Deployment of Sensor Networks: Problems and Passive Inspection*

Matthias Ringwald, Kay Römer

[1]Institute for Pervasive Computing
ETH Zurich, Switzerland
{mringwal, roemer}@inf.ethz.ch

Abstract — *Deployment of sensor networks is concerned with setting up an operational wireless sensor network in a real-world setting. Unfortunately, deployment is a labor-intensive and cumbersome task as environmental influences often degrade performance or trigger bugs in the sensor network that could not be observed during lab tests. In this paper, we, firstly, study existing sensor networks to identify and classify typical problems that have been encountered during deployment. Secondly, we investigate whether and how the existence of these problems can be detected by means of* passive inspection*, where messages exchanged in the sensor network are overheard and analyzed such that modification of the sensor network is not required. We, thirdly, show how passive inspection can be implemented in a practical tool.*

1 Introduction

Sensor networks offer the ability to monitor real-world phenomena in detail and at large scale by embedding a wireless network of sensor nodes into the real world. Here, *deployment* is concerned with setting up an operational sensor network in a real-world environment. In many cases, deployment is a labor-intensive and cumbersome task as real-world influences trigger bugs or degrade performance in a way that has not been observed during pre-deployment testing in the lab [1, 2, 3, 4, 5, 6, 7, 8, 9]. The reason for this is that the real world has a strong influence on the function of a sensor network by controlling the output of sensors, by influencing the existence and quality of wireless communication links, and by putting physical strain on sensor nodes. These influences can only be very rudimentarily modeled in simulators and lab testbeds.

To track down problems that occur during deployment, the state of the sensor network and of individual nodes must be inspected to identify and eventually fix the cause of the problem. Lab testbeds typically provide a wired back-channel from each node (i.e., every sensor node is connected to a PC by cable) to support such inspection. However, such a

*The work presented in this paper was partially supported 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.

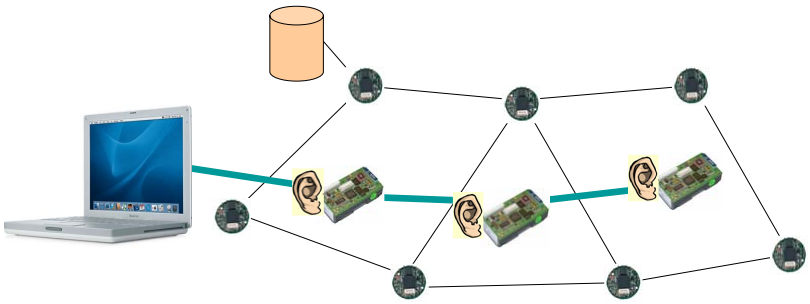

Figure 1: A deployment-support network (rectangular nodes) is a physical overlay network that overhears sensor network (round nodes) traffic and delivers it to a sink using a second radio.

wired back-channel from each node is generally not available in the field. Hence, current practice to inspect a deployed sensor network requires *active* instrumentation of sensor nodes with monitoring software and monitoring traffic is sent in-band with the sensor network traffic to the sink (e.g., [6, 10, 11]). Unfortunately, this approach has several limitations. Firstly, problems in the sensor network (e.g., partitions, message loss) also affect the monitoring mechanism, thus reducing the desired benefit. Secondly, scarce sensor network resources (energy, CPU cycles, memory, network bandwidth) are used for inspection. In Sympathy [6], for example, up to 30% of the network bandwidth is used for monitoring traffic. Thirdly, the monitoring infrastructure is tightly interwoven with the application. Hence, adding/removing instrumentation may change the application behavior in subtle ways, causing probe effects. Also, it is non-trivial to adopt the instrumentation mechanism to different applications. For example, [6, 10] assume a certain tree routing protocol being used by the application and reuse that protocol for delivering monitoring traffic.

