periodica polytechnica

Electrical Engineering
55/3-4 (2011) 139–142
doi: 10.3311/pp.ee.2011-3-4.07
web: http://www.pp.bme.hu/ee
© Periodica Polytechnica 2011

RESEARCH ARTICLE

Radiocommunication test suite for wireless sensor networks

Krisztián Veress / *Miklós* Maróti

Received 2011-11-11

Abstract

In wireless sensor networks there is a great need for a test environment where multiple network topologies and communication protocols can be easily described, tested and evaluated under different circumstances in a controlled and reproducible way. Such a testbed could reveal the effectiveness or the bottleneck of networks and protocols.

Our proposed framework is designed to perform unit tests focusing on the wireless communication by collecting specially designed statistical indices. It supports all WSN platforms, dynamic network topologies, multiple communication modes, Low-Power-Listening, etc.

Communication is tied to specific events such as timers, message sending and reception, or control messages. During unit tests, simple messages with unique payloads are transmitted and statistics are collected about the communication in progress. Model verification is also incorporated to ensure data consistency. Since statistics are collected per edge, there is a possibility to analyze the network or a part of it.

Keywords

wireless sensor network · radiocommunication · testing · performance optimization

Krisztián Veress
Department of Computer Science, University of Szeged, H-6720 Szeged, Hungary

Miklós Maróti
Department of Algebra and Number Theory, University of Szeged, H-6720 Szeged, Hungary

1 Problems addressed

When developing wireless sensor network applications, radio drivers, components or designing communication protocols, one should have enough knowledge of the parameters of the physical network and underlying components on which the application will be tested and run. These parameters could be the physical topology [8], the prospective channel noise, expected power consumption [1], and the estimated message density.

If there is a need to fine-tune a given software component, improve performance or decrease resource utilization a real testbed environment is a must. However it is a real challange to ensure the same conditions for multiple sequential tests since these networks are distributed, event-driven and are massively influenced by the surrounding environment.

Our goal was to design and implement a test suite that can be run on real hardware, uses those drivers, layers and components that real applications do (similar to Twist [4]), and simulates a certain communication policy by carrying out reproducible tests and collecting valuable information about the environment and about the communication in progress.

This kind of a framework could be a swiss multitool in ones hand, given that the effect of any predefined value's modification, component change or change in the application's internal code could be tested, measured and verified.

2 Test suite concepts

When designing our test framework we focused mainly on the simulation of an application's wireless communication, not simulating the application itself like TOSSIM [6], VIPTOS [2] and Avrora [9]. The tag cloud on our mind contained expressions like *dynamic network topologies, platform-independency, no need for reprogramming, testing on real hardware, wide range of support for communication modes, simple testcase definitions.*

The proposed test suite has to be deployed on numerous (2+) motes that are going to build up the requested network and simulate the desired communication. Controlling these motes is done via one BaseStation mote connected to a laptop or PC and a PC-based Java application capable of resetting and configuring the whole network and finally downloading the results collected during the testcases. Results are given either in raw (for quick tests) or XML + XSLT format (for automated bath tests).

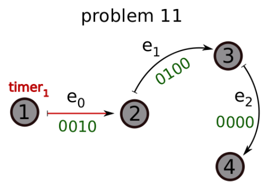
2.1 Modelling networks and communication

The model behind the scene is very simple, just a $G = (V, E)$ directed graph having V vertices as motes and an edge $(u, v) \in E$ is a communication line between mote u and v in such a way that messages transmitted by u are expected to be received by v. In this case u is the *sender* and v is the receiver on the (u, v) edge. Thus, atomic communication (messages) is tied to the edges of the directed graph.

The main idea behind the simulation is that the framework is going to send unique sequential messages on these communication links based on the applied communication policy. Then each and every node of the network track those messages that are tied to either an incoming or an outgoing edge of that node.

A communication policy has numerous global (network-wide), and local (per-edge) parameters. A policy describes *which mote* should send a message *to which mote(s)* on *what kind* of an event firing. Messages can be transmitted at the beginning of the test, based on timers or when messages are sent or received. Parameters include among others the communication mode (broadcasting, direct addressing, acknowledgements), Low-Power-Listening modes, timer frequencies, simulation time, etc.

A testcase in our test suite then consists of a network topology along with its communication policy. A sample testcase definition is given in the next example showing a multihop forwarding network using a timer in *mote 1* for sending periodic messages.



```
RT_PROBLEM_NEXT
 {1, 2, SEND_ON_TTICK , 1, 0x2 },
 {2, 3, SEND_AS_REQ   , 1, 0x4 },
 {3, 4, SEND_AS_REQ   , 1, 0 }
RT_NULL_EDGE
```

Fig. 1. Schematic and definition of a simple forward network

To avoid the need for reprogramming between tests, multiple testcases can be incorporated into the test suite. Then the PC application can select online the desired testcase to be run.

2.2 Statistics collection

The core of the framework is our statistics collection algorithm. As we have already stated above, our framework's purpose is to collect communication statistics while the established network is online. The main goal was to compile statistical figures that allow the user to deeply analyze the network's behaviour, and that are verifiable mathematically which is essential for foolproof results.

