Project Number:	***INFSO-ICT-224282***
Project Title:	GINSENG-Performance Control in Wireless Sensor Networks
Project Report Number:	224282/UCC/D2.4
Contractual Date of Delivery to CEC:	Month 30 – February 2011
Actual Date of Delivery to CEC:	
Title of Report:	Refined Methods and Tools for Performance Debugging
WP contributing to the Report:	WP2

Abstract:

This deliverable updates D2.2 with refinements and additions on the subject of performance debugging in GINSENG. The key contributions are the management information base, software for monitoring and management, a centralised passive performance debugging protocol, a distributed in-network performance debugging system, and techniques to aid in experimental performance evaluation. Results from laboratory testbeds are presented, together with a description of the role of performance debugging in the project's second software integration and preliminary evaluation (Milestone 4).

Keywords:

Wireless Sensor Network, Performance, Debugging, Management, Monitoring.

COOPERATION

Executive Summary

This deliverable updates D2.2 with refinements and additions on the subject of performance debugging in GINSENG. The key contributions are the management information base, software for monitoring and management, a centralised passive performance debugging protocol, a distributed in-network performance debugging system, and techniques to aid in experimental performance evaluation.

Performance debugging is a crucial component for applications of Wireless Sensor Networks (WSNs) in critical system environments. Any failure in the network can have potentially catastrophic consequences for the operation of the system involved as well as those in the vicinity. This document presents the current state of the GINSENG contributions for performance debugging. Our contributions explore different facets of the problem and probe the limitations of alternative solutions.

Results from laboratory testbeds are presented, together with a description of the role of performance debugging in the project's second software integration and preliminary evaluation (Milestone 4). The performance debugging is currently deployed on-site in the Sines refinery and the upcoming WP4 deliverable D4.5 will include related preliminary evaluation and analysis.

This research had yielded several peer-reviewed collaborative publications that are referenced in the Deliverable.

List of Contributors

Name	Company	Address	Phone/Fax/E-mail
Tony O'Donovan	UCC	Cork, Ireland	t.odonovan@cs.ucc.ie
Wolf-Bastian Pöttner	TUBS	Braunschweig, Germany	+49 531 391 3265
Zhitao He	SICS	Stockholm, Sweden	zhitao@sics.se
Thiemo Voigt	SICS	Stockholm, Sweden	thiemo@sics.se
Cormac Sreenan	UCC	Cork, Ireland	cjs@cs.ucc.ie

Document Approval

	Name	Address	Date
Approved by WP Leader	Adam Dunkels	SICS	14/2/2011
Approved by Quality Control Committee Member 1	Vasos Vassiliou	UCY	17/2/2011
Approved by Quality Control Committee Member 2	J. Sá Silva P. Furtado	FCTUC	20/2/2011

Table of Contents

List of Figures

List of Tables

List of Acronyms

ACK	Acknowledgement
API	Application Programming Interface
CCA	Clear Channel Assessment
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
dB	Decibel
dBm	Decibel Milliwatt
FTDI	Future Technology Devices International
GUI	Graphical User Interface
HWID	Hardware ID
IEEE	Institute of Electrical and Electronics Engineers
IPv6	Internet Protocol version 6
ISM	Industrial, Scientific and Medical
MAC	Media Access Control
MCU	Microcontroller Unit
MI	Mutual Information
MIB	Management Information Base
PAR	Photo-Active Radiation
PC	Personal Computer
RAM	Random-Access Memory
RFS	Ring File System
RSSI	Received Signal Strength Indication
RX	Receive
SFD	Start-of-Frame Delimiter
SLIP	Serial Line Internet Protocol
SNIR	Signal-to-Noise and Interference Ratio
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TSR	Total Solar Radiation
TX	Transmit
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
WSN	Wireless Sensor Network
XML	Extensible Mark-up Language

1. Introduction

Performance debugging is a crucial component for applications of Wireless Sensor Networks (WSNs) in critical system environments. Any failure in the network can have potentially catastrophic consequences for the operation of the system involved as well as those in the vicinity. This document presents the current state of the GINSENG contributions for performance debugging, building on the earlier deliverable D2.2.

