Title	Debugging low-power and lossy wireless networks : a survey
Authors(s)	Schoofs, Anthony; Ruzzelli, Antonio G.; O'Hare, G. M. P. (Greg M. P.)
Publication date	2011-03-24
Publication information	IEEE Communications Surveys & Tutorials, 14 (2): 311-321
Publisher	IEEE
Link to online version	http://dx.doi.org/10.1109/SURV.2011.021111.00098
Item record/more information	http://hdl.handle.net/10197/3547
Publisher's statement	Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Publisher's version (DOI)	10.1109/SURV.2011.021111.00098

Debugging Low-Power and Lossy Wireless Networks: A Survey

A. Schoofs, A.G. Ruzzelli, and G.M.P. O'Hare

***Abstract*—Recent economic and technical advances in wireless communication have allowed the deployment of low-power and lossy wireless networks—*LowPANs*, potentially comprised of a large number of nodes to serve new types of applications. However, the resource-constrained nature of microsensor platforms together with the unreliability and low-bandwidth of low-power and lossy wireless links have increased the risk and occurrence of network failures. Unlike with traditional wireless networks and controlled pre-deployment simulations and laboratory setups, likely events such as node crashes, inefficient networking and environmental interferences can potentially freeze a network post deployment. A survey of existing tools and related work in debugging LowPANs is presented, to provide a comprehensive state of the art of debugging tools and techniques. We divide debugging tools in two categories, *pre-deployment* tools and *post-deployment* tools, and evaluate their performance and limitations. From this study, we discuss the challenges in debugging LowPANs, providing the main issues and requirements that LowPANs' specific constraints impose on debugging tools, to help developers choose the appropriate tool for specific needs.**

***Index Terms*—lowpans, wireless sensor networks, debugging, survey.**

I. Introduction

LOW-POWER and lossy wireless networks (henceforth *LowPANs*) are a type of wireless networks with specific constraints emanating from the resource-constrained nature of microsensor platforms. Applications of such networks range from environment monitoring (e.g., wireless sensor networks [1]), to consumer electronics applications such as remote control systems for home automation and entertainment systems [2], to collections of wireless health monitoring devices worn on one's person, for monitoring vital signs such as heart rates [3], to wireless lighting control for intelligent building energy management [4].

Typically, nodes within such applications may be characterised as nodes meant to be disposable and scattered unobtrusively in large numbers; small-scale and low-price are subsequently key requirements. These objectives constrain the available energy, memory and processing power of the nodes. Moreover, the unreliability of the low-power wireless link and varying network properties introduce risks of failure at run-time. This causes a trade-off between designing both hardware and software constructs that optimise the use of the nodes' scant resources, and in parallel using such resources to achieve maximum efficiency in terms of network's resilience and management of potential faults.

Experience has shown that networks deployed in real environments suffer from the trade-off between energy efficiency and resiliency, and have shown poorer performance and different behavior than pre-deployment testing realised in controlled laboratory environments [5], [6], [7], [8]. For reasons including, but not limited to hardware and software failures, environmental interferers, and a possibly not optimised setting of the multitude of software system parameters, networks have experienced local and global failure when deployed.

Figure 1 depicts as a form of tree the relationship between a depreciated network performance and the potential root-causes of failure. Such tree references failures observed in real word deployments, and can be used to trace back to a set of possible root-causes of failure. For example, a link failure may find its origin in the number of buffers allocated on a node for storing packets. A large pool of static buffers or a high number of allowed dynamic buffer allocation may result in the acceptance of many packets by the node and the impossibility of processing them all, leading to congestion and potential packet dropping. Limiting the number of available buffers may in contrast lead to high packet rejection, thus forcing nodes to retransmit several times and eventually find other routes, leading to high control cost, additional traffic, and possibly sub-optimal routes.

Other deployments showed networks that operate as required by the application, at expenses of few overloaded and energy consuming nodes. This is likely to threaten the long term performance of those nodes and the capability of the network to match application requirements over time. Hence, there is a need for instrumentation tools that can provide network insights and debug network failures in real-time, as well as perform the necessary inspections to debug prospective long-term problems. Ensuring a long LowPAN lifetime is fundamental to achieving widespread deployment of *commercial* LowPANs, and to closing the gap with *research* deployments less critical to failure.

Debugging LowPANs derives its complexity primarily from the difficulty to both rely upon and use the limited resources available within the network. Moreover, the large number and heterogeneous nature of the nodes add scalability and flexibility requirements to the instrumentation tools. In the main, such constraints are not generally considered by traditional debugging tools. Systems for building automation combining lighting controls, HVAC equipment, IT devices and wireless sensors are an example where the global network would be comprised of numerous heterogeneous devices and networks, generating stress on the network's performance. In such context, debugging an occurring failure without appropriate tools may be lengthy and expensive, at the risk of also altering further the network performance, if inappropriate use of network resources is made.

A. Schoofs, A.G. Ruzzelli and G.M.P. O'Hare are members of CLARITY: Centre for Sensor Web Technologies, University College Dublin, Dublin, Ireland e-mail: {anthony.schoofs, ruzzelli, gregory.ohare}@ucd.ie.

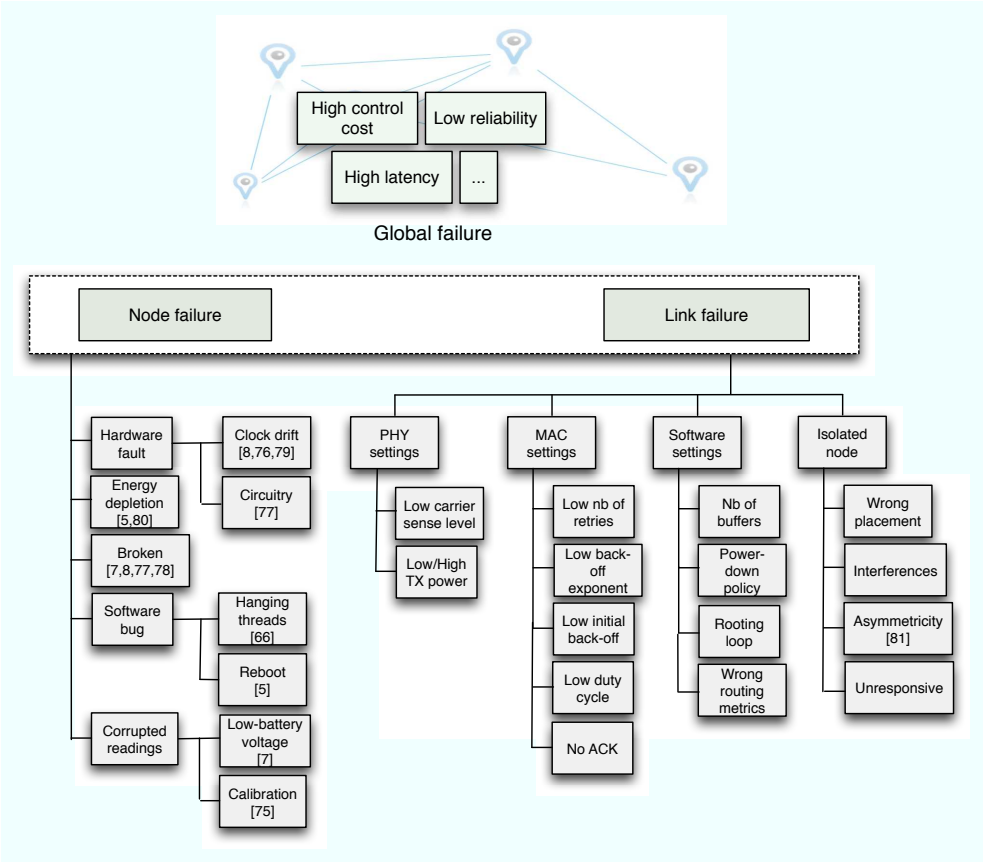

Fig. 1. Observed high-level failure and tree for discovering the root-cause of failures.

