passive monitoring, an additional monitoring network is deployed. The monitoring network sniffs the wireless channel and captures data packets transmitting in the air to monitor the network and analyze the network protocol. It does not consume any resources of the monitored network.

The hierarchical monitoring and visualization of the collected packet data at the central server in real time is supported in passive monitoring systems such as Pimoto [37].SNIF [38] is another monitoring system under this category. MIThril LiveNet [39],a flexible distributed mobile platform that can be deployed for a variety of proactive healthcare applications is another example for a passive network monitoring system.

5) Active Network Analysis: Passive monitoring has the following problems and challenges: (i) Monitoring traces may be incomplete. (ii)The monitoring traces only contain limited information. In active network analyzing methods relevant statistics for efficient root cause analysis is actively collected from the network. Below are few of the network analysis methods which come under this category.

Memento [40] and Sympathy [41] are examples for active network analyzers. Tools for sensor network management such as NUCLEUS [42] provide read/write access to various parameters of a sensor node that may be helpful to detect problems which requires active instrumentation of the sensor network. Passive distributed assertions (PDA)[43] is a mechanism, which allows a developer to assert certain properties of the distributed system state such as there should be at least one cluster head among the neighbors of each node.

6) Source level debugging: Source-level debuggers allow a developer to execute a program on statement or code block at a time and to watch the program as it is being executed and are called so because they operate on symbols and statements defined in the program's source code.

Clairvoyant [44] can operate on the deployment hardware and in the deployment environment, without requiring any additional hardware or wires. Source-level debuggers such as GDB and DBX [45] can be implemented through hardware means using an in-circuit emulator (ICE) such as the AVR JTAG ICE, which is an external piece of hardware that interfaces with GDB and controls firmware execution by physically attaching to the I/O pins on the MCU.

C. Deployment-time Tools

These are the tools which can perform the health monitoring when the network is set up for the first time in the real application field. In the following, the pre-deployment tools that are generally used for debugging WSNs are listed.

1)Deployment Time Validation: Deployment of a wireless sensor network (WSN) system is a critical step because theoretical models and assumptions are not realistic. Since these systems are often located in human unfriendly areas that are difficult to reach or even inaccessible for certain periods of time it is necessary to verify the functionality of the system at the time of the deployment, thus minimizing the expense of revisiting the site in the near future for re-deployment, maintenance, or repairs.

Deployment-support networks (DSNs) have been proposed as a novel tool for debugging in both deployment-time and post-deployment phase. In this DSN nodes are attached to WSN target devices and form an autonomous network. In [46] a deployment time validation framework SeeDTV that consists of techniques and procedures for WSN status assessment and verification is presented.

III. IMPLEMENTATION STRATEGIES

This section provides further classification of debugging techniques upon implementation strategies as: (1) software tools (2) hardware tools, and (3) hybrid tools. In addition, a comparative description of available strategies is provided.

A. Software Tools

Most of the simulators are software-based which has generally two parts: a framework which has a module for wsn debugging and a user friendly front-end .A good simulator will also provide a powerful GUI library for animation, tracing and debugging support. It can have a virtual clock if it is a time-driven simulator .A good simulator should also have network communication tools to simulate network management algorithms. NS-2, OMNET++, EmStar and J-Sim are examples of software-based simulators.

ATEMU is a software emulator for AVR processor based systems such as the MICA2. It has an AVR CPU emulation core which provides various device modules that can be attached to the CPU in the form of plug-in libraries, and are loaded at run time by the emulator. MSPSim and MEADOWS are other examples of software emulators. MSPSim has tools for monitoring stack, setting breakpoints, and profiling.

Most of the tools used for managing a testbed of ethernet-connected sensor network nodes are software-based. MoteLab consists of several different software components. The main pieces are (i) MySQL Database Backend : Stores data collected during experiments, information used to generate web content, and state driving testbed operation (ii)Web Interface : PHP-generated pages present a user interface for job creation, scheduling, and data collection, as well as an administrative interface to certain testbed control functionality (iii) DBLogger : Java data logger to collect and parse data generated by jobs running on the lab(iv)Job Daemon : Perl script run as a cron job to setup and tear down jobs.

The common software approach for record and replay technique is to allow the users to designate the modules that provide the data stream through user interfaces. During the record stage, all function calls issued by the log modules are logged into persistent storage devices such as a flash. During the replay stage, log modules are disabled. Instead, the previously recorded function calls can be issued at the right time and in the right sequence as recorded. Envirolog follows this approach by using a preprocessor.

