On the Need for Passive Monitoring in Sensor Networks

Abdalkarim Awad, Rodrigo Nebel, Reinhard German and Falko Dressler
Autonomic Networking Group, Computer Networks and Communication Systems
University of Erlangen-Nuremberg, Germany
{abdalkarim.awad,german,dressler}@informatik.uni-erlangen.de

***Abstract*—Debugging and analyzing Wireless Sensor Networks (WSNs) are important tasks for improving the quality and performance of the network. In this paper, *Pimoto* is to be presented, which is a distributed passive monitoring system implemented for debugging and analyzing WSNs. It is based on a hierarchical structure allowing to monitor different networks simultaneously and to analyze the obtained information at a dedicated PC. The system relies on three components. The first element is the monitoring node. It intercepts the radio packets in the vicinity and sends received packet information to a gateway using a second radio interface in order to prevent intrinsic interactions with the sensor network operation. The gateway has the ability to communicate directly with the monitoring node and to transfer all the collected monitoring data to the third component, a dedicated PC (server), in the hierarchy using standard TCP/IP communication. The packets are analyzed and visualized on the server using the standard network monitoring and analyzing tool Wireshark. The most important characteristic of this monitoring concept is the passive operation, i.e. the normal operation in the WSN is not influenced by the analyzer.**

I. Introduction

Wireless Sensor Networks (WSNs) consist of many small sensor nodes, which communicate using radio interfaces over a wireless channel. The main function of these networks is to sample the surrounding environment using attached sensors [1]. In the early beginnings, these networks were considered to consist of few sensor nodes connected to a central control unit. Today however, the emphasis of the research is on the area of the distributed operation in wireless sensor nodes. Such sensor networks are assumed to consist of low cost sensor nodes and to operate in a distributed and self-organized manner. However, these nodes have constraints on the resources (power, memory communication channel) [2].

The communication in WSNs is often complex and in some cases difficult to predict. Especially during the development of WSNs, methods for analyzing and debugging communication methods are strongly demanded. Usually, only the behavior of single sensor nodes can be supervised using directly attached debugging interfaces [3]. Therefore, the communication between nodes can only be estimated if all participating nodes can be analyzed simultaneously. A second problem lies in the operation and control of already deployed sensor networks. In failure situations, tools are needed to analyze the behavior of the network as a whole.

In this work, we present an architecture (see Figure 1) and a tool to passively monitor sensor networks, which we named *Pimoto*. We are able to intercept all radio packets in the network, and to store and transmit them to a central server for further analysis. The transport and the coordination of the monitoring environment is accomplished in a hierarchical way. That enables the system to scale for large numbers of nodes in a deployed sensor network.

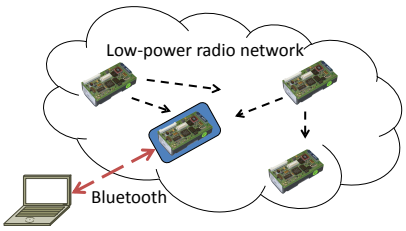

Fig. 1. Basic architecture of *Pimoto*

For easier and more comprehensive analysis, we developed a plugin for Wireshark[1] for graphical analysis. Our target are typical WSNs, therefore we implemented our approach using BTnode[2] nodes. These sensor nodes have the ability to communicate either with a low power radio signal (Chipcon CC1000) or using a Bluetooth interface. Our approach, as will be explained later in this paper, is to place a node inside the network to passively collect all radio packets in the wireless medium. Thereafter, these packets are transported by means of Bluetooth to a local PC and further to a central server to be analyzed and visualized.

The rest of the paper proceeds as follows. In Section II, we outline relevant related work. In Section III, an overview to the complete system architecture is presented. Afterwards, the implementation and some evaluation results are presented in Sections IV and V, respectively. Finally, we give a conclusion in Section VI.

II. Related work

In network monitoring, *passive* and *active* monitoring techniques are distinguished. In general, active monitoring refers to the active interaction with the system under observation. Thus, the system behavior is being influenced by the monitoring actions. Nevertheless, active monitoring allows to obtain detailed information about the system parameters. In contrast,

[1] http://www.wireshark.org/
[2] http://btnode.ethz.ch/

passive monitoring means the access to the network traffic without interfering the communicating systems. Nevertheless, additional hardware is required for this purpose. The obtained data can be used for several aims. We are focusing on passive management and control of sensor networks and on enhanced debugging during protocol development.