To remove these limitations, we propose a *passive* approach for sensor network inspection by overhearing and analyzing sensor network traffic to infer the existence and location of typical problems encountered during deployment. To overhear network traffic, a so-called *deployment support network* (DSN) [12] is used: a wireless network that is temporarily installed alongside the actual sensor network during the deployment process as depicted in Fig. 1. Each DSN node provides two different radio front-ends. The first radio is used to overhear the traffic of the sensor network, while the second radio is used to form a robust and high-bandwidth network among the DSN nodes to reliably collect overheard packets[1]. The overheard stream of packets is then analyzed to infer and report problems soon after their occurrence. This approach removes the above limitations of active inspection: no instrumentation of sensor nodes is required, sensor network resources

[1]The DSN is temporarily installed alongside the actual sensor network and removed again as soon as the sensor network works as expected. Hence, the lifetime of the DSN is short and energy consumption is not a primary issue. Therefor, we can use Bluetooth to collect overheared messages, which has been designed as a cable replacement and uses techniques such as frequency hopping to minimize packet loss.

are not used. The inspection mechanism is completely separated from the application, can thus be more easily adopted to different applications, and can be added and removed without altering sensor network behavior.

We are developing a framework called SNIF (Sensor Network Inspection Framework) to support such passive inspection of sensor networks. The framework consists of a deployment support network, a flexible packet capturer and decoder that can be configured to support a wide variety of sensor network radio configurations and MAC protocols, a data stream framework to analyze packet streams for problems, and a graphical user interface to display found problems. Details can be found in a technical report [13].

As SNIF is a generic framework, customization is needed to inspect a specific sensor network application and to detect specific problems. For this, we first need to understand which problems can occur during deployment and how they can be detected by means of passive inspection. To this end, we have studied existing installations of sensor networks to identify and classify typical problems that occurred during deployment. We report our findings in Sect. 2. In Sect. 3 we discuss how these problems can be detected by means of passive inspection[2]. Finally, in Sect. 4 we show how passive detection of these problems is implemented in SNIF.

With this paper, we hope to contribute to a more systematic understanding and treatment of deployment problems, which is an important prerequisite for the development of tools that support a more efficient deployment process.

2 Deployment Problems

This section contains a classification of the problems typically found during deployment according to our own experience and as reported in the literature. Here, a problem is essentially defined as a behavior of a set of nodes that is not compliant with the specification.

We classify problems according to the number of nodes involved into four classes: *node problems* that involve only a single node, *link problems* that involve two neighboring nodes and the wireless link between them, *path problems* that involve three or more nodes and a multi-hop path formed by them, and *global problems* that are properties of the network as a whole.

2.1 Node Problems

A common node problem is *node death* due to energy depletion either caused by "normal" battery discharge or due to short circuits. In [14], a low-resistance path between the power supply terminals was created by water permeating a capacitive humidity sensor, resulting in early battery depletion and abnormally small or large sensor readings.

Low batteries often do not cause a fail-stop behavior of the sensor nodes. Rather, nodes may show Byzantine behavior at certain low battery voltages. As reported in [9], for example, *wrong sensor readings* have been observed at low battery voltage.

Software bugs often result in *node reboots*, for example, due to failure to restart the watchdog timer of the micro controller [15]. We also observed software bugs resulting in hanging or killed threads, such that only part of the sensor node software continued to

[2]Material in Sects. 2 and 3 is based on an earlier technical report [13] by the same authors.

operate.

Sink nodes act as gateways between a sensor network and the Internet. In many applications they store and forward data collected by the sensor network to a background infrastructure. Hence, problems affecting sink nodes must be promptly detected to limit the impact of data loss [4, 7, 9].

2.2 Link Problems

Field experiments (e.g., [16, 17]) demonstrated a very high variability of link quality both across time and space resulting in temporary link failures and variable amounts of *message loss*. *Network congestion* due to many concurrent transmission attempts is another source of message loss. In [7], for example, a median message loss of 30% is reported for a single-hop network. Excessive levels of congestion have been caused by accidental synchronization of transmissions by multiple senders, for example, due to inappropriate design of the MAC layer [18] or by repeated network floods [15]. If message loss is compensated for by retransmissions, a *high latency* may be observed until a message eventually arrives at the destination.

Most sensor network protocols require each node in the sensor network to discover and maintain a set of network neighbors (often implemented by broadcasting HELLO messages containing the sender address). A node with *no neighbors* presents a problem as it is isolated and cannot communicate. Also, *neighbor oscillation* is problematic [18], where a node experiences frequent changes of its set of neighbors.