This survey evaluates the performance of existing tools for debugging LowPANs in respect to actual predominant problems with LowPANs deployments. The insights from this work are relevant to more than mere wireless sensor networks, as they address primary issues encountered with any deployment of low-power and lossy wireless networks. Section II and III provide a state of the art of instrumentation tools and techniques for debugging LowPANs. We divide debugging tools in two categories: (1) pre-deployment tools, and (2) post-deployment tools, highlighting pros and cons for each category. Section IV discusses the current limitations of debugging tools, outlining design guidelines for future LowPAN debugging tools. Finally, Section V concludes.

II. Pre-Deployment Tools

In the following, we analyse tools that are generally used for debugging LowPANs prior to deployment.

A. Software debuggers

Software debuggers allow control over a program execution, via breakpoints and access to memory state. Developers typically use development board versions of the LowPAN nodes, allowing firmware testing via Joint Test Action Group (JTAG) interfacing. For instance, the SoftBaugh MRF1611CC1100 module [9] exposes a JTAG interface and pins for debugging programs running within a TI MSP430 [10]. Once debugged, the code is moved to the tiny version of the nodes for deployment. The execution of code on a development kit might however be different than on the node itself; different interfaces, sensors, power source, and other hardware discrepancies may influence the code execution path and latency.

Furthermore, unpredictable events generate code paths that cannot be easily reproduced in software debugger environments. An approach consists of exploring the exhaustive states that a sensor network can find itself in, as done with T-Check [74] for applications developed within TinyOS, and with KleeNet [75] for applications developed within Contiki. Such techniques allow detection of bugs that would generally appear solely after deployment. Memory resources and computational power may however generate silent problems such as memory corruption only observable at run-time.

B. Software simulators

In contrast to debuggers which focus on program execution, simulation tools simulate key characteristics of LowPANs. They are able to predict a network's behavior at different levels over time without setting up a real deployment. Developers first develop and implement the network protocols in the simulator's language, then run the simulation and eventually analyse the network simulated performance. They duplicate the functions of one system using a different system, so that the second system behaves like—and appears to be—the first

TABLE I
ERRONEOUS AXIOMS OF WIRELESS-NETWORK RESEARCH, SOURCE: KOTZ *et al.* [17].

Axiom 0	The world is flat
Axiom 1	A radio's transmission area is circular
Axiom 2	All radios have equal range
Axiom 3	If I can hear you, you can hear me
Axiom 4	If I can hear you at all, I can hear you perfectly
Axiom 5	Signal strength is a simple function of distance

system. A number of exemplars exist [11], [12], [13], [14], [15], [16].

Due to the combinatorial explosion resulting from the large number of parameters and possible parameter values when considering testing a LowPAN deployment, realising an exhaustive testing of all possible parameter value settings is generally not feasible. Simulators are therefore important tools to rapidly test the wide range of application and networking scenarios that may occur at run-time. However, a simulator model of a real-world system is necessarily a simplification of the real-world system itself. Many assumptions are made on the environment and unexpected failures are not simulated. Erroneous axioms of LowPAN research and simulation, Table I, denote the primary reasons why simulators are not at this point able to match real deployment performance evaluation [17].

C. Software emulators

Emulators are able to give real time verification of the developed code, by connecting a hardware processor or a node to a software development environment and executing programs upon it. EmStar [18], VMNet [19], ATEMU [20], SENSE [21] are exemplars. This focus on exact reproduction of external behavior is in contrast with the adoption of an abstract model of the system being simulated.

LowPAN emulators address the issue of simulators' assumptions by incorporating within the simulation process real environmental parameters. Factors such as external interferers are taken into account and the system reflects better the conditions and environment wherein nodes will be deployed. However, LowPAN emulators are limited by the small coverage of nodes hooked to the software emulator. The environmental conditions in a laboratory room will not be similar to the ones that will be experienced in the remote locations of a real deployment. Furthermore, the data provided by the nodes is limited to a number of physical parameters, and network protocols are simulated in software.

D. Testbeds

LowPAN testbeds are used for the real time analysis and evaluation of sensor network applications. They are generally used to support network and middleware research efforts, and many of them are generic to support a wide range of applications. Sensor network testbeds, such as the ones presented therein, provide means for controlled experimentation in an indoor or outdoor setting.

MoteLab [22] provides a public, permanent testbed of 190 Tmote sky nodes for development and testing of sensor network applications via an intuitive web-based interface. Registered users can upload executables, associate those executables with motes to create a job, and schedule the job to be run on MoteLab. During the job all messages and other data are logged to a database which is presented to the user upon job completion and then can be used for processing and visualisation. Prior application prototyping is therefore required using a simulation environment or with a few nodes. CitySense is an urban scale sensor network testbed that is being developed by researchers at Harvard University and BBN Technologies [24]. CitySense consists of 100 wireless sensors deployed across a city. Kansei [23] is a testbed of 210 Crossbow Extreme Scale Motes (XSM) hooked individually onto 210 Extreme Scale Stargates (XSS). The Stargates are connected using both wired and wireless ethernet. Kansei exports a web interface on which experiments can be scheduled and the results retrieved. TWIST [25] is another flexible and scalable testbed for indoor deployment, handling heterogeneous node platforms and supporting active power supply control of the nodes. The latter allows mimicking both nodes' death or addition of new nodes, enabling controlled observation of self-configuration algorithms within the network.

Testbeds provide an easy solution for real testing of LowPAN applications prior to real-deployments. The heterogeneous nature of testbeds enables testing in a variety of environments, and observed communication patterns, energy depletion of nodes, and other network specifics will be a good reflection of what developers should expect with real deployment. Nevertheless, two main limitations remain in testbed environments; both the network topology and the environmental conditions are different than that of the final real deployment.