In spite of the fact that network analysis is critical for understanding and improving the performance of networks, there is no much work done in this area. In this section, we show selected approaches that have been described in the literature.

The Sensor Network Management System (SNMS) [4] introduces a set of TinyOS components , which can be integrated into developed application. The goal is information exchange between these components and an instant analysis. For this, SNMS also introduces the appropriate tools. The user has an overview, by means of the TinyOS components and the gathered data, of the behavior and the condition of the specific SNMS-enabled node. In order to avoid unnecessary communication, which can also lead to unwanted interferences between the nodes, the information is sent only on explicit inquiries of the user to the evaluating instance. If unexpected events occur on the nodes, information is stored to these nodes locally on the sensor nodes so it is available for evaluation. SNMS also allows to configure the frequency and event types.

A further monitoring system is Sympathy [5], [6]. Sympathy is an active monitoring system in which the sensor nodes are supplied with an additional piece of software. All sensor nodes periodically send local information to a dedicated sink node, which is used to gather and evaluate data from the sensor network. By means of this evaluation the errors and their location can be recognized. Sinks receive their information from three sources. First, each sensor node continuously runs additional software packages. This is needed to obtain information about the conditions in the sensor node and to prepare it to be sent to a Sympathy sink in periodic intervals. The second source is the sink node itself. The same software is installed in sinks. Thus, also information about the conditions of sinks are noted and processed locally, i.e. sinks supervise their own conditions. The last source is the direct environment of the sink node as this node continuously monitors the data traffic in the vicinity and evaluates it.

Wit [7] is a passive monitoring system. Although it may be the closest approach to our system, it has been developed for another purpose. Wit is designed to test the performance of the MAC layer (802.11) in wireless networks, whereas our system is designed to analyze the protocol behavior in WSNs. Wit uses three stages to construct an enhanced trace of the system behavior. In the first phase, a merging process is accomplished to obtain a better view from multiple monitoring sources. In the second stage, a formal language technique is used to infer which packets were received by their destinations and also to infer packets which are not logged by any monitor. Finally, Wit derives network performance measures from this enhanced trace.

The raising demand for sensor network monitoring encouraged also the ScatterWeb project [8] to include monitoring techniques. The integrated tool ScatterViewer allows to manage sensor nodes and to collect status information. It also represents an active monitoring application that may influence the network behavior.

In [9], an approach that targets non-intrusive monitoring in already deployed WSNs is presented. Here, the concept of Deployment Support Networks (DSN) [10] is proposed. In the suggested approach monitoring nodes are deployed in an existing wireless sensor network, these nodes collect and decode packets. To ease the decoding of packets, a packet structure is assumed; thereafter the decoded packets are ready for further analysis.

Another passive monitoring environment has been developed, which was named TWIST [3]. It collects information from the participating sensor nodes using an USB-based cable network. This architecture allows to specifically debugging all connected nodes while it does not support the collection of radio messages. Additionally, an expensive infrastructure (USB cables) must be provided.

III. PIMOTO

In order to enable a comprehensive monitoring in WSNs, we developed *Pimoto*, a distributed monitoring environment for passive monitoring in sensor networks. In the following, we describe its architecture and characteristics. Two aspects should be supported by our passive monitoring system. First, we want to allow hierarchical monitoring, i.e. multiple deployed monitoring nodes have to be interconnected by a network separate from the WSN. Secondly, the collected packet data should be visualized at the central server in real-time, i.e. the latency from monitoring, transmission, and analyzing should be minimized.