Another technique which can be realized through software means is symptom mining technique. Sentomist introduces the notion of event-handling interval to systematically anatomize the long-term execution history of an event-driven WSN system into groups of intervals. It can then apply a customized outlier detection algorithm to quickly identify and rank abnormal intervals .Dustminer focuses on those patterns, correlating them with (infrequent) events that may have caused them, hence uncovering the true root of the problem.

For live monitoring and control of WSN one method is manually reprogramming the nodes individually, which is quite tedious and not scalable. The second method is remotely reprogramming them in batches over the air, such as with Deluge [47]. Octopus is also an example for such a tool where the users may only require quick plug-and-play monitoring of a physical environment, with the ability to support in-situ network reconfiguration. As a third approach a plethora of self-organizing sensor network methodologies has recently been proposed [48, 49]. Other techniques rely on distributed collection of more comprehensive state information of nodes beyond the local neighborhood, by piggybacking the state information of a source node in data packets and snooping on this information at intermediate forwarding nodes [50].The fourth approach is to rely, at least partially, on the base station, which already has a reasonably comprehensive view of the network state resulting from the convergence of all data packets to the base station. The fifth approach is a hybrid approach which appears to be the best solution, where sensor nodes manage their local configuration in a greedy distributed manner, and the base station manages network level parameters centrally. MARWIS follows this approach with its hierarchical architecture.

One of the software approaches for active network analysis is to incorporate active network data collection code in the nodes as well as the sink. This code will make the system to extract the network information which is otherwise not a part of the usual network flow. A good active network analyzer should distill out the important metrics, events and generic correlators that help find bugs quickly, and to transmit this data in ways that minimize energy consumption and probing effects. Sympathy introduces the idea of correlating seemingly unrelated events, and providing context for these events, in order to track down bugs and find their root causes. A distributed monitoring of a subset of well connected neighbors using a variance-bound based failure detector achieves the lowest rate of false positives, suitable for use in practice and this approach is used in Memento.

Source level debugging can be implemented using a software approach without requiring any additional hardware or wires. Furthermore, no modifications need to be made to the application's source code. Clairvoyant is an example.

B. Hardware Tools

Sensor-Network Asynchronous Processor (SNAP) [51], is an example for hardware-based simulator and is designed to be both a processor core for a sensor-network node and a component of a chip multiprocessor, the Network on a Chip (NoC), which will execute a novel sensor-network simulator.

TABLE I. Implementation Strategies of Debugging Techniques

Technique Used	Implementation Strategy	General Classification			Classification on Implementation Strategy		
		Pre-Deployment	Post-Deployment	Deployment-time	Hardware	Software	Hybrid
Deployment Time Validation	DSN	–	✓	✓	✓	–	–
	SeeDTV	–	–	✓	–	–	✓
Record and Replay	Envirolog	–	✓	–	–	✓	–
	Coresight TraceMacrocells, MSP430 GoodFET	–	✓	–	✓	–	–
	TOSHILT,Aveksha,Flash Box	–	✓	–	–	–	✓
Logging & Symptom Mining	Sentomist, Dustminer	–	✓	–	–	✓	–
Live Monitoring & Control	Octopus	–	✓	–	–	✓	–
	Marwis	–	✓	–	–	–	✓
Passive Network Analysis	SNIF,Livenet	–	✓	–	✓	–	–
	PIMOTO,PMSW	–	✓	–	–	✓	–
Active Network Analysis	Sympathy, Memento	✓	✓	–	–	✓	–
	PDA	✓	✓	–	–		✓
Source Level Debugging	Clairvoyant	–	✓	–	–	✓	–
	GDB using AVR JTAG ICE	✓	✓	–	✓	–	–
Testbuds	Motelab,Kansei,Emtos	✓	–	–	–	–	✓
Emulators	ATEMU	✓	–	–		✓	–
	JTAG ICE	✓	–	–	✓	–	–
Simulators	SNAP	✓	–	–	✓	–	–
	Mannasim, Tossim, SENS, Prowler	✓	–	–	–	✓	–

Because SNAP can be used to build both physical and simulated nodes, researchers studying sensor networks will be able to use a single software interface.

Emulators can also be implemented through hardware. The JTAG ICE is a complete tool for On-chip Debugging on all AVR 8-bit microcontrollers with the JTAG interface Atmel AVR devices have extended this functionality to include full Programming and On-chip Debugging support.

The Record and Replay debugging technique can also be implemented through hardware means. CoreSight TraceMacrocells is an example for real-time trace functionality which is used in ARM-based system-on-chip to produce a cycle-accurate trace of execution.