III. POST-DEPLOYMENT TOOLS

In contrast to pre-deployment tools considered in the previous section we now review those tools that are used for debugging LowPANs post deployment. Post-deployment tools can be classified as *passive*, *active* and *opportune*, as depicted in Figure 2. *Passive* tools require no action on the part of the nodes and the network, and do not interfere with the network's operation. They rely on communication packets received at the network sinks or overheard within the network. *Active* and *Opportune* tools require nodes' code instrumentation to facilitate debugging. When the network user interacts with the nodes to request or retrieve information on the nodes' state

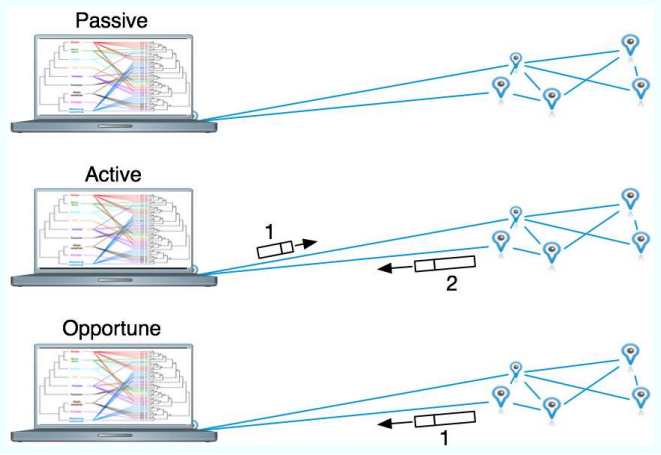

Fig. 2. Debugging tools are classified as passive, active and opportune, depending on the debugging tool's interaction model with the LowPAN.

or on the network operation, tools are said to be *active*. The interference of active tools on the network will depend on the network configuration, in respect to its size, traffic load, duty cycle and topology. When the nodes are instrumented such that they alone opportunely transmit nodes and network information to the network controller, tools are qualified as being *opportune*. Nodes themselves transmit information when appropriate e.g. by piggybacking it on an application message.

A. Centralised Debugging

Centralised approaches concentrate the activities of fault detection and particularly those regarding decision-making to a particular node. Such approaches generally assume central controllers or debugging devices with no constraints on processing power nor energy and with the capability to run complex fault management algorithms [26], and come in a variety of forms.

1) Passive packet analyzers: Network packet analyzers are computer software or computer hardware that can intercept and log traffic passing over a digital network [27]. They usually provide either graphical or command lines user interfaces, and are able to decode hundreds of protocols passively [28], [29], [30]. Live capture and off-line analysis are facilitated with coloring rules to sort types of packets and ease output exportation.

This centralised *and* passive capture of network communication is a core element of network debugging. The capture of packets terminating at sink nodes provides real-time information on nodes-to-sink communication, enabling discovery of potential issues such as packet loss and node faults. FIND [31] targets systems where the measured signal attenuates with distance. Nodes are ranked based on their sensing readings as well as their physical distances from the event. A node is considered faulty if there is a significant mismatch between the sensor data rank and the distance rank. Ding *et al*. [32] determine faulty nodes by checking whether the difference between a nodes reading and its neighbors is above a threshold.

Passive packet analyzers cannot intercept local and possibly remotely located communication, needed for debugging LowPANs of larger scale, such as for node-to-node communication. Furthermore, when multiple sink nodes exist in the network, correlating operations captured by the different devices is not automatic as packet analyzers are generally not synchronised. Additionally, traditional packet analyzers do not support the sequencing of multiple packets for detection of protocol specific operations e.g. LowPAN formation process; the user interface displays a packet level view on the network communication and does not provide this higher level view on the network's processes.

2) Active packet analyzers: Passive packet analyzers have no access to remote communication and nodes' internal state. In order to bypass these limitations, a more complete approach consists of recovering debug data via in-network communication between the network controller and nodes, providing hooks to the failures' origins [33], [34], [35], [36]. This approach allows the network controller to query nodes for specific data that might indicate root causes of failure. With Sympathy [33], a set of metrics are transmitted to the network controller for efficient failure detection. This approach has a minimal impact on the memory usage of the nodes but has yielded up to 30% of the network bandwidth with monitoring traffic. Hasan Khan *et al*. [37] propose to transmit power measurements taken locally on the node to determine the internal health condition of an unresponsive host and the most likely cause of its failure. The opportunity of piggybacking such metrics or other information into routing update messages has been explored [38]. Similar centralised active debugging approaches include Nucleus [35] and Memento [36]; they however do not prevent high interference with the network behavior as they use in-network messaging.

With MEGS [39], the global state of the network is recreated off-line, by accumulating all update messages from network nodes. The state of the network and the nodes is recreated on a separate PC and assertions can be made on detecting incoming issues or predicting the state of the network in a certain amount of time. The authors mention the use of a side channel for transmitting debug messages to reduce any interference with the network. Yet, this approach introduces some overhead in requiring the nodes to support channel hopping and extra energy for generating packets.

3) Logging and tracing: Data logging is the practice of recording sequential data. Tracing is a specialised use of logging to record information about a program's execution, in order to understand its behaviour. Since software tracing is low-level, the possible volume of trace messages may create noise for application software and alter its performance. Debugging with traces is time-consuming as the amount of unfiltered data logged after a program execution might be excessive. The possibility that additional code prone to bug is included in the nodes' software is also a threat to the resident software. Finally, the overhead in terms of execution latency and memory usage is not negligible and may affect the robustness and timeliness of the application execution.

Trace capabilities are often made available in operating systems for embedded systems. For instance, FreeRTOS [40] provides Trace hook macros that can be used to set a digital or analogue output to indicate which task is executing and allowing a logic analyzer or an oscilloscope to be used to view and record the task execution sequence and timing, or to log task execution sequences, task timing, kernel events and API calls for offline analysis or eventually integrate kernel events into third party debuggers [40]. Plugging measuring devices and accessing the logged data post-deployment is nevertheless not always feasible.

4) Traditional IP network utilities: A set of tools for debugging wired and wireless IP networks are traditionally used. These include:

- Port scanner - To probe network machines for discovering open ports and potential security threats;
- Bandwidth monitor - To track the number of bytes flowing in and out of a machine at any given time;
- Wireless networks manager - To discover nearby wireless networks of different types and indicate their signals strengths;
- Virtual machines - To control with a single machine several different computers and operating systems;
- Reachability tester - To test whether a machine is reachable;
- Path tracer - To trace the network path from a machine to another;
- Remote controller - To provide access to a command-line interface on a remote machine;
- Network statistics manager - To display network connections, routing tables, and network interface statistics.

Other higher level systems are able to reconstruct the state of a whole IP network and its inner communication. The different approaches used by those systems have been summarised elsewhere [41]. XTrace [42] attaches metadata to each packet that is updated by each processing node. LibLog [43] and similar systems log all information locally and the latter is retrieved later for off-line analysis. Pip [44] takes the approach to look at message paths at a single layer to reconstruct information about the distributed application. The approach of MagPIe [45] is also about tracing execution paths, using a schema to correlate events and create a total causal ordering. All the aforementioned traditional IP tools are centralised *and* active methods, may require the participation of the network user, and exert influence on the network behavior by using the available bandwidth, the nodes' energy for forwarding and receiving packets, and possibly reducing software control, robustness and efficiency by introducing debug software on the nodes.