A common issue in wireless communication are *asymmetric links*, where communication between a pair of nodes is only possible in one direction. In a field experiment [16] between 5-15% of all links have been observed to be asymmetric, with lower transmission power and longer node distance resulting in more asymmetric links. If not properly considered, asymmetric links may result in fake neighbors (received HELLO from a neighbor but cannot send any data to it) and broken data communication (can send data to neighbor, but cannot receive acknowledgments).

Another issue is the physical length of a link. Even though two nodes are very close together, they may not be able to establish a link (*missing short links*). On the other hand, two nodes that are very far away from each other (well beyond the nominal communication range of a node), may be able to communicate (*unexpected long links*). Experiments in [16] show that at low transmit power about 1% of all links are twice as long as the nominal communication range. These link characteristics make node placement highly non-trivial as the signal propagation characteristics of the real-world setting have to be considered [19] to obtain a well-connected network.

Most sensor network MAC protocol achieve energy efficiency by scheduling communication times and turning the radio module off in-between. Clock drift or repeated failures to re-synchronize the communication phase may result in failures to deliver data as nodes are not ready to receive when others are sending. In [3], for example, excessive *phase skew* has been observed (about two orders of magnitude larger than the drift of the oscillator).

2.3 Path Problems

Many sensor network applications rely on the ability to relay information across multiple nodes along a multi-hop path. In particular, most sensor applications include one or more sink nodes that disseminate queries or other tasking information to sensor nodes and sensor nodes deliver results back to the sink. Here, it is important that a path exists from a sink to each sensor node, and from each sensor node to a sink. Note that information may be changed as it is traversing such a path, for example due to data aggregation. Two common problems in such applications are hence *bad path to sink* and *bad path to node*. In [15], for example, *selfish nodes* have been observed that did not forward received traffic, but succeeded in sending locally generated messages.

Since a path consists of a sequence of links, the former inherits many of the possible problems from the latter such as *asymmetric paths*, *high latency*, *path oscillations*, and *high message loss*. In [7], for example, a total message loss of about 58% was observed across a multi-hop network.

Finally, *routing loops* are a common problem, since frequent node and communication failures can easily lead to inconsistent paths if the software isn't properly prepared to deal with these cases. Directed Diffusion [20], for example, uses a data cache to suppress previously seen data packets to prevent routing loops. If a node reboots, the data cache is deleted and loops may be created [21].

2.4 Global Problems

In addition to the above problems which can be attributed to a certain subset of nodes, there are also some problems which are global properties of a network. Several of these are failures to meet certain application-defined quality-of-service properties. These include *low data yield*, *high reporting latency*, and *short network lifetime* [22].

Low data yield means that the network delivers an insufficient amount of information (e.g., incomplete sensor time series). In [9], for example, a total data yield of only about 20-30% is reported. This problem is related to message loss as discussed above, but may be caused by other problems such as a node crashing before buffered information could be forwarded, buffer overflows, etc. One specific reason for a low data yield is a *partitioned network*, where a set of nodes is not connected to the sink.

Reporting latency refers to the amount of time that elapses between the occurrence of a physical event and that event being reported by the sensor network to the observer. This is obviously related to the path latency, but as a report often involves the output of many sensor nodes, the reporting latency results from a complex interaction within a large portion of the network.

The lifetime of a sensor network typically ends when the network fails to cover a given physical space sufficiently with live nodes that are able to report observations to the observer. The network lifetime is obviously related to the lifetime of individual nodes, but includes also other aspects. For example, the death of certain nodes may partition the network such that even though coverage is still provided, nodes can no longer report data to the observer.

3 Passive Indicators

An indicator is an observable behavior of a sensor network that hints (in the sense of a heuristic) the existence of one of the problems discussed in the previous section. In the context of our work we are particularly interested in indicators that can be observed purely by overhearing the traffic of the sensor network as this does not require any instrumentation of the sensor nodes. We call such indicators *passive*. The structure of this section mirrors that of the previous section, discussing passive indicators for the problems outlined there.

In fact, passive indicators heavily depend on the protocols used in the sensor network. However, there are a number of protocol elements that are commonly found in sensor network applications that can be exploited. For example, many protocols exchange regular beacon messages, all packets need to contain the per-hop destination MAC address, some packets also contain the per-hop source MAC address to identify the sender of the message, and some packets do contain a monotonically increasing sequence number.