Since TinyOS allows us to track the state of any message sent or received by the radio chip, we have decided to count every occurences of any distinct communication event. Moreover, to get a completely clean map of the network's behaviour, every message sending attempt, success and failure is logged.

We have created two classes of statistics, the sender's, and those of the receiver. Actually, the statistics are collected per edge to let the implementation be as simple as it can, but eventually these numbers characterise the endpoint motes.

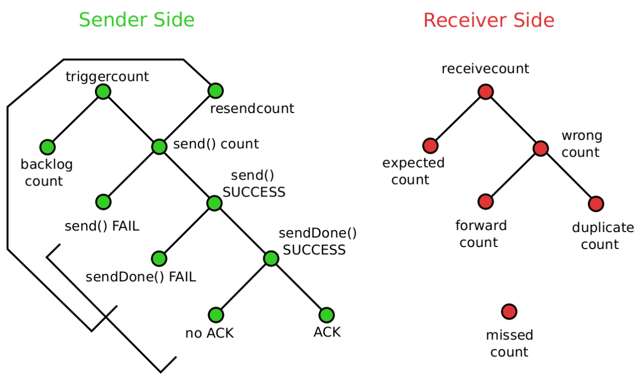


Fig. 2. Relation between the statistics at both ends

To ensure that our numbers are consistent, simple equations must hold between them. These are also shown in Fig. 2 by the hierarchical lines. Some examples:

$$\text{send}=\text{sendFail} + \text{SendSuccess}$$
$$\text{trigger}+\text{resend}=\text{backlog} + \text{send}$$
$$\text{resend}=\text{sendFail} + \text{sendDone} + \text{noAck}$$

The final report of a test run contains these enlisted statistics (and some extra) for each edge separately. Furthermore, to understand the receiver side statistics, let us consider an artificial sniffing of a single edge communication the result of which is shown in Figure 3.

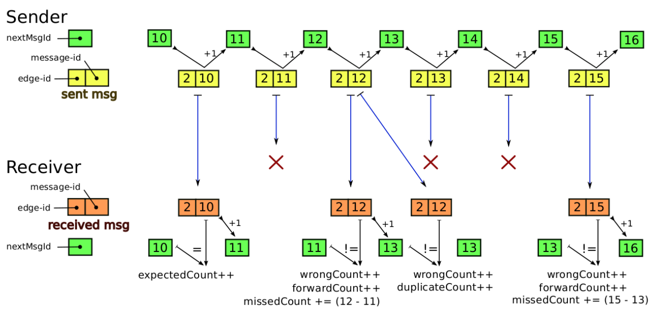


Fig. 3. Receiver statistics collection demonstrated

It can be clearly seen that each message consist of two simple values: an *edgeid* and an auto-incremented *msgid*. On both ends of an edge, the carefully maintained *nextMsgId* values must match, otherwise different communication failures are detected based on the next received message.

3 Preliminary results

We launched our framework and fed it with multiple testcases to demonstrate its capabilities.

3.1 Bandwidth measurements

Defining testcases with policies containing ’continously sending’ edges let us measure the network’s **maximal througput**. Using problem 4 (see Fig. 4), we could measure the effective speed of the Iris (18.17 kbps) and TelosA (14.33 kbps) platforms by evaluating the mean of results of multiple tests with packet size of 16 bytes (Fig. 5).

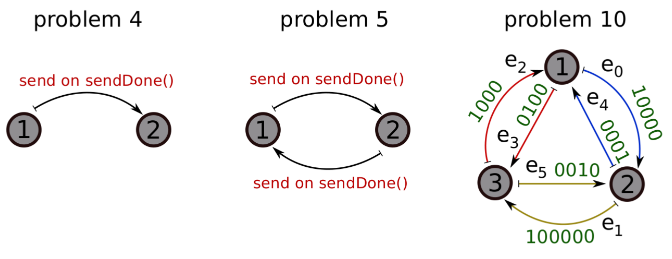


Fig. 4. Heavy wireless traffic problems

Tab. 1. non-LPL and LPL values for $tf = 50$ msecs

	edge 0		edge 1		edge 2		edge 3		edge 4	
wi(msecs)	0	10	0	10	0	10	0	10	0	10
trigger	19	19	19	19	19	19	19	19	38	
backlog	0	0	0	0	0	0	0	0	10	6
sendDone	19	19	19	19	19	19	19	19	28	32
receive	19	19	19	19	19	19	19	19	28	31

3.2 Parameter optimization

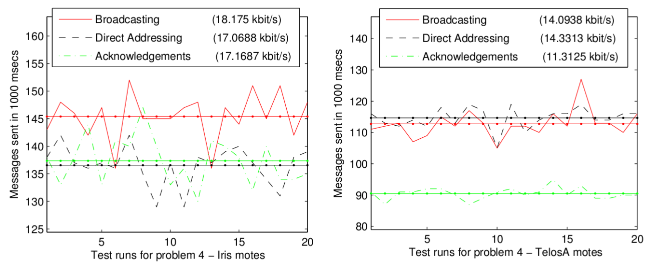


Fig. 5. Effective speeds of Iris and TelosA

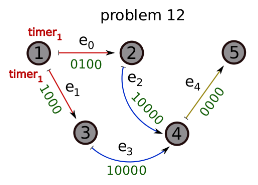
Furthermore, after changing backoff parameters [7, 10] of the random CSMA/CA algorithm in the RF230 radio chip’s driver, we measured 47.62 kbps on the Iris motes with the same configuration as before. This means a **162.1% performance increase** without introducing additional message loss and communication failures! These changes has also been incorporated into the new version of TinyOs.

Packet loss information can also be derived from the `missedCount` and `sendSuccessCount` statistical values to characterise the wireless channel and to help choose the appropriate communication mode.

3.3 Component verification

Furthermore we were able to use our tool with success for verifying the Low Power Listening [3, 5] components and radio stack layers. The LPL component must guarantee the proper message transmission while it duty cycles the radio chip to save power. If a message transmission request occurs in a particular mote, it wakes up and starts transmitting the message over and over again for at least *wakeup interval* time, which should guarantee in principle the reception of the message.

The best opportunity to analyze the behaviour of the LPL layer is to construct such networks where message transmission requests occur at known times and set the *wakeup interval* so that these two do not match.



```
RT_PROBLEM_NEXT
 {1, 2, SEND_ON_TTICK , 1, 0x4  },
 {1, 3, SEND_ON_TTICK , 1, 0x8  },
 {2, 4, SEND_AS_REQ   , 1, 0x10 },
 {3, 4, SEND_AS_REQ   , 1, 0x10 },
 {4, 5, SEND_AS_REQ   , 1, 0 }
RT_NULL_EDGE
```

Fig. 6.

As an example the results for the (*wakeup_interval* = *10msec*,*timer_freq* = *50msec*) pair are presented in table 1. It clearly shows that every message has been transmitted successfully either the LPL was enabled or disabled.

We have also been able to successfully debug acknowledgement related errors which occured when Iris and TelosB motes were used in LPL mode in a multi-hop network. The problem was originated from the fact that Iris is faster than TelosB in responding acknowledgement packets, so it did not wait long enough for acknowledgements from Telos motes.

The optimal value of the receive check interval of LPL was also measured with this framework.

References

1 **Aslam S, Farooq F, Sarwar S**, *Power consumption in wireless sensor networks*, FIT ’09: Proceedings of the 6th International Conference on Frontiers of Information Technology, posted on 2009, 1–9, DOI 10.1145/1838002.1838017, (to appear in print).

2 **Cheong E, Lee E A, Zhao Y**, *Viptos: a graphical development and simulation environment for tinyos-based wireless sensor networks*, SenSys ’05: Proceedings of the 3rd international conference on Embedded networked sensor systems, posted on 2005, 302–302, DOI 10.1145/1098918.1098967, (to appear in print).

3 **Moss K K D, Hui J**, *Low Power Listening. Core Working Group, tep 105 documentary for tinyos 2.x edition.*

4 **Handziski V., Köpke A, Willig A, Wolisz A**, *Twist: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks*, REALMAN ’06: Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality, posted on 2006, 63–70, DOI 10.1145/1132983.1132995, (to appear in print).

5 **Jurdak R, Baldi P, Videira Lopes C**, *Adaptive low power listening for wireless sensor networks*, IEEE Transactions on Mobile Computing **6** (2007), no. 8, 988–1004, DOI 10.1109/TMC.2007.1037.

6 **Levis P, Lee N, Welsh M, Culler D**, *Tossim: accurate and scalable simulation of entire tinyos applications*, SenSys ’03: Proceedings of the 1st international conference on Embedded networked sensor systems, posted on 2003, 126–137, DOI 10.1145/958491.958506, (to appear in print).

7 **Li Y, Long K P, Zhao W L, Yang F R**, *Rwbo(pd,w): a novel backoff algorithm for ieee 802.11 dcf*, J. Comput. Sci. Technol. **20** (2005), no. 2, 276–281, DOI 10.1007/s11390-005-0276-x.

8 **Santi P**, *Topology control in wireless ad hoc and sensor networks*, ACM Comput. Surv. **37** (2005), no. 2, 164–194, DOI org/10.1145/1089733.1089736.

9 **Titzer B L, Lee D K, Palsberg J**, *Avrora: scalable sensor network simulation with precise timing*, IPSN ’05: Proceedings of the 4th international symposium on Information processing in sensor networks, posted on 2005, 67, DOI 10.1109/IPSN.2005.1440978, (to appear in print).

10 **Yun L, Ke-Ping L, Wei-Liang Z, Qian-Bin C**, *A novel random backoff algorithm to enhance the performance of ieee 802.11 dcf*, Wirel. Pers. Commun. **36** (2006), no. 1, 29–44, DOI 10.1007/s11277-006-6176-8.