Another drawback is that such tools require the network to run IP (Internet Protocol). Early work has shown the feasibility of IP-based sensor networks [46], [47]. The design of lightweight IP stacks [46], [47], [48] that fit the small memory of microsensor platforms has been realised. Ongoing standardisation efforts at the IETF [49] are specifying standard header compression [50], routing protocol [51] and other application layer [52] IP protocol modifications to significantly reduce the protocols' overhead and to integrate low-power networks as part of the Internet. With IP enabled, IP network utilities are then theoretically available to debug networks. Yet, many issues remain. First, the solutions are still limited to IP-based networks, then not applicable to non-IP LowPANs. Second, as LowPANs' debugging needs are different than that of less-constrained networks, the tools do not provide as such all the information needed to fully debug a LowPAN. Finally, low-power networks are characterised by their resource constraints, and debugging tools should provide a LowPAN-dedicated method for minimising its influence on those.

B. Distributed Debugging

Distributed debugging distributes fault detection over the nodes and provides opportunities for significantly reducing the traffic generated by centralised methods.

1) Network self diagnosis: Diagnosis and management of failures by the network itself can be realised via distributed network protocols and application support [53]. There is a trade-off between prolonging the network's lifetime and providing high quality fault management schemes [26]. Indeed, complex schemes often involve overhead communication that consumes extra energy, and software support that potentially introduces software bugs in addition to using memory. Many approaches tackled those challenges by minimising the overhead cost of exchanging messages with a network controller while maintaining effective diagnosis services. Local decisions by a single node or a group of nodes are often sufficient to derive whether a fault has occurred in the network; only then the information is communicated to the central network controller which takes further actions. However, those approaches are tightly coupled with specific failures. Harte *et al*. monitor the status of each sensor node to detect physical malfunctions [54]. Ritter *et al*. detect and distinguish network partition from node failure [55]. Others detect malicious nodes and faulty sensor readings [56], [57]. Those are not sufficient for debugging fully an entire network, and the combination of problem specific protocols to provide a complete network fault detection is overkill.

Fault management architectures enable versatile and global approaches to detecting network failures. Iwanicki *et al*. [53] argue that diagnosis failures requires extra application support on a selected group of nodes. Furthermore, they claim that networks will be run by specialised technicians, and therefore include them in the debugging process. When abnormal operation has been detected at the central controller, the first step consists in installing an application extension on the surrounding nodes via a remote call from the administrator. Those nodes aggregate, in a form of metrics corresponding to the problem to detect, information about what his happening in the considered area. The information is sent to the network controller, which can narrow down the root causes of failure. This approach addresses the scale and resources limitations of large-scale networks, while relying on stable and optimised mechanisms. Yet, the large diversity of failures that may happen in the network increases

the software support on the nodes, and both the associated overhead and effect on application behavior are not negligible.

2) Remote debuggers: Remote debugging consists in remote debugging from one computer to another. Beutel *et al*. have attached powerful nodes to sensor nodes with adapter cables for direct process monitoring [58]. This enables the retrieval of nodes' state information without packet exchange over the LowPAN. However, this technique is limited to networks of small size comprised of accessible nodes.

Enabling remote debugging on deployed nodes during network operation provides information on nodes' state. With Clairvoyant [59], Yang *et al*. have implemented and evaluated a post-deployment source-level debug support, located on the nodes, and executed from a remote controller. Source-level debuggers allow a developer to execute a program one statement at a time and watch the program as it is being executed. Remote debugging is used to debug a process on a target machine. A number of reasons make classical remote debuggers not fit for low-power wireless networks; software debuggers do not accomodate the multi-hop, resource-constrained, and time dependent nature of LowPANs [59]. To address this issue, Clairvoyant makes use of GDB GNU Project Debugger [60] as a child process and Trickle [61] to disseminate packets throughout the network. Clairvoyant modifies the target binary and does not change the source code neither requires extra hardware. No special effort from the developer before debugging a program is required: the source code does not need to be modified and no libraries need to be linked into the executable. The program execution is also not affected unless the developer issues a debugging command. Yet, the software support on the node requires 32 KB of Program memory and 1 KB of Data memory. Besides, a message dispatcher on the target node is required to separate application messages from debug messages. As a consequence, one cannot stop program at arbitrary points when an event happens, and uncertain delay in executing commands is introduced. Applications are also restricted to use a low-level radio stack similar to that of Clairvoyant. Finally, debug traffic interferes with the application communication.

3) Packet sniffers and visualisation software: Packet sniffers are devices that have the capability to *hear* something that was not meant for one's *ears*. Generally, overhearing is realised without the knowledge of the speakers. In the context of wireless communication, a packet sniffer captures transmitted packets and eventually decodes and analyzes their content according to the protocols header format specifications.

Commercially available packet sniffers often come with visualisation software. Daintree's Sensor Network Analyzer [62] includes hardware platforms for sniffing network packets, and a protocol decoder to drill down to packet, field, and byte level for many protocols (extendable to new protocols). Besides, it includes visualisation capabilities to view all network devices and interactions simultaneously. Intuitive tools to perform complex functions such as both multi-node and multi-channel capture, and commissioning are available. Other similar commercial systems exist [63], [64], [65], [66] and are widely used in the industry at deployment time. They often provide both passive and active debugging, to allow the installer generate protocol compliant frames that may trigger operations in the network e.g. network formation. These kits are very expensive but are essential tools to deploy wireless nodes in large numbers. They are mostly limited to debugging purpose, with a few of them used for performance evaluation of the network e.g. ZigBee-only performance evaluation and protocol compliancy with Daintree's analyzers.

4) Distributed packet sniffers: Debugging wireless networks in a distributed fashion *and* passively does not rely on nodes to detect faults locally. A first technique consists in overhearing the network communication with a parallel *capture* network comprised of sniffer devices distributed over the network, and generating as much understanding of the network operations from it, e.g. [67], [68]. Network topology, bandwidth usage, routing paths and other communication metrics can be reconstructed [68]. Capture of traffic over the air can be realised with typical low-power wireless nodes, by disabling address recognition and receiving all packets that the transceiver intercepts. Data processing and eventual data forwarding towards a central analyzer requires a robust data communication from the capturing device to the target device. A deployment support network (DSN) was proposed to capture the network communication, providing a Bluetooth radio front-end in addition to the low-power radio [67]. The time-synchronised capturing nodes form a robust Bluetooth scatternet with a laptop equipped with a Bluetooth radio serving as the system sink [58]; data is therefore communicated back to the sink with little network interference. Further work has shown how data analysis and filtering of network packets captured in-situ can benefit from an integration within Wireshark [28], providing a standard monitoring environment, showing packets as if they were captured at the network sink [69]. A second intrusive technique introduces additional debugging messages generated by the network nodes and captured by the sniffer devices. Those messages contain specific information such as a node's internal state, and brings extra information that cannot be captured with a completely passive method. This second technique affects the network's behavior but to a lower extent than active packet analysers, since debug messages are not forwarded back to the network sink but are captured locally by the overhearing system. Römer *et al*. [70] introduced Passive Distributed Assertions (PDA) to improve the discovery of root-cause of failures, which are generally identified via an access to the nodes' internal state. PDA requires small modifications to the nodes' software so that well-chosen information of a node's state is transmitted and made visible to the sniffing device. The approach is to derive a set of assumptions about the network under consideration and check whether the collected information conforms to the expected assertions. In order to reduce the negative influence of in-network messaging on the network's behavior, the method schedules transmissions when it minimises interference with the sensor network e.g. in idle periods or when neighboring nodes are off. In any case, packets are ignored when received by network nodes.