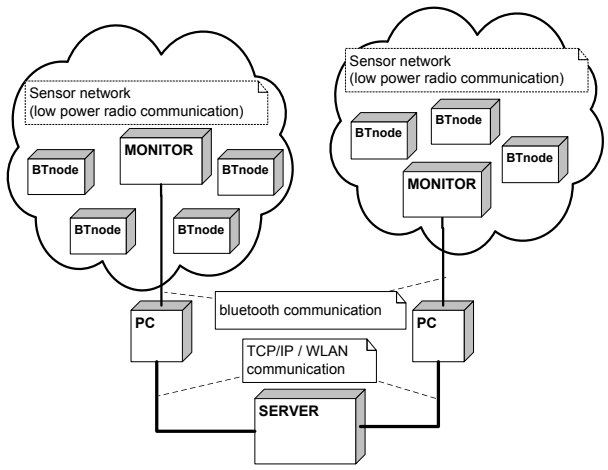

Fig. 2. Hierarchical structure of *Pimoto* supporting so called monitor islands

A. Principle architecture for passive monitoring

There are mainly two common approaches to monitoring the network, passive and active techniques. The passive approach uses devices to watch the traffic as it passes by. These

devices can be special purpose devices or can be built on common network devices like routers, access points, switches, or even normal hosts. The passive monitoring devices are polled periodically and information is collected. Although this technique does not increase the traffic on the network for the measurements, the amount of data gathered can become substantial if one is trying to capture information of all the data packets. In comparison, the active approach relies on the capability to inject packets into the networks or send packets to servers or applications, following them and measuring service obtained from the network. Because of energy constrains and for real-time monitoring without influencing the sensor network application, the passive approach is preferable. The gathered data can be used to debug, and analyze the network performance.

The basic concept is depicted in Figure 1. A three-node sensor network is analyzed by a fourth node, which is a dedicated *Pimoto* monitoring node is placed within the radio range of the WSN. This monitoring node collects as much information from the sensor network as possible, i.e. it sniffs all the radio packets. Finally, it forwards the received information to a PC using its Bluetooth radio interface. Bluetooth is used for several reasons. First, it is "just available" on standard BTnode sensor nodes. Thus, no additional radio hardware needs to be implemented. Secondly, the Bluetooth protocol already provides much better features to establish reliable data communication compared to the typical low-power radio used in sensor networks. Finally, the independent radio interface allows to transmit monitoring data to a gateway PC while simultaneously receiving packets on the other radio interface. Without any change of the functionality, Bluetooth can easily be replaced by any other wireless or wired communication interface.

B. Hierarchical setup of monitoring islands

The hierarchical structure is depicted in Figure 2. As can be seen, monitoring nodes are placed inside a sensor network (monitoring islands). These nodes collect all the radio traffic. Using a second radio interface (in our case, we employ Bluetooth), the monitoring data is delivered to a gateway PC, which may control multiple monitoring nodes. The gateway PC forwards the data using a TCP/IP network, e.g. over WLAN-based wireless infrastructure, to a central server. At this place, the standard network monitoring application Wireshark is used to receive, to decode, and to visualize the packet information.

The hierarchical structure of architecture plays an important role regarding the distribution of resources. As is shown in Figure 2, several *monitoring islands* are considered to operate simultaneously. Each monitoring island is assigned to exactly one monitoring node for the transmission of the packets to an associated PC. This association will usually be a one to one mapping. However, if multiple monitoring nodes are in the same vicinity, they can all be associated to a single PC. In this case, the Bluetooth communication between the monitoring node and the PC can be a limiting factor. Finally, the actual architectural decision depends on the application scenario.

If one would like to supervise several spatially separated from each other sensor networks, the distributed scenario remains as only meaningful solution. Nevertheless, this scenario has disadvantages as it requires more hardware and strongly depends of high quality time synchronization between the gateway PCs.

C. Push and pull techniques

The terms *push* and *pull* were originally used in the marketing domain. Different sales and advertising strategies are called push and pull marketing, respectively [11]. Classical examples of push marketing are mail distributions, radio, and television. The information "is pushed" into the communication channel whether desired by the consumer or not. In contrast, the pull method requires the customer to become active and request the desired information explicitly. A well-known example is the world wide web.

The same techniques can be used to transmit the received monitoring data from the monitoring nodes to the server. In the entire system, monitoring data is flowing from the monitoring node over the connected PC, which is maintaining a single monitoring island, towards the server for further analysis. One of the most important questions regarding the performance and the quality of the monitoring system is the design decision for one of these techniques.