A hardware approach for passive network analysis is by using a deployment support network to overhear sensor network traffic. Using a second radio interface (Bluetooth can be employed), the monitoring data is delivered to a PC, which may control multiple monitoring nodes. The PC forwards the data using a TCP/IPnetwork, e.g. WLAN-based, to a central server. At this place, the standard network monitoring application Wireshark is used to receive, decode, and visualize the packet information. Pimoto is an example for such a passive network analyzer.

Source-level debuggers such as GDB and DBX can be implemented through hardware means. The processor's built-in hardware support cannot be used for WSNs because the 8-bit microcontroller units (MCUs) most commonly used on sensor nodes do not provide hardware support for software debuggers. Another approach is hardware emulation through an in-circuit emulator (ICE) such as the AVR JTAG ICE, which is an external piece of hardware that interfaces with GDB and controls firmware execution by physically attaching to the I/O pins on the MCU. ICEs are most commonly used for debugging embedded devices, but they require a physical wire and additional hardware for every node and thus do not scale with the number of nodes or the geographic size of a network. Furthermore, they cannot be used spontaneously because the hardware must be deployed in advance.

C. Hybrid Tools

Many hybrid hardware/software approaches are designed in record and replay technique to overcome the limited memory capacity of Flash [52]. A second MCU and flash memory are added to provide dedicated recording. The compiler is modified to insert additional code at each location in the program that needs to be recorded. Events are directly sent via UART or GPIO to the dedicated MCU.

Hybrid approaches can also be used for passive network analysis. Such tools consists of a deployment support network to overhear sensor network traffic, a packet decoder to access the contents of overheard packets, a data stream processor to analyze packet streams for problems, a decision tree to infer the state of each sensor node, and a user interface to display these states. SNIF is an example. The MIThril LiveNet is another passive network analyzer based on hybrid implementation strategy. To solve the problems of inadequate passive network analysis trace, two processing steps are proposed [53]. First, an online merging procedure combines the incomplete traces of various monitors into a single more complete trace. Next, an inference procedure based on finite state machine reconstructs packets that were not captured by any monitor and determines whether a packet was received by its destination.

One of the hybrid approaches for Source level debugging is by inserting into the program code some assertions like traditional assertions in C programs. To verify that such assertions holds, affected nodes would broadcast small amounts of additional information that can be overheard by a sniffer network. Analyzing the resulting message trace, one can detect failed distributed assertions, giving valuable hints about possible failure causes. Using a sniffer network to check distributed assertions minimizes interference with the sensor network. Firstly, no complex distributed protocols are needed in the sensor networks to check assertions. Secondly, as long as a node can send messages, failures in the sensor network do not affect the ability to check distributed assertions. Passive distributed assertions (PDA) are an example.

IV DISCUSSION

Majority of the WSN debuggers are targeted to handle only a subset of all possible failures in WSN. A generic form of debugger that handles all kinds of failures occurring in a wireless sensor network is hard to find in the current state of art. Source level debuggers can handle only node failures, likewise simulators are good enough for detecting link failures. Some debuggers deals with interaction failures which results from the interaction complexity between the nodes while others take care of transient errors. When designing a WSN debugger, the designer should have a clear idea about the intended application in order to decide on the data that need to be monitored for effectively analyzing the root cause of the failures. Due to the complex and resource constrained nature of WSN there is high probability for wide variety of failures to be present in the networking environment. Debugging these failures by designing a general debugger is a hard task to achieve. Therefore it is more advantageous to have debuggers capable of identifying specific failures in such situation.

In addition to the classification made in [1], debugging tools also falls under deployment time validation category. These tools are very imperative to verify the functionality of the system at the time of the deployment, thus lowering the risk of early failures. Coincidentally, the validation minimizes the expense of revisiting the site in the near future for re-deployment, maintenance, or repairs.

The classification is also done in detail on the implementation strategies of the WSN debuggers. The software approaches need the instrumentation of the application code and will utilize the limited bandwidth capacity of WSN. One of the characteristic of a good debugging tool is that the inspection mechanism should be uninterruptable in the case of application network failures. This can be realized better through hardware implementation strategy. This may limit the scalability factor of WSN and also incurs an additional overhead of maintaining hardware monitoring devices. An observation made here is that the software and hardware means has its own pros and cons. Since the software implementation is prone to high degree of application intrusiveness and hardware implementation is liable to high degree of scalability the decision to select an appropriate implementation strategy is highly dependent on the deployment environment and the nature of intended application. The classification of the debugging tools is summarized in Table I.