There are five core contributions, corresponding to the main sections of the document:

- Management Information Base – in which we present the revised on-node database that is used to store performance-related statistics and information;
- Management and Monitoring – where we describe back-end software infrastructure to transport and process debugging information;
- Sink-based detection and diagnosis system – a passive centralised approach to performance debugging;
- In-network debugging of performance problems – an alternative that is based on the use of on-node storage and processing;
- Support for experimental evaluation – tools and techniques to aid in system performance testing.

2. Management Information Base (MIB)

A management information base is a database that can be utilised when managing devices in a network. Database entries form a hierarchy and are addressed using object identifiers. There are many standardised MIBs used extensively in communication networks, both public and private. SNMP (Simple Network Management Protocol) is a widely supported communication protocol that allows management of a wide variety of devices such as consoles, routers, switches and gateways through the use of one such MIB. Each SNMP managed device holds a MIB that can be accessed and queried using a specified set of commands.

The GINSENG MIB is an on-node database that contains information on all aspects of the sensor node's operations and can be referenced and updated from any of the separate modules on the node that comprise the GINSENG system. In addition to collecting data for monitoring purposes the MIB is used by GinMAC, topology control and overload control. It is divided into several parts with each containing a subset of related elements.

- Node
  The node section is the largest and contains data related to the node's state and operation. Elements in this section can be used to determine much of the node's current and previous states including the node's uptime, how long it has been connected to its current parent, how many frames it has sent and received, its average queue length and details of radio transmission success, failures and utilisation.
- Packet Statistics

  This section keeps track of statistics concerning packets sent and received, both unicast and broadcast, and whether or not communication was successful. It also differentiates between packets generated by the node or addressed to it and those that were forwarded on behalf of other nodes. The data recorded is categorised by packet type so we can determine for example how much of the node's communication is related to control overhead or performance debugging.

- Neighbour

  The neighbour section holds data related to the node's neighbours and includes parent and children. The parent is a special case and is always the first neighbour allowing easier access to modules like topology and routing to allow, for example, finding a new parent node if required.

- Time

  Given the nature of the GINSENG application and the time constraints it is required to operate under, the time section of the MIB is crucial throughout the node's system. It is used for scheduling tasks such as sampling data, listening for incoming messages and transmitting outgoing messages and time-stamping events and messages.

For the First Software Integration (Milestone 2) the MIB was used as a repository for the performance related data that was periodically sent to the sink. Since then it has been expanded considerably and is used by modules throughout the GINSENG system. The Second Software Integration (Milestone 4) still uses the MIB for performance monitoring purposes; the periodic performance related messages sent to the sink are populated with data from the MIB. This data is now far more detailed, especially with respect to the packet statistics and the nature of the packets sent and received by the node. For Milestone 4 many of the different modules in the GINSENG system utilise the MIB, for example, topology control tests data obtained from the MIB against various thresholds to determine if a link has become unstable and a new parent is required.

3. Management and Monitoring

3.1 Reliable PC WSN interconnection

The goal of GINSENG is to create a performance controlled WSN. Originally, the intention was to use IPv6 on top of GinMAC in order to provide transparent connectivity between middleware components and the sensor nodes. However, due to severe program memory limitations on the sensor nodes it was decided to operate without IPv6. Therefore, the need for a reliable and timely interconnection between the sensor network and the backend components such as the middleware arose. This section presents the architecture and the interfaces of the interconnection mechanism that is used in GINSENG.