3.1 Node Problems

Node death Many commonly used MAC and routing protocols (e.g., [17, 23]) require every node to transmit a beacon message at regular intervals, in particular for the purpose of synchronization and neighbor management. Failure to transmit any such message for a certain amount of time (typically a multiple of the inter-beacon time) is an indicator for node death. Also, node death can be assumed if a node is no longer considered a neighbor by any other node (see Sect. 3.2).

Node reboot When a node reboots, its sequence number counter will be reset to an initial value (typically zero). Hence, the sequence number contained in messages sent by the node will jump to a smaller value after a reboot with high probability even in case of lost messages, which can serve as an indicator for reboot. Note that a reboot cannot be detected this way when the node crashes just before the sequence number counter would wrap around to its initial value. However, a simple fix would be to set the sequence counter to some value other than the initial value at wrap-around, such that a wrap-around could be distinguished from a reboot.

Wrong sensor readings These can only be passively observed when application messages contain raw sensor readings. The way how the decision whether a certain sensor reading is wrong or not is implemented strongly depends on the application. One could for example exploit the fact that sensor values of nearby nodes are correlated in many applications. For other applications, the range of valid sensor values might be known a priori.

3.2 Link Problems

Discovering neighbors Depending on node density, a node in a sensor network may have a large number of other nodes within communication range with largely varying link quality. Most multi-hop routing protocols maintain a small set of neighbors with good link quality. Unfortunately, the set of neighbors chosen by a node cannot be passively

observed directly. However, there are two ways to learn about the neighbors of a node. Firstly, by overhearing the destination addresses of messages a node sends we can learn a subset of the neighbors. The second approach exploits link advertisements sent by nodes to estimate link quality. Since links are often asymmetric in sensor networks, link quality estimation must consider both directions of a link. Since a sensor node can only measure the quality of one direction of a link directly (e.g., by means of the fraction of beacon messages being received), nodes broadcast link advertisement messages containing the addresses and quality of the incoming links from their neighbors. These messages can be passively observed to obtain information about the neighbors of a node and the quality of the associated link.

Knowing the neighbors of a node, we can detect neighbor oscillation and isolated nodes. If the locations of nodes are known, we can also discover missing short links and unexpected long links.

Message loss Again, it is not directly possible to decide whether or not a node has received a message by means of passive observation. However, in many situations reception of a message by a node does trigger the transmission of another message by that node (e.g., acknowledgment, forwarding a message to the next hop). If such a property exists, failure to overhear the second message within a certain amount of time after the first message has been overheard is an indicator for message loss. Note that with this approach, it is not possible to tell apart message loss from selfish nodes that receive but fail to forward messages. Another issue with this approach is that the DSN may fail to overhear the second message although it has actually been sent. In this case, one would take the wrong conclusion that message loss occurred.

Latency To estimate the latency of a link, the same approach as for detecting message loss is used. The time elapsed between overhearing the causal and the consequential message gives an estimate of the latency, which includes processing delays in the node.

Congestion The level of link congestion (i.e., frequency of collisions) perceived by a sensor node cannot be passively observed. The level of congestion experienced by a deployment support node overhearing the traffic that is being addressed to this sensor node can be used as a rough approximation.

Phase skew Again we can exploit the existence of beacon messages that are sent at regular time intervals. A change of the time difference between receipt of beacons of neighboring nodes indicates phase drift. Averaging over multiple beacon intervals can help eliminate variable delays introduced by, e.g., medium access.

3.3 Path Problems

Discovering paths In order to discover the path between two sensor nodes (e.g., from node to sink and from sink to node), we would need access to the routing information maintained by sensor nodes. As for the neighbor table, this is not directly possible with passive observation. There are two possible ways around. Firstly, we can reconstruct a

path by tracking a multi-hop message as it travels from source to destination. Using the per-hop sender and receiver addresses of overheard messages, we can reconstruct multi-hop paths. However, for this we need a way to decide whether two messages belong to the same multi-hop transmission. This is easily possible if the message payload contains a unique identifier such as an end-to-end sequence number. Also, if the message payload is relayed unmodified along the path, we can compare packet contents to decide (with some probability) whether two packets belong to the same multi-hop message transmission. Things get more difficult if message contents are changed along the path, for example due to in-network data aggregation. In this case, one might be able to exploit temporal correlations between messages, assuming that a node will forward a (modified) message soon after it has received the original message.