V CONCLUSION

The debugging of WSNs is a tedious task due to its complex and unpredictable behavior. In this paper a survey on the debugging techniques of wireless sensor networks are presented. The generalized classification of debugging techniques based on their usage in application life cycle is further extended as deployment time validation. The debuggers are also classified based on the implementation strategies and the pros and cons of each category are discussed. A summary of the classification is provided in Table I which will enable in implementing an efficient debugger for wireless sensor networks.

REFERENCES

[1] A. Schoofs, G.M.P. O'Hare, and A.G. Ruzzelli "Debugging Low-Power and Lossy Wireless Networks: A Survey" IEEE COMMUNICATIONS SURVEYS & TUTORIALS,Volume:PP,Issue:99,doi:10.1109/SURV.2011.021111.00098

[2] K. Langendoen, A. Baggio, and O. Visser., Murphy Loves Potatoes:Experiences from a Pilot Sensor Network Deployment in PrecisionAgriculture, In 14th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS), Rhodes, Greece, 2006

[3] S. Guo, Z. Zhong and T. He, "FIND: Faulty Node Detec-tion for Wireless Sensor Networks," *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, California, 2009, pp. 253-266.

[4] S. McCanne and S. Floyd. The ns network simulator. Available on the web from the following site: http://www.isi.edu/nsnam/ns/.

[5] http://en.wikipedia.org/wiki/Omnet%2B%2B, Description: an introduction of Omnet++ in wiki webpage.

[6] R. Barr, Z. J. Haas, R. van Renesse, "JiST: Embedding Simulation Time into a Virtual Machine." In *Proc. 5th EUROSIM Congress on Modeling and Simulation*, Paris,France, September 2004

[7] Global Mobile Information Systems Simulation Library(GloMoSim).[Online].Available:http://pcl.cs.ucla.edu/projects/glomosim/

[8] E. Egea-López, J. Vales-Alonso, A. S. Martínez-Sala, P. Pavón-Mariño, J. García-Haro:"Simulation Tools for Wireless Sensor Networks", Summer Simulation Multiconference - SPECTS 2005

[9] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. Emstar: a software environment for developing and deploying wireless sensor networks. In Proceedings of the 2004 USENIX Technical Conference, June 2004.

[10] http://www.mannasim.dcc.ufmg.br, Description: an introduction of Mannasim webpage.

[11] http://www.eecs.harvard.edu/~shnayder/ptossim/, Description: a webpage introduced PowerTOSSIM.J. Elson, S. Bien, N.

[12] S. Sundresh, W. Kim and G. Agha, "SENS: A Sensor, Environment and Network Simulator," Proceedings of the 37th Annual Symposium on Simulation, Virginia, USA, 2004, p. 221.

[13] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, and M. Karir, "ATEMU: A Fine-Grained Sensor Net-work Simulator," Proceedings

of the 1st Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, City, 7 October 2004, pp. 145-152.

[14] J. Eriksson, A. Dunkels, N. Finne, F. Österlind and T. Voigt, "Mspsim—An Extensible Simulator for Msp430- Equipped Sensor Boards," Proceedings of the European Conference on Wireless Sensor Networks, Poster/Demo session, The Netherlands, January 2007.

[15] A. K. Dwivedi1, O. P. Vyas2, '' An Exploratory Study of Experimental Tools for Wireless Sensor Networks, Wireless Sensor Networks,"Vol. 3 No. 7, 2011,PP. 215-240

[16] Q. Luo, L. M. Ni, B. He, et al., "MEADOWS: Modeling, Emulation, and Analysis of Data of Wireless Sensor Networks," Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004), Toronto, 2004.

[17] Atmel Corporation. Mature AVR JTAG ICE. http://www.atmel.com/dyn/products/tools- card.asp?tool-id=2737.

[18] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: A wireless sensor network testbed. In Proceedings of the Fourth International Conference on Information Processing in Sensor Networks (IPSN'05), Special Track on Platform Tools and Design Methods for Network Embedded Sensors (SPOTS), April 2005.

[19] Ohio State University, "Kansei: Sensor Testbed for At-Scale Experiments," Poster, 2nd International TinyOS Technology Exchange, Berkeley,

[20] Intel Research Berkeley, "Mirage: Microeconomic Resource Allocation for SensorNet Testbeds," http://mirage.berkeley.intel-research.net/.

[21] L. Luo, T. He, G. Zhou, L. Gu, T. F. Abdelzaher, and J. A. Stankovic, "Achieving repeatability of asynchronous events in wireless sensor networks with envirolog," in INFOCOM, apr. 2006.

[22] http://www.arm.com/products/system-ip/debug-trace/trace-macrocells-etm/index.php

[23] T. Goodspeed, "Goodfet." http://goodfet.sourceforge.net, May 2010.