3.1.1 System Architecture

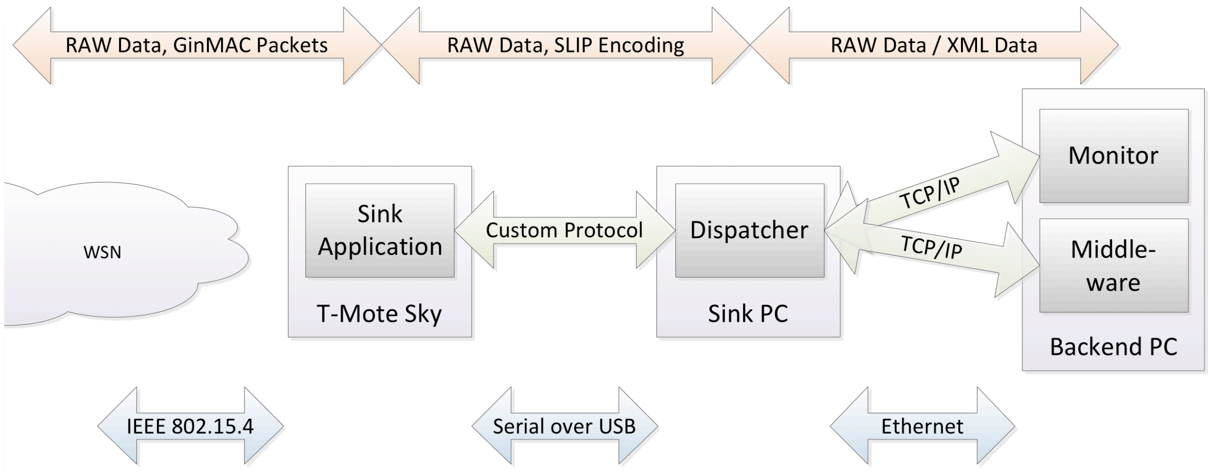


Figure 1: Architecture Overview

Figure 1 shows the interconnection architecture. The sink node is a standard T-Mote Sky running the sink application. This application receives data from GinMAC, packages this data into special frame containers and sends it to the PC. Also, the PC can send such frames and the sink application running on the node will hand over the data to GinMAC which takes care of delivering the data to nodes in the field.

The dispatcher is software running on the sink PC. It receives RAW data packets from the sink node and converts this information into XML. The XML as well as the RAW packets are made available to other interested parties over TCP. The dispatcher opens several TCP ports to allow other applications to send and receive data to and from the sensor network.

It is important to note, that the serial connection between the sink node and the sink PC is not reliable - bytes could be lost or modified. Also, timing on the sink node is extremely critical, since it is the synchronisation source for all other nodes in the network. Subsequently, the communication protocol between the sink node and the sink PC has to ensure reliability. Also, the PC has to respect the time constraints of the sink node. In addition, packets have to be delivered to the PC in time to reflect the real-time character of GINSENG and all packets need a precise timestamp.

3.1.2 Communication Protocol

The wired data communication between the sink node and the sink PC is realized using the USB port of the T-Mote Sky. On the PC side, this creates a virtual serial port and terminates in an FTDI USB-Serial convertor on the node. The outputs of the FTDI chip are connected to the MSP430 MCU on the T-Mote Sky.

3.1.2.1 Data Format

To allow packet-based communication over the stream-based serial port, SLIP encoding [1] is used to mark boundaries of packets and to synchronize communication.

Inside the SLIP encoding, the following packet types are defined:

- DATA
  Data packets contain a GinMAC packet and additional meta-data such as source ID, timestamp, type, number of hops, sequence number and a CRC-16 checksum. The sender of data packets waits for an ACK packet in return. If such an ACK packet is not received, the DATA packet will be retransmitted.

- ACK
  Acknowledgement packets confirm the successful reception and CRC verification of a DATA packet.
- PRINT
  Print packets only flow from sink node to sink PC and contain information that is to be printed on the console of the sink PC to allow debugging of the sink node. Print messages only use a type field and the message to print and avoid additional overhead such as a checksum.
- XON
  XON messages are part of the flow control mechanism and are sent from the sink node to the sink PC to signal that the PC may now send data to the sink node.
- BEACON
  Beacon message are only sent from sink node to sink PC and mark the beginning of a new GINSENG epoch on the sink node. This message is used for time synchronisation purposes.