Another issue is that some protocols due not include the per-hop source address in messages as it is not needed due to the lack of acknowledgments. For example, TinyOS 1.x does not provide a field for per-hop source address in the standard packet header. One possible heuristic to infer the missing per-hop source address nonetheless exploits the fact that the per-hop source address of a forwarded packet equals the per-hop destination address of the original packet.

Secondly, we can overhear routing messages (if there exist any) to discover paths. While these messages typically indicate that a node has established a route, it is often impossible to reconstruct the route. To construct a spanning tree, for example, it is sufficient that nodes broadcast messages containing their address and distance to the sink, but not their parent in the tree. The latter would be necessary to reconstruct the spanning tree from overheard traffic.

If paths can be discovered, we can also easily detect path oscillations and find missing paths from nodes to sink and vice versa. Using similar techniques as for links, we can estimate message loss and latency along a path.

Loops Like for path discovery, we need a mechanism to decide whether or not two packets belong to the same multi-hop message exchange. If such a mechanism exists, a message that is addressed to a node that previously sent the same message indicates a routing loop. Alternatively, discovered paths can be directly examined for loops.

3.4 Global Problems

As discussed in Section 2, global problems such as low data yield, high reporting latency, or insufficient network lifetime are typically due to a combination of different node, link, and path problems. Hence, the indicators for the latter problems discussed above can be considered as indicators for these global problems.

Partitions Knowledge of the routing paths as discussed above allows to obtain an approximate view on the routing topology and to detect network partitions.

3.5 Discussion

As we have shown throughout this section, for many common problems that occur during deployment of sensor networks, passive indicators exist that allow to infer the existence

of a problem from overheard network traffic. However, in some situations we had to make assumptions about the underlying sensor network protocols that may not hold for all applications (e.g., indicators for message loss). For other problems, passive indicators provide only an approximate view of the ground truth (e.g., indicators for network congestion).

In such situations, we can resort to *semi-active observation*, where sensor nodes are instrumented with code to emit messages containing additional information about the state of the sensor node (e.g., level of congestion, battery voltage, etc.). These messages are overheard by the DSN and ignored by other sensor nodes. While this approach requires instrumentation of sensor nodes and transmission of additional messages, these messages do not have to be routed through the sensor network.

Another alternative is to modify existing or to develop new protocols that support passive inspection *by design* by including small amounts of additional information into protocol messages. Summarizing our observations in this section, we make the following recommendations for the design of such protocols:

- Include the per-hop source address in all messages to identify communication partners.
- Include a sequence number in all messages so that the DSN can detect when it failed to overhear a message.
- Include information in multi-hop messages that allows to decide whether or not two packets belong to the same multi-hop message exchange.
- Include information in routing messages that allows to reconstruct the routing graph.

4 Passive Inspection with SNIF

In this section we outline how passive indicators discussed in the previous section can be implemented in SNIF. For this, consider the architecture of SNIF as depicted in Fig. 2, which 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. The key design goal for SNIF is generality that is, it should support passive inspection of a wide variety of sensor network protocols and applications.

4.1 DSN

Using a deployment support network of BTnode Rev3 nodes [24], sensor network traffic is overheard as illustrated in Fig. 1. Each DSN node provides two radios which operate in different frequency bands and are thus free of interference: A ChipCon CC1000 (also used on MICA2 nodes) to overhear sensor network traffic, and a Zeevo Bluetooth 1.2 radio to deliver overheard packets to the DSN sink. Bluetooth has been chosen due to its robustness: originally, it has been designed as a cable replacement and employs techniques such as frequency hopping and forward error correction to minimize packet loss.