Considering a pull-based architecture, the server needs to ask for the desired information. This could be done periodically or on-demand by intervention of an user. Thus, the server sends an inquiry to the PC, which forwards this inquiry to the monitoring node. Then, the desired information is delivered to the server via the PC. In contrast, the push architecture relies on the transmission of monitoring data initiated by the monitoring node. This action can be triggered by a condition, e.g. timers, partial consumption of the internal memory, or the receipt of a packet with special contents.

We decided to use the push technique as the storage capabilities of the employed monitoring nodes (actually, these are typical embedded sensor nodes) are strongly limited. Thus, the intercepted radio packets must be forwarded as soon as possible to re-use the memory for further data. Additionally, the demanded real-time behavior of the analysis as well as the typically very low data rates in WSN contributed to our decision.

D. Communications protocols

The developed system consists of four components. First, an application on the monitor node running a thread to intercept the radio packets in the sensor network to be stored and transferred to the gateway PC. Secondly, an application on the gateway PC, which processes the radio packets and sends these again to a server. Third, the communication between PC and server is supervised by Wireshark. Finally, a plugin developed particularly for this type of radio packets performs the interpretation of the data. All the components relevant for the communication are described below.

1) Monitoring node → gateway PC: For data distribution, we decided to use the push solution as mentioned before, i.e. the monitoring node supplies its data to the gateway PC in regular intervals. The communication is performed using the Rfcomm protocol. It guarantees reliable data exchange over Bluetooth. The employed BTnode sensor nodes primarily use the Berkeley MAC (BMAC) protocol. Table I shows the original packets format used by BMAC.

Field	Meaning
Source address (2 Byte)	BMAC source address
Destination address (2 Byte)	BMAC destination address
Length of Data (2 Byte)	Length of the data
Type (1 Byte)	Application type
Data (length of data)	Packet data

TABLE I
BMAC DATA FORMAT

2) Gateway PC → Server: The communication between the gateway PC and the server takes employs standard TCP/IP protocols. In particular, TCP is used as transport protocol for reliable data transmission between the gateway PC and the server. Apart from forwarding of the monitoring packets, the gateway PC is responsible for providing meta information about the original packet. Such information is stored in additional fields before transmitting the monitored data to the server. According to these updates, Wireshark can perform detailed traffic and protocol analysis. In Table II, the additional fields are shown. Apart from the MAC address of the monitoring node, two timestamp fields are added that become necessary for the later computation of the reception time through Wireshark.

Field	Meaning
Monitor address (6 Byte)	MAC Address of monitoring node
Source address (2 Byte)	BMAC source address
Destination address (2 Byte)	BMAC destination address
Length of Data (2 Byte)	Length of the data
Type (1 Byte)	Application type
Seconds (4 Byte)	Seconds of the reception time
Milliseconds (2 Byte)	Milliseconds of the reception time
Data (length of data)	Packet data

TABLE II
EXTENDED BMAC DATA FORMAT

IV. IMPLEMENTATION

In the following, we describe the used hardware and software components and outline the most important aspects of our implementation of *Pimoto*.

A. Hardware and software of BTnode and PCs

For implementing *Pimoto*, we used the BTnode sensor nodes rev3. This sensor node incorporates two radio interfaces, the Chipcon CC1000 for low power radio transmissions and a Bluetooth interface. Such a BTnode provides 64 kByte RAM and some additional flash and eeprom capacity for programs and permanent configuration parameters.

On the BTnodes, we used Nut/OS as the operating system. It specifically supports small embedded systems and has been developed particularly for the ATMEL ATmega128 micro controller.

For the gateway and server PCs, we used standard PC components (desktops and laptops) that we further extended using a BlueFRITZ! Bluetooth adapters . The theoretically maximum transmission rate is approximately 723 kBits/s for Bluetooth communication over 100 m.

On all PCs, we used Linux as the operating system. It already provides support for our Bluetooth adapter in form of the BlueZ library and some associates kernel modules.

Finally, we used Wireshark, which was formerly known as Ethereal, for graphical networks analysis. It is able to analyze the packet either online or offline. Packet monitoring can be done based on any available network interface.

B. BTnode monitoring software

We developed a sensor program that sets the radio interface in the *promiscuous mode*, i.e. in a mode that allows to capture any packet regardless of its destination address. As this capability is currently not supported by the used BMAC protocol implementation, we extended the BTnut driver accordingly. For the Bluetooth communication, we employed the Rfcomm protocol. It represents a simple serial connection between the sensor node and the gateway PC. This PC forwards all received monitoring data to the server using TCP/IP.

The principles of the BTnode monitoring software are depicted in Figure 3. Two threads are concurrently monitoring radio packets and transmitting them to the gateway PC. Both threads are coupled by a shared memory that was implemented in form of a ring buffer.

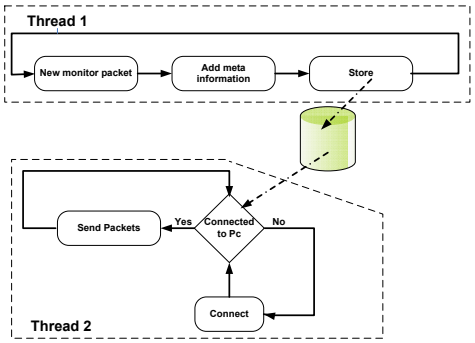

Fig. 3. Architecture of the BTnode implementation

C. Gateway PC

The substantial task of the application on the gateway PC consists of receiving packets from the monitoring node and passing them forward to the server. However, before a data exchange with the monitor node takes place, the gateway PC is kept in a waiting state. The connecting inquiry takes place via the monitoring node. After establishing the connection, monitoring data can be exchanged. The gateway PC immediately forwards the received packets, i.e. the monitored radio communication, to the server.

The gateway PC also computes timestamps for each packet that it received from the monitoring node and stores this meta information in two additional fields in the packet header (see also Table II). The format of the seconds field is UNIX standard time, i.e. the seconds since January 1, 1970. In the second field, the milliseconds are stored. These values are later used by Wireshark to compute the reception time of the radio packets. Additionally, a six byte field is used for the identification of the monitoring node. In this, the Bluetooth MAC address of the monitoring node is stored.

Synchronization of the timestamps that are stored with each captured packet has been proven to be the most challenging issue. We need accurate timestamps in order to (re-)order the received packets in the analysis stage. Unfortunately, the clocks of the monitoring nodes cannot easily be synchronized. Therefore, we used a trick to synchronize the timestamps in the second hierarchy, i.e. at the gateway PCs. Each sensor node provides a 4-byte counter showing the milliseconds since its last reboot. Thus, the uptime of the sensor node can be exactly measured for 49.7 days (then, the counter overflows). We used this uptime measure as the recorded timestamp in each collected packet. Then, prior submission to the connected PC, the timestamp is subtracted from the current submission time, i.e. depicting the time difference between reception and forwarding. The gateway PC can then update the timestamp according to its, e.g. NTP-synchronized, local time.

D. Wireshark plugin

At the server, we used the standard monitoring environment Wireshark for data analysis and filtering. Our plugin "BTnode Radio Protocol" decodes and interprets the received data packets. Currently, we support all features from the BMAC protocol while other elements (and protocols) can easily be added. The graphical analysis is depicted in figure 5. Shown is a decoded BMAC packet including source and destination address, length, type, timestamp, and payload data.

V. Evaluation

We executed a number of experiments to analyze the functionality of *Pimoto* as well as its performance limitations. In the simplest scenario, one sending sensor node and one monitoring node, we were able to receive all the transmitted packets. Thus, we decided to setup some more complex scenarios. In the following, four different setups are described and the measurement results are discussed.

A. Starting simple – illustration of the functionality

The first test is intended to illustrate the functionality of *Pimoto*. For this purpose, we created with a small network consisting of two sensor nodes that continuously send radio packets to each other. The monitor node is placed within the range of the sensor network to intercept these packets and to transfer them for further evaluation to the gateway PC. The structure is to be seen in Figure 4.

Table III shows the number of the monitored packets compared to the received packets. It is clear that the monitoring

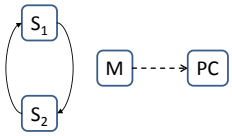

Fig. 4. Scenario for the first experiments. Two nodes (S_1 and S_2) are sending packets to each other, the *Pimoto* monitor (M) is sniffing all the traffic.

node can monitor more packets than what both nodes can receive because it is always ready to receive while the sensor nodes periodically switch between sending and receiving packets. Moreover the result shows that, when the packet size is increased, the number of monitored as well as the received packets declines.

Data size	Sent packets	Received packets	Sniffed packets
69	100	51	83
119	100	52	79
219	100	50	79
419	100	42	65

TABLE III
RESULTS FROM THE FIRST SCENARIO.

In Figure 5, a screenshot from Wireshark is shown after receiving the monitoring data. Wireshark provides the capability to use filters to display only the desired data. These packets can now be examined much more conveniently since all unnecessary data traffic is faded out. In our example, we used the expression `btnode.typ == 3 && btnode.src == 1` to only display packets of node 1 and of type 3.

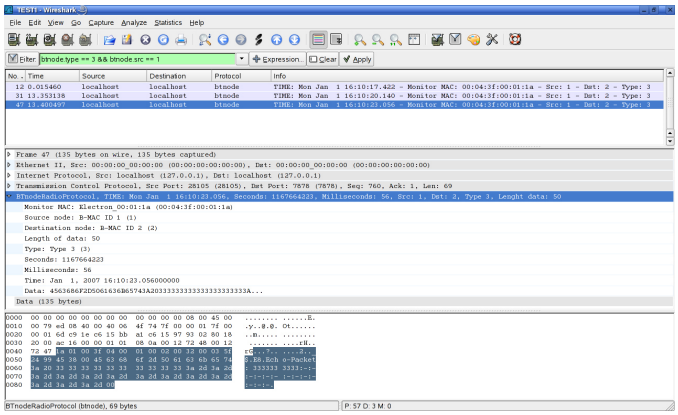

Fig. 5. Filtered packet output based on a limiting expression.

As an example, a measured BMAC packet is listed in the following. This BMAC message is sent from node 7 to node 8.

```
BTnodeRadioProtocol,
  TIME: Tue Aug  7 11:32:43.950,
  Seconds: 1186479163,
  Milliseconds: 950,
  Src: 7, Dst: 8, Type 1, Lenght data: 8
    Monitor MAC: 00:04:3f:00:01:1a
    Source node: B-MAC ID 7 (7)
```

```
Destination node: B-MAC ID 8 (8)
Length of data: 8
Type: Type 1 (1)
Seconds: 1186479163
Milliseconds: 950
Time: Aug  7, 2007 11:32:43.950000000
Data: 0800020008000200
```

This mechanism essentially supports the debugging of communication protocols used in sensor networks. Also, it provides means for comprehensive studies of communication aspects even for non-experts such as students in a networking course.

B. Overlapping monitoring islands and time synchronization

A second test has been conducted to provide information on the accuracy of the computed reception times of radio packets that are simultaneously received by multiple monitoring nodes. The structure from the previous experiment is supplemented with a further monitoring node as shown in Figure 6.

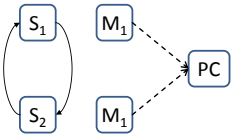

Fig. 6. Scenario for the receiving time experiments. Two nodes (S_1 and S_2) are sending packets to each other, two *Pimoto* monitors (M_1 and M_2) are sniffing all the traffic.

Both monitoring nodes intercept the radio packets exchanges between the two sensor nodes. Since both monitoring nodes (M_1 and M_2) are in the same distance to sensor S_1 and sensor S_2, an almost identical timestamp can be expected for each packet received by both monitoring nodes. We performed a number of measurements. The average deviation of the reception time was about 39.7 milliseconds. The average ratio of successfully monitored packets was 96.3%. For typical sensor networks using data rates up to 19.2 kbit/s, this deviation seems feasible.

C. Behavior under load

In the next experiment, which is depicted in Figure 7, the behavior of the implemented monitoring system was examined under load. The test environment consists of three sensor nodes, which send themselves 100 packets each of 69 byte in the first test and 300 packets of the same size in the second test.

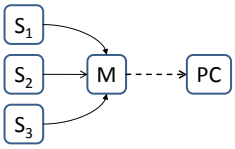

Fig. 7. Scenario for the load test. Three nodes (S_1, S_2, and S_3) are sending packets to a *Pimoto* monitor (M).

Five test runs were accomplished for each experiment to produce statistical relevant data. In the first test, on average 85% of the packets were recorded. In the second test, on average 80% were received. All results are listed in Table IV.