[24] M. M. H. Khan, H. K. Le, H. Ahmadi, T. F. Abdelzaher, and J. Han, "Dustminer: Troubleshooting interactive complexity bugs in sensor networks," in Proc. of the ACM SENSYS, Nov.2008, pp. 99–112.

[25] Yangfan Zhou, Xinyu Che,; Lyu, M.R., Jiangchuan Liu: Sentomist: Unveiling Transient Sensor Network Bugs via Symptom Mining.In Proceedings of the IEEE 30th International Conference on Distributed Computing Systems, 2010

[26] Q. Cao, T. Abdelzaher, J. Stankovic, and T. He. The liteos operating system: Towards unix-like abstractions for wireless sensor networks. In Proceedings of the Seventh International Conference on Information Processing in Sensor Networks(IPSN'08), April 2008.

[27] Raja Jurdak, Antonio G. Ruzzelli, Alessio Barbirato, Samuel Boivineau: Octopus: monitoring, visualization, and control of sensor networks,in Journal Wireless Communications & Mobile Computing

[28] D. Gay, P. Levis, R. von Behren, et al. "The nesC language: A holistic approach to networked embedded systems". ACM SIGPLAN Notices,38(5):1–11, 2003.

[29] TinyOS Network Programming J. Hui. http://www.cs.berkeley.edu/ jwhui/deluge/

[30] G. Wagenknecht, M. Anwander, T. Braun, T. Staub, J. Matheka, and S.Morgenthaler. "MARWIS: A Management Architecture for Heterogeneous Wireless Sensor Networks," In proc. 6th International Conference on Wired/Wireless Internet Communications 2008 (WWIC08), 2008.

[31] Falko Dressler, Rodrigo Nebel and Abdalkarim Awad, Distributed Passive Monitoring in Sensor Networks"

[32] M. Ringwald, K. Römer: SNIF: A Comprehensive Tool for Passive Inspection of Sensor Networks. GI/ITG Fachgespräch Sensornetze '07

[33] B. Chen, G. Peterson, G. Mainland, et al., "LiveNet: Using Passive Monitoring to Reconstruct Sensor Network Dynamics," Proceedings of the 4th International Conference on IEEE/ACM Distributed Computing in Sensor Systems, Santorini Island, June 2008

[34] S. Rost and H. Balakrishnan, "Memento: A Health Monitoring System for Wireless Sensor Networks," Proceed-ings of the 3rd IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Net-works (IEEE SECON), Reston, 2006.

[35] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In Proc. SenSys'05, 2005.

[36] G.Tolle and U. C. Berkeley, "Nucleus Network Man-agement System. "www.tinyos.net/ttx-02-2005/developments/Nucleus%20NMS.ppt

[37] Romer,K,Junyan Ma :PDA: Passive distributed assertions for sensor networks. Information Processing in Sensor Networks, 2009. IPSN 2009.

[38] J. Yang, M. L. Soffa, L. Selavo and K. Whitehouse, "Clairvoyant: A Comprehensive Source-Level Debugger for Wireless Sensor Networks," Proceedings of the 5th International Conference on Embedded Networked Sensor Systems, Sydney, 2007, pp. 189-203.

[39] The GDB developers. GDB: the GNU project debugger. http://sourceware.org/gdb.

[40] D. Jia, B. H. Krogh, and C. Wong. TOSHILT: Middleware for Hardware-in-the-loop Testing of Wireless Sensor Networks.http://www.ece.cmu.edu/ w˜ ebk/sensor-networks/toshilt/toshilt.html.

[41] TinyOS Network Programming J. Hui. http://www.cs.berkeley.edu/ jwhui/deluge/

[42] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie. "Protocols forSelf-Organization of a Wireless Sensor Network", IEEE Personal Communications, 2000.

[43] R. Winter, J. Schiller, N. Nikaein, and C. Bonnet. "CrossTalk: A Data Dissemination based Crosslayer Architecture for Mobile Ad-hoc Networks". In Proc. Applications and Services in Wireless Networks,2005.

[44] C. Kelly, V. Ekanayake, R. Manohar, "SNAP: A Sensor-Network Asynchronous Processor." In Proc. 9th Int. Symposium on Asynchronous Circuits and Systems,

[45] S. Choudhuri and T. Givargis, "Flashbox: a system for logging non-deterministic events in deployed embedded systems," in SAC Symposium on Applied Computing,ACM, 2009.

[46] Xianghua Xu; Chao Tong; Jian Wan: Improve the Completeness of Passive Monitoring Trace in Wireless Sensor Network. Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific