Demo Abstract: Passive Inspection of Deployed Sensor Networks with SNIF

Matthias Ringwald, Marc Cortesi, Kay Römer
Institute for Pervasive Computing
ETH Zurich, Switzerland
{mringwal,roemer}@inf.ethz.ch, mcortesi@student.ethz.ch

Andrea Vitaletti
Department of Informatics
University of Rome "La Sapienza", Italy
Email: andrea.vitaletti@dis.uniroma1.it

***Abstract*— We demonstrate a tool that allows inspection and debugging of deployed wireless sensor networks (WSN) by analyzing overheard radio messages. This tool can identify common problems such as node crashes, reboots, routing problems, and network partitions without instrumentation of sensor nodes.**

Existing approaches to identify performance problems and bugs in deployed WSN such as Sympathy [5] require to add logging and debugging code to the application running on the sensor nodes, with monitoring traffic being sent in-band with the sensor network data to the sink. These approaches result in significantly increased resource consumption in the WSN and bugs in the sensor network may also affect the monitoring mechanism.

The presented tool is an implementation of our Sensor Network Inspection Framework (SNIF) [6], which consists of hardware to overhear network traffic, a data stream framework to decode and analyze overheard messages, and a graphical user interface to display the network topology and node states. We will first provide an overview of the hard- and software before the demo setup is presented. We also briefly dicuss closely related work.

I. System Overview

The system consists of four components:

- A distributed network sniffer that can be configured to work with arbitrary radio configurations and MAC layers
- A packet description language to decode overheard messages
- A data stream framework to analyze decoded packets
- A graphical user interface to visualize network topology and node states

The network sniffer is based on a so-called deployment support network (DSN) [1] – an additional wireless ad hoc network that is temporarily installed alongside the inspected WSN during deployment. Our implementation is based on BTnode revision 3 [1], which provides two radio front-ends: a Zeevo ZV 4002 Bluetooth radio and a Chipcon CC1000 low-power radio. The Bluetooth radio is used to form a robust and high-bandwidth ad hoc network among the DSN nodes (a so-called Bluetooth scatternet), while the Chipcon radio is used to overhear sensor network traffic.

Using a configuration language, the Chipcon radio can be tuned to an arbitrary frequency and data rate within its specification. Our tool works with any sensor MAC protocol by locating a user-defined start-of-packet symbol and packet length indicator in the received byte stream. All overheard packets are forwarded across the DSN to a sink.

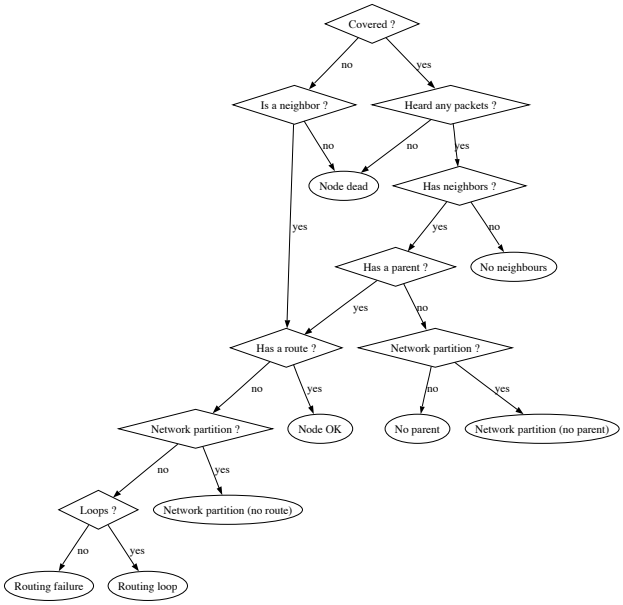

Fig. 1. Node state decision tree.

The packet description language is based on C syntax with some additional annotations, such that users can copy and paste packet descriptions from the source code of the sensor network application. Specifically, a packet is defined by a C struct that can contain other, nested structs. A packet decoder allows to access the contents of a field of an overheard message given the symbolic name of the field.

Overheard packets are then processed by a data stream framework to find indicators for specific bugs (e.g., a node reboot). Essentially, this framework allows to construct a directed graph of data stream operators. Overheard packets flow along the edges of the graph and are modified by the operators. Besides basic data stream operators such as aggregation over a time window, we provide a number of specific operators to deal with overheard packets. These operators have two purposes. Firstly, they deal with incomplete information resulting from message loss and from data that is not included in messages (e.g., sender address) but would be needed to identify bugs. Secondly, they find indicators for specific problems. For example, if no messages have been

received from a sensor node for a certain amount of time, this indicates that the node is dead. There exist a number of such operators to find indicators for different problems. Eventually, found indicators are evaluated in a binary decision tree to find the actual root cause of a problem. This binary decision tree is also implemented as a data stream operator. In the example tree depicted in Fig. 1, the leaf nodes represent different states of a sensor node, while inner nodes represent binary decisions which are taken based on the output of data stream operators. Please refer to [6] for details.

The graphical user interface of the debugger shows a graph of the sensor network topology as depicted in Fig. 2. Node colors indicate different classes of node states as computed by the binary decision tree (green: ok, yellow: warning, red: problem, gray: node cannot be properly observed). Edges between the nodes indicate network links, while the thickness of the edges indicates the amount of packets sent over the link during a specified time period. Nodes can be selected to display a textual summary of their states.

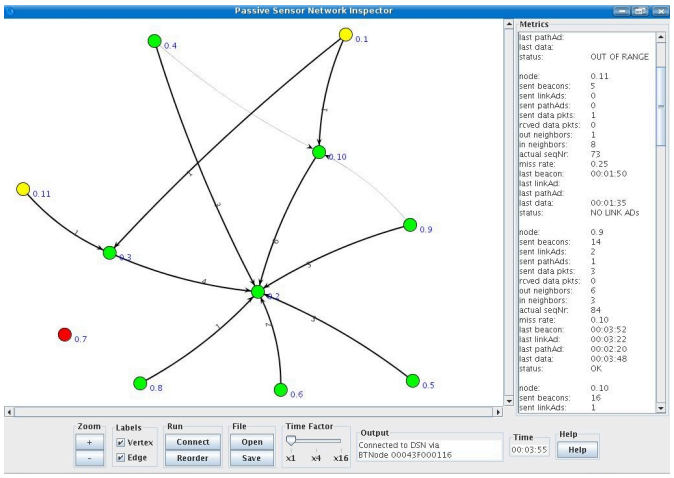

Fig. 2. GUI of the inspection tool. Nodes are colored to provide a quick glance on the network health. In the right box, node states are detailed.

II. Demo Setup

The demo setup consist of a large table with roughly 20 sensor nodes, about 3 DSN nodes, and one laptop computer executing SNIF and displaying the GUI. The transmit power of the sensor nodes is reduced to obtain a true multi-hop sensor network on top of a table. The DSN nodes will be marked so as to distinguish them from sensor nodes. They form a Bluetooth scatternet and send all overheard packets to the laptop.

The sensor network runs a typical multi-hop data gathering application based on the Extensible Sensing System (ESS) [3], where nodes send sensor readings along the edges of a routing tree to a sink. Essentially, sensor nodes broadcast beacon messages at regular intervals to allow neighbor discovery, link advertisement messages are sent to allow bidirectional link estimation, path announcement messages are sent to construct the routing tree, and data messages containing sensor readings are sent to the sink. Based on this protocol (which was not modified for inspection with SNIF), our demonstrator can detect the problems depicted in Fig. 1.

In the basic setup, the sensor network works as expected. To experiment with our inspection tool, a visitor can inject different types of faults into the sensor network: node reboots (pressing the reset button of a node), node failures (switching off a node), network partition (switching off several nodes), and routing failures (pressing a special button at a sensor node). These errors will then be detected by SNIF and displayed in the GUI.

III. Related work

Most closely related to SNIF is work on active debugging of sensor networks, notably Sympathy [5] and Memento [7]. However, both systems require instrumentation of sensor nodes and introduce monitoring protocols in-band with the actual sensor network traffic. Also, both tools only support a fixed set of problems, while SNIF provides an extensible framework.

Tools for sensor network management such as NUCLEUS [9] provide read/write access to various parameters of a sensor node that may be helpful to detect problems. However, this approach also requires active instrumentation of the sensor network.

Complementary to SNIF is work on simulators (e.g., SENS [8]), emulators (e.g., TOSSIM [4]), and testbeds (e.g., MoteLab [10]) as they support development and test of sensor networks *before* deployment in the field. EmStar [2] integrates simulation, emulation, and testbed concepts into a common framework.

IV. Acknowledgments

The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

References

[1] J. Beutel, M. Dyer, L. Meier, and L. Thiele. Scalable topology control for deployment-sensor networks. In *IPSN '05*.

[2] L. Girod, J. Elson, A. Cerpa, T. Stathapopoulos, N. Ramananthan, and D. Estrin. EmStar: A software environment for developing and deploying wireless sensor networks. In *USENIX 2004*.

[3] R. Guy, B. Greenstein, J. Hicks, R. Kapur, N. Ramanathan, T. Schoellhammer, T. Stathopoulos, K. Weeks, K. Chang, L. Girod, and D. Estrin. Experiences with the extensible sensing system ess. Technical Report 61, CENS, 2006.

[4] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Sensys 2003*.

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

[6] M. Ringwald, K. Römer, and A. Vialetti. Snif: Sensor network inspection framework. Technical Report 535, ETH Zurich, Zurich, Switzerland, 2006.

[7] S. Rost and H. Balakrishnan. Memento: A Health Monitoring System for Wireless Sensor Networks. In *SECON 2006*.

[8] S. Sundresh, W. Kim, and G. Agha. SENS: A Sensor, Environment and Network Simulator. In *Annual Simulation Symposium 2004*.

[9] G. Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. In *EWSN 2005*.

[10] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: a wireless sensor network testbed. In *IPSN 2005*.