Additionally, statistical aspects of results are shown in Figure 8. All results are shown as boxplots. For each data set, a box is drawn from the first quartile to the third quartile, and the median is marked with a thick line. Additional whiskers extend from the edges of the box towards the minimum and maximum of the data set, but no further than 1.5 times the interquartile range. Data points outside the range of box and whiskers are considered outliers and drawn separately.

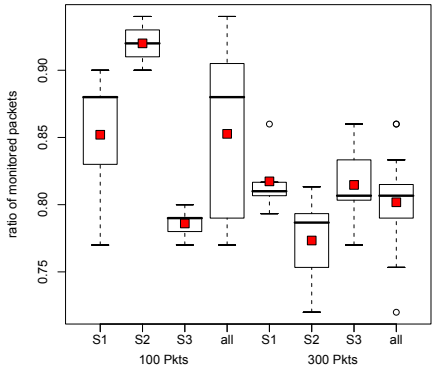

Fig. 8. Results from load test using 100 and 300 packets, respectively

	burst of 100 packets	burst of 300 packets
Src S_1	0.85	0.82
Src S_2	0.92	0.77
Src S_3	0.78	0.81
Total	0.85	0.80

TABLE IV
RESULTS FROM LOAD TEST. DEPICTED IS THE AVERAGE RATIO OF SUCCESSFULLY MONITORED PACKETS

D. Behavior at different data rates

In this final experiment, the influence of the data rate has been analyzed. We used the setup as depicted in Figure 9. In this experiment, we used four sensor nodes transmitting data to a single receiver and placed a monitoring node within the radio range of all nodes. We performed five runs each based sending 100 packets with a packet size of 69 byte from each source to the central sink.

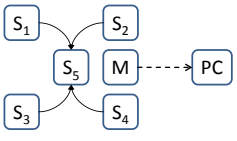

Fig. 9. Scenario for the load test. Three nodes (S_1, S_2, and S_3) are sending packets to a *Pimoto* monitor (M).

Figure 10 shows the results from this test. Again, we used boxplots to illustrate the statistical characteristics of the

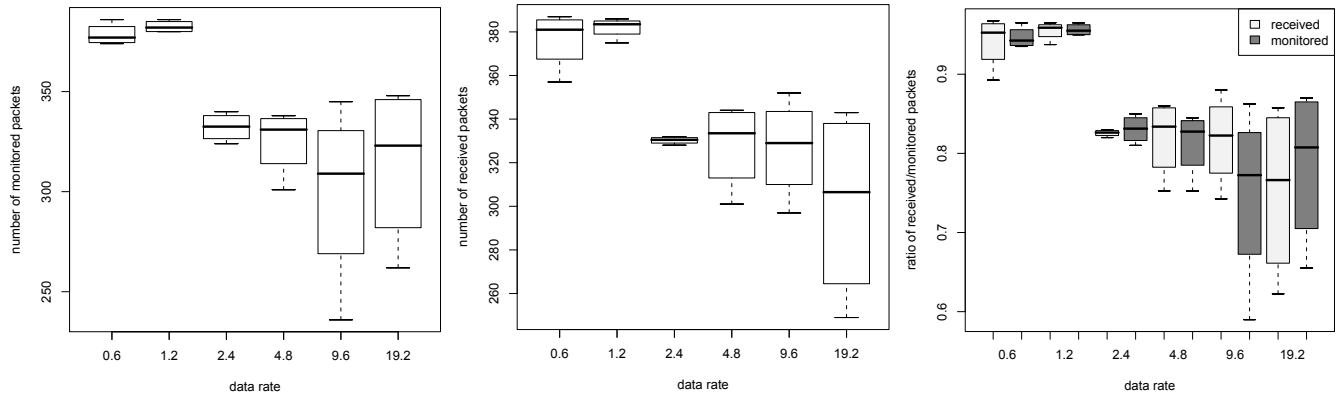

Fig. 10. Results from test of different data rates. Shown are the number of received packets at the monitoring node (left) and at the sink node (middle). Additionally, the ratio of successfully received packets is shown (right)

multiple runs of this experiment. Shown are the number of received packets at the monitoring node (left) and at the sink node (middle). This comparison allows to determine the ability of the sensor node to cope with the increased complexity of the monitoring program compared to a simple sink node. Additionally, the ratio of successfully received packets is shown (right).