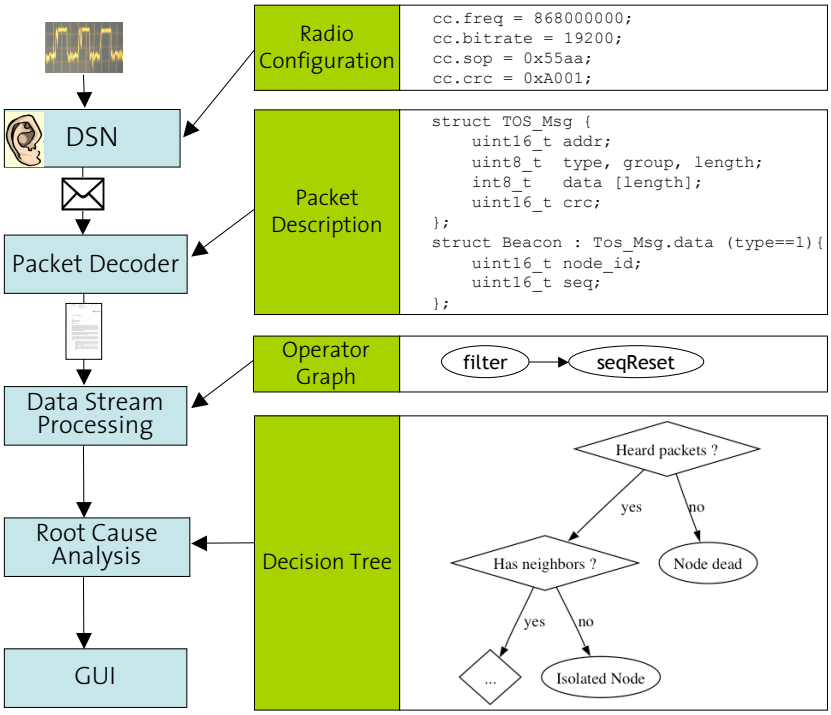

Figure 2: Architecture of SNIF.

As illustrated in Fig. 2, the ChipCon radio and the radio driver can be configured to support different radio frequencies (`cc.freq`), bit rate (`cc.bitrate`), start-of-packet delimiters (`cc.sop`), and checksum polynomials (`cc.crc`). The SOP is a special bit sequence that precedes every radio packet. In the received stream of bits, a DSN node looks for the radio preamble (a bit sequence to synchronize sender and receiver) that is succeeded by the SOP and captures the following packet. This way, the DSN can receive packets regardless of the specific MAC protocol used by the sensor network. Overheard packets are time-stamped and routed to the DSN sink (a laptop computer with Bluetooth networking), where duplicate packets (resulting from multiple DSN nodes overhearing the same sensor node) are removed and packets are sorted by increasing time stamps.

4.2 Packet Decoder

The next step is to decode the contents of overheard packets. As there are no standard protocols for sensor networks, SNIF provides a flexible packet description language, which is an annotated version of C structures. Fig. 2 shows an example of a TinyOS message (`TOS_Msg`) holding a beacon data unit if the message type of the TinyOS message equals 1. The result of packet decoding is a record consisting of a list of name-value pairs, where each pair holds the name and value of a data field in the packet.

4.3 Data Stream Processor

The resulting stream of packets (i.e., records holding name-value pairs) is then fed to a data stream processor to detect any problems with the sensor network. The data stream

processor executes operators that take a stream of records as input and produce a different stream of records as output. The output of an operator can be connected to the input of other operators, resulting in a directed operator graph. SNIF provides a set of standard operators, e.g., for filtering, aggregation over time windows, or merging of multiple streams into one. In addition, application-specific operators to detect specific problems in the sensor network may be required. Fig. 2 shows an simple operator graph that is used to detect node reboots as described in Sect. 3. The first operator (`filter`) reads the packet stream generated by the DSN and removes all packets that are not beacon packets. Recall that beacon packets are regularly transmitted by each node to support neighbor discovery and contain the sender address and a sequence number. The second operator (`seqReset`) remembers the last sequence number received from each node and checks if a newly received sequence number is smaller than the previous one for this node, in which case the node has rebooted unless there was a sequence number wrap-around (i.e., maximum sequence number has been reached and sequence counter wraps to zero). If a reboot is detected, the `seqReset` operator outputs a record holding the address of the node that rebooted. In this example, `filter` is a standard operator provided by SNIF, while `seqReset` is an application-specific operator.

Development of such application-specific operators is the core task to customize SNIF for detection of a specific deployment problem. In practice, each such operator implements a passive indicator as described in Sect. 3. The key challenge in implementing such operators is dealing with *incomplete information*, as the DSN may fail to overhear some sensor network messages. For example, the seqReset operator mentioned above implements heuristics to tell apart reboots from sequence number wrap-around even in case of lost messages.

For each possible problem discussed in Sect. 2, an operator graph must be defined to detect this problem. In some cases, standard operators are sufficient, but in many cases application-specific operators are required.