TABLE II
DEBUGGING TOOL FAILURE DETECTION: THE ACCURACY OF THE TOOL TO DETECT A PROBLEM IS INDICATED BY AN INCREASING NUMBER OF "+"

Debugging tools	**Node failure**	**Link failure**	**Global failure**
Pre-deployment			
Software debuggers	+++		
Software simulators		++	++
Software emulators	+	++	++
LowPAN testbeds	+	+++	+++
Post-deployment			
Passive packet analyzers	++	++	
Active packet analyzers	+++	+++	+++
Logging/Tracing	+++	+	
Network self-diagnosis	+	+	+
Remote debuggers	++	+	
Packet sniffers/Visualisation	++	++	+
Distributed packet sniffers	++	+++	+++

IV. DISCUSSION

Having reviewed in detail debugging tools for low-power and lossy wireless networks, we now consider how this informs the deployment of a LowPAN offering, and specifically what insights it affords as to appropriate debugging methodologies.

A. Tools and failure detection

Insight on pre-deployment tools general to embedded systems have been discussed [71]. We therefore reflect here on the efficacy of post-deployment techniques for detecting failures, providing insights specific to LowPANs. In order to gain an effective cross-comparison we utilise the failure topology introduced in Figure 1. Table II summarises the appropriateness of each class of debugging tool in the detection of different failure classes.

Post-deployments tools operate on a running network, providing insights on inter-node communication and externalising the *real* network performance. Due to the number of nodes and their potentially remote location, as well as the complexity of communication patterns, capturing every failure is inherently difficult. The common goal to all the techniques presented in this survey is to acquire the state of the nodes, and possibly create offline a global state of the network [39], which will likely exhibit the root cause of any failure. The capability of capturing local information on the network's performance increases the accuracy of the debugging, as local issues become apparent. Insights on communication patterns at node, link, and network level can be deduced. Furthermore, capturing events and communication patterns at a sink node or over the air in the LowPAN is useful when they can be correlated against time and process. Many protocols consist of a succession of protocol packets to be exchanged between nodes. For instance, routing protocols initiating route discoveries may transmit a route request packet and wait for a route reply from the destination node. A debugging tool should be able to correlate these two packets and conclude whether the route discovery process is valid. Only reporting a packet loss (e.g. unsuccessful route reply packet transmission) would be of less value. Furthermore, in this example, associating timing to such events would provide latency of packet transmissions and control over the compliancy of the protocol packet exchange. When more than one debugging device is used, time synchronization and an awareness of networking protocol specifications is beneficial into correlating captured events and network information.

Both centralised and distributed techniques leave fault diagnosis to be solved at a powerful node. This allows for reasoning on a larger set of data, and for the correlation of packets captured at different moments. The problem is with the retrieval of data from the network. Nodes' state cannot be accurately inferred with *passive* tools, and *active* and *opportune* tools have side-effects on the network. There is therefore a trade-off between the degree of debugging accuracy and detail that one wants to obtain and the degree of interference that the user is ready to apply to the network and the associated nodes. Early consideration of this trade-off is important in choosing the right debugging approach.

B. Tools and side-effects

A simple example depicts the implications of debugging LowPANs, as every action taken in the debugging process may have an impact on the network itself. Say congestion is occurring. Two possible approaches, amongst others, may be envisioned to detect congestion patterns and incidents within the LowPAN. The CODA receiver-based congestion detection technique [73] demonstrates how nodes can locally assess congestion. Nodes combine (a) the captured channel loading conditions and (b) their current internal buffer occupancy. This approach requires nodes to both listen to the frequency channel and introduce new packets into the network to inform other nodes about a diagnosed congestion. Even though CODA channel monitoring is activated when nodes are already set to receiving mode for receiving packets or while performing *Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)* to assess the frequency channel, nodes may need to keep their transceiver on for longer, and together with the additional transmission of packets subsequently use more energy. A second technique is an adaptation of the

CODA receiver-based congestion detection, where congestion is tentatively detected by passive overhearing of the frequency channel with a packet sniffer device. The combination of (a) the captured channel loading conditions and (b) the *estimated* current buffer occupancy of nodes, may provide comparable congestion detection without the side-effect on both nodes and network's behavior, at the expense of additional hardware.

Understanding the side-effects of each debugging technique helps inform the appropriate way to debug a specific network. Debugging approaches may be tailored ad hoc to a specific deployment, where side-effects of the chosen instrumentation tools are of a lesser concern. In other deployment, such similar approaches may reveal themselves as inappropriate. Table III highlights the drawbacks and side-effects of the tools investigated within this paper. Post-deployment debugging tools suffer from the constraints associated with low-power nodes on one hand, and the low-rate and unreliability of the wireless links on the other hand. Many existing debugging tools are directly applicable to LowPANs; yet they interfere more significantly upon LowPANs than their traditional cousins upon less constrained wireless networks. In-network messaging and nodes' code instrumentation are likely to impact on the network behavior.

By trying to observe a network failure, a debugging tool may conjure it away, or generate phenomena in the network that would not exist without the instrumentation tool's presence. More than the effects on the network's communication patterns, debugging often requires pieces of software code to be modified, leading to similar *vanishing bug* issues. Two types of *probe effects* that occur when studying a faulty system:

- **Introduction of software bugs.** *Heisenbugs* are software bugs that disappear when an attempt is made to observe them. For example, using a not-optimised compiler for generating a debug-mode version of a program may not externalise a bug that would appear with an optimised compiler. Another example is a bug caused by a race condition, where separate processes or threads of execution depend on some shared state. The result of the process is unexpectedly and critically dependent on the sequence or timing of other events. Introducing debugging code in a software program may indeed change delays and timing execution of concurrent processes. *Bohrbugs* are bugs that are hard to find and which remain in the software during the operational phase. They usually appear under specific conditions. For example, overflow bugs are difficult to isolate and generally manifest at run-time.
- **Modification of network communication patterns.** Similar to *Heisenbugs*, the insertion of debugging messages in the network may engender new paths and delays for the ongoing network communication, resulting in the study of a non-functional or different network behavior. Active and opportune debugging of the network involves bandwidth usage; packets are injected and processed by nodes along the way towards their destination. By augmenting the bandwidth usage, packet reliability, latency and subsequent nodes' energy usage are affected. Packets injected in a faulty network might also not successfully provide useful information as they would experience the network failure the same as network packets.

C. Tools and robustness

Robustness relates in a debugging context to both the robustness of the debugging software/hardware tools and their influence on the robustness of the diagnosed system. On the software side, in addition to increasing memory usage, introducing debugging code on nodes to support remote debugging once the network is deployed reduces software control, as additional software increases the risk of bugs and may also reduce the stability of the system. With respect to debugging instrumentation tools, distributed debugging approaches usually consist of extra devices deployed in the LowPAN area, to capture the nodes' state [72] and the network communication [67]. A robust, long-range and high-speed communication medium for the debugging devices is necessary to handle large data transfer over long distance. Furthermore, such debugging devices should not experience similar problems as the LowPAN under investigation nor highly interfere with it. For centralised methods, devices running the debugging tools should identically be resilient to failure and have sufficient processing power.

D. Debugging as part of the development process

Leaving aside failure detection and issues of side-effects, early consideration of other technical aspects is required to decide on the appropriate post-deployment debugging tool. Because of the wide range of application scenarios, different network characteristics will drive different requirements. As we have seen, most techniques require nodes code instrumentation to support debugging. If such software support is not implemented as an integrated part of the system software, addition of such can prove problematic. Debug software support should be installed within the system software before the deployment, or integrated in a modified binary, as done in [59], or possibly transmitted to the nodes as a firmware upgrade. The latter may nevertheless be impossible due to the network failure. Another important consideration is that in commercial systems, software frameworks, databases and source codes are generally closed or protected, preventing code instrumentation. Nodes from different vendors with proprietary software may as well need to interoperate. In such scenario, centralised and distributed passive tools are more appropriate. When opting for tools that do not require code instrumentation, the user must know beforehand the debugging limitations of such tools, such as the difficulty to know accurately the state of the nodes at a given time. Another aspect that must be considered before deployment is the size of the network to be deployed, as it also influences the ability of a tool to fulfill its debugging task. Centralised methods do not scale well, because of the overhead generated by the traffic associated to the exchange of packets with the central entity. With large-scale deployments, one approach to debugging is to use centralised principles in a localised fashion, helping reduce debugging traffic. Other approaches are distributed packet overhearing [70].

TABLE III
THE SIDE-EFFECTS OF DEBUGGING TOOLS.

Debugging tools	In network messaging	Influence on application and process execution	RF access to node	Memory usage	Protocol dependency	Probe effect	Assumptions
Pre-deployment							
Software debuggers	No	No	No	No	No	Yes	Yes
Software simulators	No	No	No	No	Yes	No	Yes
Software emulators	No	No	No	No	Yes	No	Yes
LowPAN testbeds	No	No	No	No	No	No	Yes
Post-deployment							
Passive packet analyzers	No	No	No	No	Yes	No	No
Logging/Tracing	Maybe	Yes	Yes	Yes	No	No	No
IP network utilities	Yes	Yes	Yes	Yes	Yes	No	No
Network self-diagnosis	Yes	Yes	No	Yes	Yes	No	Maybe
Remote debuggers	Yes	Yes	Yes	Yes	Yes	Maybe	No
Active packet analyzers	Yes	Yes	Yes	Yes	Yes	Maybe	No
Packet sniffers/Visualisation	No	No	No	No	Yes	No	No
Distributed packet sniffers	No	No	No	No	Yes	No	No

E. Tools and future-proof

For emerging communication protocols, MAC and network protocols are typically implemented in software on the nodes, to enable adaptation to evolving de-facto or industry standards. Subsequently, the range of sensor technologies and new protocols that need to be supported will evolve over time and this introduces other requirements for the debugging tools' software architectures. Small changes in the implementation of one software component of the debugging tool should not lead to needs for significant reorganization of other components. Software modularity is advocated to facilitate adaptation to rapidly evolving protocols, as well as interoperability, to allow reuse, collaboration and combination of debug information by multiple tools in different debugging contexts.

F. Tools and long-term failures

Current approaches to post-deployment debugging target the detection of problems that occur in a LowPAN e.g. by measuring various metrics and analysing nodes' internal states. Such approaches focus on present or short-term fault detection, and do not provide a preview of potential faults that may occur in the longer-term, such as predicting energy depletion over a certain path. The domain of LowPAN debugging lacks such a tool capable of providing a best-effort prediction of long-term issues. Even though LowPAN communication patterns will regularly change due to environmental interference, the ability to predict post deployment how a *deployed* network may evolve over time is of utter value, as it would provide the network fitter the assurance that a deployed network has no major flaw. Such tools would not be able to infer in advance unexpected failures such as those described in this survey, but should be able to deduce points of pressure in the network that may cause failures over a certain time. Post-deployment *predictors* may base predictions on real measurements realised on a deployed network, such as measuring communication metrics with an auxiliary support network. Constructing the profile of a deployed LowPAN, in terms of various information including communication paths, network dependencies, average delays and reliability of packet transmission between communicating nodes, and using this information as input to a simulator to predict the network evolution is an important novel research direction.

V. CONCLUSIONS

The core issues with debugging low-power and lossy wireless networks are the resource-constrained nature of the node and the unreliability of the wireless link. The inherent nature of traditional debugging tools, whereby nodes are remotely accessed by a debugging controller, is inappropriate due to the interference in-network messaging imposes on the network. The challenges in debugging LowPANs is multi-faceted; it involves more than reducing the side-effects on the network. The capability to capture both local and global information to detect failures at node and link level to evaluate network performance metrics, as well as the ability to analyse the wide range of network protocols and to evolve over time to decode new protocols operations add complexity to the debugging tool. This paper offers the first comprehensive review of existing techniques for debugging LowPANs. It reflects the appropriateness of differing debugging options for LowPANs and offers heuristics as to the selection of appropriate classes of tools. Pre-deployment tools rapidly test the wide range of application and networking scenarios that may occur at run-time. They provide an inexpensive test method in a short amount of time, in a controlled and reproducible environment, thereby facilitating system validation. However, they are not able to predict what interferers will be present in a facility at a given location, frequency, and time. Post-deployment tools do not test the network under particular conditions, but detect and analyse occurring failures, and may also evaluate the current network's performance. The recovery of information from the network for efficient debugging will be dependent on the technique used and on the compromises the user is ready

to make. Retrieving actively accurate information on nodes' internal state will cost nodes' energy and bandwidth usage, thereby depreciating the network's lifetime. Retrieving passively accurate information on network local communication would require packet sniffers to be deployed in the network, thereby increasing hardware costs. Currently, post-deployment debugging techniques are detached from each other; they have different objectives and solve only a subset of problems for specific network protocols. There are opportunities for hybrid and more complete methods, whereby failures detection is tackled on all aspects.

VI. Acknowledgments

This work is supported by Science Foundation Ireland under grant 07/CE/I1147 and through the SFI-funded National Access Programme (NAP 252).

References

[1] G. J. Pottie and W. J. Kaiser, *Wireless Integrated Network Sensors*, In Communications of the ACM 43, no. 5, pp 51–58, 2000
[2] RF4CE, The Remote Control Standard for Consumer Electronics, www.zigbee.org/rf4ce
[3] Body Sensor Network (BSN) node, http://vip.doc.ic.ac.uk/bsn/index.php?article=926
[4] D. Delaney, G.M.P. O'Hare, and A.G. Ruzzelli, *Evaluation of Energy Efficiency in Lighting Systems using Sensor Networks*, in 1st ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings (BuildSys'09), Berkeley, CA, U.S.A., 2009
[5] K. Langendoen, A. Baggio, and O. Visser., *Murphy Loves Potatoes: Experiences from a Pilot Sensor Network Deployment in Precision Agriculture*, In 14th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS), Rhodes, Greece, 2006
[6] J. Tateson, C. Roadknight, A. Gonzalez, S. Fitz, N. Boyd, C. Vincent, and I. Marshall, *Real World Issues in Deploying a Wireless Sensor Network for Oceanography*, In Workshop on Real-World Wireless Sensor Networks (REALWSN'05), Stockholm, Sweden, 2005
[7] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong, *A Macroscope in the Redwoods*, In 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys'05), San Diego, CA, U.S.A., 2005
[8] G. Barrenetxea, F. Ingelrest, G. Schaefer and M. Vetterli, *The Hitchhiker's Guide to Successful Wireless Sensor Network Deployments*, In the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08), Raleigh, NC, U.S.A., 2008
[9] MRF1611CC1100 packet radio module for the MSP430, http://www.softbaugh.com/ProductPage.cfm?strPartNo=MRF1611CC1100
[10] Texas Instruments MSP430 16-bit Ultra-Low Power MCUs, http://www.ti.com/msp430
[11] OMNet++ Webpage, http://www.omnetpp.org
[12] Network Simulator 2 (NS-2) Web Page, http://nsnam.isi.edu/nsnam/index.php/Main_Page
[13] GloMoSim Webpage, http://pcl.cs.ucla.edu/projects/glomosim/
[14] WSNet Webpage, http://wsnet.gforge.inria.fr/index.html
[15] TOSSIM Webpage, http://www.eecs.berkeley.edu/ pal/research/tossim-.html
[16] A. Sobeih, W. P. Chen, J. C. Hou, L. C. Kung, N. Li, H. Lim, H. Y. Tyan, and H. Zhang, *J-Sim: A Simulation Environment for Wireless Sensor Networks*, In 38th Annual Simulation Symposium, pp. 175-187, San Diego, CA, U.S.A., 2005
[17] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan, and C. Elliott, *Experimental Evaluation of Wireless Simulation Assumptions*, In the ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM'04), Venice, Italy, 2004
[18] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin, *EmStar: a Software Environment for Developing and Deploying Wireless Sensor Networks*, USENIX'04 Annual Technical Conference, Boston, MA, U.S.A., 2004
[19] VMNet: A WSN Emulator for Application Performance Evaluation, http://www.cse.ust.hk/vmnet
[20] ATEMU - Sensor Network Emulator / Simulator / Debugger, http://www.hynet.umd.edu/research/atemu
[21] SENSE - Sensor Network Simulator and Emulator, http://www.ita.cs.rpi.edu/sense/index.html
[22] MoteLab Website, http://motelab.eecs.harvard.edu
[23] Kansei: Sensor Testbed for At-Scale Experiments, http://ceti.cse.ohio-state.edu/kansei
[24] CitySense - An Open, Urban-Scale Sensor Network Testbed, http://www.citysense.net
[25] V. Handziski, A. Kpke, A. Willig, and A. Wolisz, *TWIST: A Scalable and Reconfigurable Testbed for Wireless Indoor Experiments with Sensor Networks*, In 2nd ACM International Workshop on Multi-hop Ad Hoc Networks: from theory to reality 2006 (ACM REALMAN 2006), pp. 63-70, Florence, Italy, 2006
[26] M. Yu, H. Mokhtar, and M. Merabti, *Fault Management in Wireless Sensor Networks*, IEEE Wireless Communications, vol.14, no.6, pp.13–19, 2007
[27] K. J. Connolly, *Law of Internet Security and Privacy*, Aspen Publishers, pp. 131, 2003
[28] Wireshark website, http://www.wireshark.org
[29] Tcpdump website, http://www.tcpdump.org
[30] Microsoft Network Monitor, http://support.microsoft.com/kb/933741
[31] S. Guo, Z. Zhong, and T. He, *FIND: faulty node detection for wireless sensor networks*, In 7th ACM Conference on Embedded Networked Sensor Systems (Sensys'09), Berkeley, CA, U.S.A., 2009
[32] M. Ding, D. Chen, K. Xing, and X. Cheng, *Localized Fault-Tolerant Event Boundary Detection in Sensor Networks*, In 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'05), Miami, FL, U.S.A., 2005
[33] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, *Sympathy for the sensor network debugger*, In 3rd ACM Conf. on Embedded Networked Sensor Systems (SenSys'05), pages 255267, San Diego, CA, U.S.A., 2005
[34] R. Jurdak, A. G. Ruzzelli, A. Barbirato and S. Boivineau, *Octopus: Modular Visualization and Control for Sensor Networks*, Wiley Wireless Communications and Mobile Computing, 2009(9): 1-21
[35] G. Tolle and D. Culler, *Design of an Application-Cooperative Management System for Wireless Sensor Networks*, In 2nd European Conference on Wireless Sensor Networks (EWSN'05), Istanbul, 2005
[36] S. Rost and H. Balakrishnan, *Memento: A Health Monitoring System for Wireless Sensor Networks*, In 3rd Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON'06), Reston, VA, U.S.A., 2006
[37] M. M. Hasan Khan, H. K. Le, M. LeMay, P. Moinzadeh, L. Wang, Y. Yang, D. K. Noh, T. F. Abdelzaher, C. A. Gunter, J Han, and X. Jin, *Diagnostic powertracing for sensor node failure analysis*, In 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'10), Stockholm, Sweden, 2010
[38] J. Staddon, D. Balfanz, and G. Durfee, *Efficient Tracing of Failed Nodes in Sensor Networks*, In 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), Atlanta, GA, U.S.A., 2002
[39] M. Lodder, G.P. Halkes, and K.G. Langendoen, *A Global-State Perspective on Sensor Network Debugging*, In 5th Workshop on Embedded Networked Sensors (HotEmNets 2008), Charlottesville, VA, U.S.A., 2008
[40] R. Barry, FreeRTOS™, http://www.freertos.org
[41] A. Tavakoli, *Wringer: A Debugging and Monitoring Framework for Wireless Sensor Networks*, In 5th ACM Conference on Embedded Networked Sensor Systems (SenSys'07), Sydney, Australia, 2007
[42] R. Fonseca, G. Porter, R. Katz, S. Shenker and I. Stoica, *X-trace: A Pervasive Network Tracing Framework*, In 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI'07), pp. 271-284, Cambridge, MA, 2007
[43] D. Geels, G. Altekar, S. Shenker and I. Stoica, *Replay Debugging for Distributed Applications*, In USENIX'06 Annual Technical Conference, pp. 289–300, Boston, MA, U.S.A., 2006
[44] P. Reynolds, C. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah and A. Vahdat, *Pip: Detecting the Unexpected in Distributed Systems*, In 3rd Symposium on Networked Systems Design and Implementation (NSDI'06), San Jose, CA, U.S.A., 2006
[45] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat and R. A. F. Bhoedjang, *MagPIe: MPIs Collective Communication Operations for Clustered Wide Area Systems*, In Proceedings of the 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 34(8):131–140, ACM Press, 1999