Analyzing the results, it becomes obvious that the monitoring node is able to process almost all radio packets for data rates less that 2.4 kbit/s. For higher rates, the reception ratio decreases and the variance of this ratio increases. This effect results from the processing demands in the sensor node. A receiving thread must complete its work on the current packet before the transceiver can receive the next one. A second effect is the high collision probability with those high rates as four nodes are sending simultaneously.

In summary, we are quite convinced with the achieved results are they already allow to perform high quality debugging in typical sensor networks.

VI. Conclusion

In this paper, we presented *Pimoto*, a distributed passive monitoring system developed particularly for Wireless Sensor Networks. The primary objectives were to intercept radio data packets in a completely passive way. Additionally, we wanted to support distributed monitoring in a hierarchical way.

Pimoto visualizes the packets collected by several monitoring nodes, which transmitted the monitoring data over a hierarchically operating system to a central server for further analysis. In particular, the packets are sniffed at MAC level by sensor nodes that employ a second radio channel, i.e. a Bluetooth interface, for communication with a gateway PC. This gateway in turn sends the received data to the server. In our system we use the concept of "monitoring islands", which has the advantage of monitoring several networks simultaneously by multiple monitoring nodes and, perhaps, multiple gateways. At the server, we use a Wireshark plugin for protocol analysis. This tools allows to analyze and visualize the packet contents.

In conclusion, it can be said that the developed toolkit allows to perform the envisioned management and control tasks, which are usually required in WSNs. Protocol development as well as failure detection are simplified by means of graphical protocol analysis.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Elsevier Computer Networks*, vol. 38, pp. 393–422, 2002.

[2] D. Estrin, D. Culler, K. Pister, and G. S. Sukhatme, "Connecting the Physical World with Pervasive Networks," *IEEE Pervasive Computing*, vol. 1, no. 1, pp. 59–69, January 2002.

[3] V. Handziski, A. Köpke, A. Willig, and A. Wolisz, "TWIST: A Scalable and Reconfigurable Testbed for Wireless Indoor Experiments with Sensor Networks," in *7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (ACM Mobihoc 2006): 2nd ACM International Workshop on Multi-hop Ad Hoc Networks: from theory to reality 2006 (ACM REALMAN 2006)*, Florence, Italy, May 2006, pp. 63–70.

[4] G. Tolle and D. Culler, "Design of an Application-Cooperative Management System for Wireless Sensor Networks," in *2nd European Workshop on Wireless Sensor Networks (EWSN)*, Istanbul, Turkey, January/February 2005, pp. 121–132.

[5] N. Ramanathan, E. Kohler, L. Girod, and D. Estrin, "Sympathy: A Debugging System for Sensor Networks," in *1st IEEE Workshop on Embedded Networked Sensors (EmNetS-I)*, Tampa, Florida, November 2004, pp. 554–555.

[6] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, "Sympathy for the Sensor Network Debugger," in *3rd ACM Conference on Embedded Networked Sensor Systems (ACM SenSys 2005)*, San Diego, California, November 2005, pp. 255–267.

[7] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Analyzing the MAC-level Behavior of Wireless Networks in the Wild," in *ACM SIGCOMM 2006*, Pisa, Italy, October 2006, pp. 75–86.

[8] H. Ritter, R. Winter, and J. Schiller, "A Partition Detection System for Mobile Ad-Hoc Networks," in *First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON 2004)*, Santa Clara, California, October 2004, pp. 489–497.

[9] M. Ringwald and K. Römer, "Monitoring and Debugging of Deployed Sensor Networks," in *GI/ITG KuVS Fachgespräch Systemsoftware für Pervasive Computing*, Erlangen, Germany, October 2005.

[10] 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 *4th European Workshop on Wireless Sensor Networks (EWSN)*, vol. LNCS 4373. Berlin, Germany: Springer, January, pp. 195–211.

[11] A. Silberstein, "Push and Pull in Sensor Network Query Processing," in *Southeast Workshop on Data and Information Management (SWDIM '06)*, Raleigh, North Carolina, March 2006.