4.4 Root Cause Analysis

The next step is to derive the state of each sensor node, which can be either "node ok" or "node has problem X". Note that the operator graphs mentioned above may concurrently report multiple problems for a single node. In many cases, one of the problems is a consequence of another problem. For example, a node that is dead also has a routing problem. In such cases, we want to report only the primary problem and not secondary problems. For this, we use a decision tree, where each internal node is a decision that refers to the output of an operator graph, and each leaf is a node state. In the example tree depicted in Fig. 2, we first check (using the output of an operator graph that counts packets received during a time window) if any messages have been received from a node. If not, then the state of this node is set to "node dead". Otherwise, if we received packets from this node, we next check if this node has any neighbors (using an operator graph that counts the number of neighbors contained in link advertisement packets received from this node). If there are no neighbors, then the node state is set to "node isolated". Here, the check for node death is above the check for isolation in the decision tree, because a dead node (primary problem) is also isolated (secondary problem).

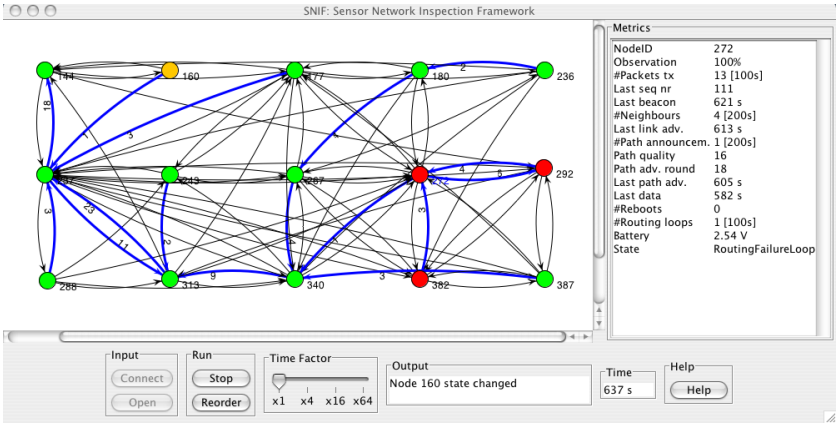

Figure 3: SNIF user interface.

4.5 GUI

Finally, node states and additional information are displayed in the graphical user interface. The core abstraction implemented by the user interface is a network graph, where nodes and links can be annotated with arbitrary information. The user interface also supports recording and playback of executions. A snapshot of an instance of the user interface is shown in Fig. 3. Here, node color indicates state (green: ok, gray: not covered by DSN, yellow: warning, red: severe problem), detailed node state can be displayed by selecting nodes. Thin arcs indicate what a node believes are its neighbors, thick arcs indicate the paths of multi-hop data messages.

4.6 Discussion

In summary, to inspect a specific type of sensor network application, radio configuration and packet descriptions must be provided, operator graphs to detect problems (possibly including application-specific operators) and a decision tree to infer node states must be developed, and the user interface must be configured to display relevant information. The key challenge here is to deal with incomplete information due to missing packets. In [25] we show that it is possible to implement loss-tolerant operators to detect many of the problems discussed in this paper. The use of SNIF to inspect a real sensor network has been demonstrated [26].

5 Related Work

Due to space constraints, we limit our discussion to the literature on wireless sensor networks. To our knowledge, all existing approaches to inspection of deployed sensor networks are active, that is, they require an instrumentation of the sensor network. This includes work on active debugging of sensor networks, notably Sympathy [6] and Memento [10]. Both systems require active instrumentation of sensor nodes and introduce monitoring protocols in-band with the actual sensor network traffic. Tools for sensor network management such as NUCLEUS [11] 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 our approach is work on simulators (e.g., SENS [27]), emulators (e.g., TOSSIM [28]), and testbeds (e.g., MoteLab [29]) as they support development and test of sensor networks *before* deployment in the field. In particular, testbeds typically provide a wired back-channel from each node, such that sensor nodes can be instrumented to send status information to an observer. EmStar [30] integrates simulation, emulation, and testbed concepts into a common framework where some nodes physically exist in a testbed or in the real world, while the majority of nodes is being emulated or simulated. Physical nodes need instrumentation and a wired back-channel. In [31], a deployment support network is used to provide a wireless back-channel to deployed sensor nodes. However, sensor nodes need to be physically wired to DSN nodes (requiring as many DSN nodes as there are sensor nodes) and sensor node software must be instrumented.