In summary, to ensure that data packets that get corrupted during transmission are detected, a CRC-16 code is appended to all except print messages. To ensure the successful reception of DATA messages the receiver has to issue an ACK message for the sequence number of each received DATA message. DATA messages are retransmitted after a timeout. Lost XON and BEACON messages will not be detected, but will be periodically (re-)transmitted anyway.

3.1.2.2 *Timing and Flow Control*

Timing on the sink node is extremely critical since it is the time synchronisation source for the whole network. Incoming serial data create interrupts that may disturb the timing, so the sink PC may only send data when the sink node is idle. Consequently, the flow control on the serial line is implemented in a master-slave manner. The sink node is the master that dictates timing and the PC has to adhere to it.

The sink node stores incoming data from the sensor network in a buffer. Whenever the sink node is idle, all new packets will be transmitted to the sink PC. The sink node constantly checks if enough time is left until the next active slot and interrupts transmission if time is running out. If all new data messages have been transmitted, old and unacknowledged messages that have exceeded a timeout will be retransmitted to the sink. Subsequently, if there is time until the next active slot left, a XON message will be transmitted to the sink PC. This message includes the time that is left until the next active slot and the number of buffers that are currently remaining. Now, the sink PC starts to transmit DATA and ACK packets to the sink node. The sink node buffers incoming data packets and only acknowledges them to the PC when the packets have been successfully handed over to GinMAC.

3.1.2.3 *Overflow Handling*

Since the available memory on the sink node is limited, the buffer may overflow if the sink PC is not responding or when the serial communication is too unreliable. Depending on the underlying GinMAC schedule, the reporting frequency and the number of nodes in the network, it is also possible that the capacity of the serial connection will be insufficient. To handle such overload situations, old packets have to be deleted. Packets coming from the PC that have not been acknowledged will be deleted first. These packets will be retransmitted anyway and deleting them is no loss. If still no buffers are left and data is constantly coming in from the sensor network, data packets that have been transmitted to the PC are deleted. These deletions start with the lowest sequence number since these have been (re)transmitted most often and have likely been received on the PC side. If all buffers are still occupied, packets from the sensor network that have not been sent to the PC will be deleted. These are packet losses and the network is thus overloaded.

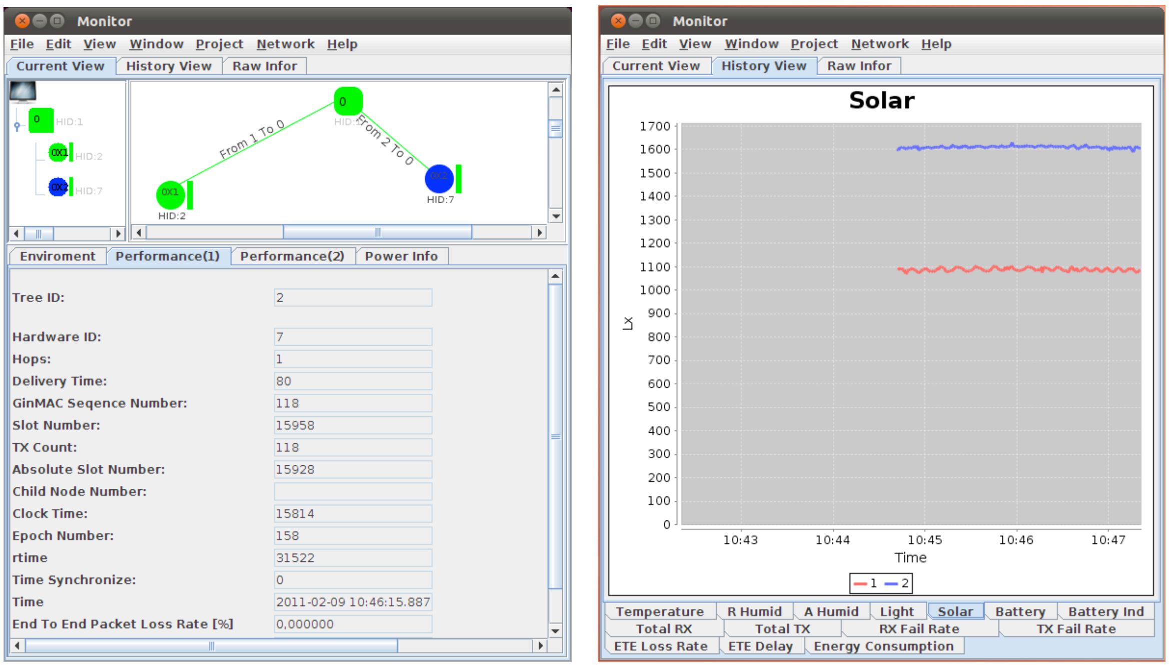
3.1.3 Dispatcher

The dispatcher is a Java application that runs under Windows and Linux operating systems. It connects to the sink node and handles the serial communication protocol. Incoming data is CRC-checked and discarded if the checksum does not match. Also, duplicate data packets are identified and dropped. All remaining data packets are interpreted according to their format. If the format is known, the information is extracted and transformed into XML data and augmented with meta-information such as a timestamp. Otherwise, the packet remains in RAW representation. Data packets are available on three different TCP server ports: RAW data, XML string and XML object data.

External applications that need information from the sensor network can connect to any of these ports to get the specific format. A *packetresolver* library is provided to enable third party applications to interpret incoming packets. If external applications want to send information to the sensor network, the data has to be packaged using the *packetresolver* library and then sent on to the dispatcher. The data is then forwarded to the sink node by the dispatcher and from there on to the receiver node. Data to the sink node is also acknowledged; if the dispatcher does not receive an ACK after a timeout, the DATA message will be retransmitted.

To allow an offline analysis of data that was received by the dispatcher, all incoming data will be written to XML-based log files. These files are rotated frequently to limit their size and allow inspecting historic development of certain values.

3.2 Monitor



Tree ID:	2
Hardware ID:	7
Hops:	1
Delivery Time:	80
GinMAC Seqence Number:	118
Slot Number:	15958
TX Count:	118
Absolute Slot Number:	15928
Child Node Number:	
Clock Time:	15814
Epoch Number:	158
rtime	31522
Time Synchronize:	0
Time	2011-02-09 10:46:15.887
End To End Packet Loss Rate [%]	0.000000

Figure 2: Screenshots of the monitor: Performance info on the left, history view on the right.

As outlined in Figure 1, the monitor is a software component that runs on a PC. It communicates with the Dispatcher via TCP/IP and offers a graphical user interface (GUI) to the user. Data coming in from the sensor network is analysed and presented to the user. Also, the monitor offers options to send raw data directly to the sensor nodes.

The goal of the monitor is to provide a tool to network operators to locate and quantify problems in the network. This tool should be intuitive to use and help to track down the cause of the problems. In a later stage, it may even be extended to configure parameters in the network, thereby working around existing problems.

Currently, GINSENG nodes send a variety of information to the sink for live processing and archival purposes. The monitor displays the sent by the nodes and creates a context between them. It also reconstructs the current tree structure of the network based on information received from the nodes. A full 3-2-1 tree visualisation can be seen in Figure 3. Also, Figure 2 shows two screenshots of the monitor. On the left node-specific performance-related data is displayed. This information can help to allow an in-depth view of the situation of a single node. On the right, a graph based history view for all nodes in the network is shown. This view allows comparing the current parameters of different nodes. In addition, the view allows looking at the recent history of this specific value.