[46] A. Dunkels, *Full TCP/IP for 8 Bit Architectures*, In Proceedings of the First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys 2003), San Francisco, CA, U.S.A., 2003
[47] Z. Shelby, J. Riihijärvi, O. Raivio, P. Mähönen and P.Huuskonen, *NanoIP: The Zen of Embedded Networking*, In 38th annual IEEE International Conference on Communications (ICC2003), Anchorage, AK, U.S.A., 2003
[48] M. Durvy, J. Abeille, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels, *Making sensor networks IPv6 ready*, In the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08), Raleigh, NC, U.S.A., 2008
[49] Internet Engineering Task Force (IETF) Home page, http://www.ietf.org
[50] IETF 6LoWPAN Working Group charter, http://www.ietf.org/html.charters/6lowpan-charter.html
[51] IETF ROLL Working Group charter, http://www.ietf.org/html.charters/roll-charter.html
[52] IETF CoRe Working Group charter, http://www.ietf.org/html.charters/core-charter.html
[53] K. Iwanicki and M. van Steen, *Towards a Versatile Problem Diagnosis Infrastructure for Large Wireless Sensor Networks*, In 2nd International Workshop on Pervasive Systems (PerSys'07), pp. 845–855, Villamoura, Portugal, 2007,
[54] S. Harte, A. Rahman, and K. M. Razeeb, *Fault Tolerance in Sensor Networks Using Self-diagnosing Sensor Nodes*, In IEEE International Workshop on Intelligent Environment (IE 05), pp 7–12, Essex, UK, 2005
[55] H. Ritter, R. Winter, J. Schiller, *A Partition Detection System for Mobile Ad-Hoc Networks*, In 1st IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'04), pp. 489-497, Santa Clara, CA, 2004,
[56] S. Ganeriwal and M. B. Srivastava, *Reputation-Based Framework for High Integrity Sensor Networks*, In 2nd ACM workshop on Security of ad hoc and sensor networks (SASN'04), Washington, DC, U.S.A., 2004
[57] M. Krasniewski, P. Varadharajan, B. Rabeler, S. Bagchi and Y. C. Hu, *TibFit: Trust Index Based Fault Tolerance for Arbitrary Data Faults in Sensor Networks*, In Proceedings of the International Conference on Dependable Systems and Networks (DSN'05), Yokohama, Japan, 2005
[58] J. Beutel, M. Dyer, L. Meier, and L. Thiele, *Scalable Topology Control for Deployment-Support Networks*, In 4th International Conference on Information Processing in Sensor Networks (IPSN'05), Los Angeles, CA, U.S.A., 2005
[59] J. Yang, M.L. Soffa, L. Selavo, and K. Whitehouse, *Clairvoyant: A Comprehensive Source-Level Debugger for Wireless Sensor Networks*, In 5th ACM Conference on Embedded Networked Sensor Systems (Sensys'07), Sydney, Australia, 2007
[60] GDB: The GNU Project Debugger, http://www.gnu.org/software/gdb
[61] P. Levis, N. Patel, D. Culler, and S. Shenker, *Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks*, In 1st Symposium on Networked Systems Design and Implementation (NSDI'04), pp. 15-28, San Francisco, CA, U.S.A., 2004
[62] Daintree's Sensor Network Analyzer Webpage, http://www.daintree.net/products/sna.php
[63] Frontline MeshDecoder Webpage, http://www.fte.com/products/MeshDecoder-01.asp
[64] ZENA Wireless Network Analyzer, http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en520682
[65] Whiznets ZigBee Network Analyzer, http://www.whiznets.com/index.php/Zigbee-Network-Analyser.html
[66] Perytons Analyzers, http://www.perytons.com/index.html
[67] M. Ringwald and K. Romer, *SNIF: A Comprehensive Tool for Passive Inspection of Sensor Networks*, In GI/ITG KuVS Fachgesprach Sensornetze, Aachen, Germany, 2007
[68] B. Chen, G. Peterson, G. Mainland and M. Welsh, *LiveNet: Using Passive Monitoring to Reconstruct Sensor Network Dynamics*, In 4th IEEE/ACM International Conference on Distributed Computing in Sensor Systems (DCOSS'08), Santorini Island, Greece, 2008
[69] F. Dressler, R. Nebel, and A. Awad, *Distributed Passive Monitoring in Sensor Networks*, In 26th IEEE Conference on Computer Communications (INFOCOM 2007), Demo Session, Anchorage, AK, U.S.A., 2007
[70] K. Romer and M. Ringwald, *Increasing the visibility of sensor networks with passive distributed assertions*, In Workshop on Real-World Wireless Sensor Networks (REALWSN'08), Glasgow, Scotland, 2008
[71] M. Mekni and B. Moulin, *A Survey on Sensor Webs Simulation Tools*, The Second International Conference on Sensor Technologies and Applications (SENSORCOMM'08), Cap Esterel, France, 2008
[72] M. Dyer, J. Beutel, L. Thiele, T. Kalt, P. Oehen, K. Martin, and P. Blum, *Deployment Support Network - a Toolkit for the Development of WSNs*, In 4th European conference on Wireless Sensor Networks (EWSN07), pages 19521, Delft, The Netherlands, 2007
[73] C. Y. Wan, S. B. Eisenman, and A. T. Campbell, *CODA: Congestion Detection and Avoidance in Sensor Networks*, In 1st ACM Conference on Embedded Networked Sensor Systems (SenSys'03), Los Angeles, CA, U.S.A., 2003
[74] P. Li and J. Regehr, *T-check: bug finding for sensor networks*, In 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'10), Stockholm, Sweden, 2010
[75] R. Sasnauskas, O. Landsiedel, M.H. Alizai, C. Weise, S. Kowalewski, and K. Wehrle, *KleeNet: Discovering Insidious Interaction Bugs in Wireless Sensor Networks Before Deployment*, In the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'10), Stockholm, Sweden, 2010
[76] P. Buonadonna, D. Gay, J. M. Hellerstein, W. Hong, and S. Madden, *TASK: Sensor Network in a Box*, In 2nd European Workshop on Wireless Sensor Networks (EWSN'05), Istanbul, Turkey, 2005
[77] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. *Wireless Sensor Networks for Habitat Monitoring*, In 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), Atlanta, GA, U.S.A., 2002
[78] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, *Lessons from a Sensor Network Expedition*, In 1st European Workshop on Wireless Sensor Networks (EWN'04), Berlin, Germany, 2004
[79] P. Chen et al., *Experiments in Instrumenting Wireless Sensor Networks for Real-Time Surveillance*, In International Conference on Robotics and Automation, Orlando, FL, U.S.A., 2006
[80] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees, *Deploying a Wireless Sensor Network on an Active Volcano*, In IEEE Internet Computing, 10(2):1825, 2006
[81] R. Szewcyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler, *An Analysis of a Large Scale Habitat Monitoring Application*, In 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04), Baltimore, MD, U.S.A., 2004
[82] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker, *Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks*, Technical Report CSD-TR 02-0013, UCLA, 2002