6 Conclusions

In this paper we identified problems commonly encountered during the deployment of sensor networks in real-world settings. We found four classes of problems: node problems that affect only individual nodes, link problems that affect two nodes and the wireless link formed between them, path problems that affect three or more nodes and the multihop path formed by them, and global problems that are properties of the sensor network as a whole. In addition, we discussed how the above problems can be detected by overhearing and analyzing sensor network traffic. We found that many problems can indeed be detected by exploiting common elements of sensor network protocols. We also outlined, how such passive indicators can be implemented in a practical tool called SNIF [13].

References

[1] P. Buonadonna, D. Gay, J. M. Hellerstein, W. Hong, and S. Madden. TASK: Sensor Network in a Box. In *EWSN 2005*.

[2] B. Greenstein, E. Kohler, and D. Estrin. A Sensor Network Application Construction Kit (SNACK). In *SenSys 2004*.

[3] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *WSNA 2002*.

[4] P. Padhy, K. Martinez, A. Riddoch, H. L. R. Ong, and J. K. Hart. Glacial Environment Monitoring using Sensor Networks. In *REALWSN 2005*.

[5] J. Polastre, R. Szewczyk, A. Mainwaring, D. Culler, and J. Anderson. Analysis of Wireless Sensor Networks for Habitat Monitoring. In C. S. Raghavendra, K. M. Sivalingam, and T. Znati, editors, *Wireless Sensor Networks*, chapter 18. Kluwer Academic Publishers, 2004.

[6] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the Sensor Network Debugger. In *SenSys 2005*.

[7] R. Szewcyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An Analysis of a Large Scale Habitat Monitoring Application. In *SenSys 2004*.

[8] 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 *REALWSN 2005*.

[9] 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 *SenSys 2005*.

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

[11] G. Tolle and D. Culler. Design of an Application-Cooperative Management System for Wireless Sensor Networks. In *EWSN 2005*.

[12] J. Beutel, M. Dyer, L. Meier, and L. Thiele. Scalable Topology Control for Deployment-Sensor Networks. In *IPSN 2005*.

[13] M. Ringwald, K. Römer, and A. Vialetti. SNIF: Sensor Network Inspection Framework. Technical Report 535, Departement of Computer Science, ETH Zurich, 2006.

[14] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a Sensor Network Expedition. In *EWSN 2004*.

[15] O. Visser K. Langendoen, A. Baggio. Murphy Loves Potatoes: Experiences from a Pilot Sensor Network Deployment in Precision Agriculture. In *WPDRTS 2006*.

[16] 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.

[17] A. Woo, T. Tong, and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *SenSys 2003*.

[18] N. Ramanathan, E. Kohler, and D. Estrin. Towards a Debugging Systems for Sensor Networks. *Int. J. Network Management*, 15, 2005.

[19] A. Terzis, R. Burns, and M. Franklin. Design Tools for Sensor-Based Science. In *EmNets 2006*.

[20] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed Diffusion for Wireless Sensor Networking. *IEEE/ACM Trans. Netw.*, 11(1), 2003.

[21] A. Sobeih, M. Viswanathan, D. Marinov, and J. C. Hou. Finding Bugs in Network Protocols Using Simulation Code and Protocol-Specific Heuristics. In *LNCS*, volume 3785. Springer, 2005.

[22] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman. A Taxonomy of Wireless Micro-Sensor Network Models. *ACM MC2R*, 6(2), 2002.

[23] 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.

[24] BTnodes. A Distributed Environment for Prototyping Ad Hoc Networks. www.btnode.ethz.ch.

[25] M. Ringwald, K. Römer, and A. Vialetti. Passive Inspection of Sensor Networks. In *DCOSS 2007*.

[26] M. Ringwald, M. Cortesi, K. Römer, and A. Vialetti. Demo Abstract: Passive Inspection of Deployed Sensor Networks with SNIF. In *EWSN 2007*.

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

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

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

[30] 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*.

[31] M. Dyer, J. Beutel, T. Kalt, P. Oehen, L. Thiele, K. Martin, and P. Blum. Deployment Support Network - A Toolkit for the Development of WSNs. In *EWSN 2007*.