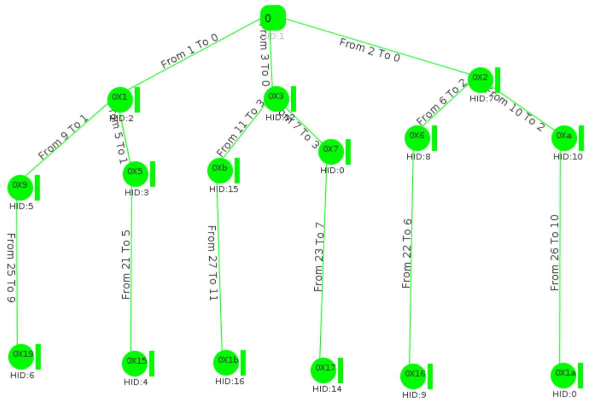


Figure 3: Live visualisation of a 3-2-1 tree.

4. Sink based detection and diagnosis tool

PerDB is a sink-based performance debugging system developed at GINSENG partner UCC and its initial design [14] was previously described in Deliverable D2.2. Since then the system has evolved [17] and PerDB has been implemented for the GINSENG system, integrating closely with GinMAC and other components. Thus we include a more complete description here.

4.1 System Design

PerDB offers several aspects that enable lightweight anomaly detection and some automatic diagnosis. It also includes a probing mechanism to allow the network operator request supplementary data should it be required for a diagnosis. An outline of each of these components follows. The PerDB approach is centralised, relying on back-end performance debugging software and receipt of periodic in-band debugging information from nodes. This is in contrast to the in-network approach presented later in this document.

4.2 Data Capture

One of the main assumptions of the GINSENG scenario is that the network is planned and that information is available for each node. What is known about each node is used to build a node profile. The profiles are contained in an XML file that is loaded at start-up and includes data specific to each node such as packet loss threshold and delivery delay threshold. The node profiles are added to the system's debug table. The debug table holds all node profiles and is polled regularly for anomalies.

The system connects to the dispatcher (described in Section 3.1.3) and reads its XML stream. This XML is parsed and the appropriate node profile is updated with the information obtained. After a message is received a scan of the debug table is initiated to detect any new anomalies that may have been introduced by the update.

4.3 Analysis Protocol

4.3.1 Detection

The node profiles contained in the debug table are scanned periodically and upon receipt of messages via the dispatcher. Each node profile in the table is checked and any thresholds that have been exceeded are flagged as anomalies.

There are three types of exception that can be detected by PerDB; these are latency, packet loss and disconnection. Latency is a measure of the time it takes a packet to travel from source to sink. Packet loss denotes the percentage of packets lost out of the total amount sent. Disconnection indicates that a node has not been heard from in some time and has become disconnected from the network.

Each of the metrics described above can be calculated using a small number of fields 'piggybacked' in each application message. These fields are:

- The transmitted packets sequence number
- The number of hops taken by the transmitted packet to reach the sink
- The timestamp the message was generated by the source node
- The timestamp the message was received at the sink node

Given the timestamp the message was generated at the source and the timestamp the message was received at the sink it is possible to calculate the latency of the message. This value is checked against the latency threshold in the node's profile and if the threshold is exceeded then an alert is generated. Similarly packet losses can be calculated using the sequence number and the count of messages received and is checked against the packet loss threshold. Finally the disconnection exception is based on the time since the last packet was received from the node.

All exceptions are flagged and the network operator is notified. Due to the connected nature of the network if there are multiple exceptions then it is likely that they are related. If several neighbouring nodes exhibit similar anomalous behaviour there is probably a common cause that once remedied will resolve all anomalies. With this in mind alerts are ordered by ascending hopcount since a problem with the closest node could be the cause of problems in its descendants and fixing the parent could also fix the child. Once a problem has been detected then diagnosis is attempted.

4.3.2 Diagnosis

PerDB contains some basic automatic diagnosis functionality that works by matching the information currently available and the most common causes of performance problems. It works as follows:

If the issue is disconnection:

- Check the number of disconnection alerts. If only one alert is present, it can be assumed that the node is a leaf node. This will only affect data coming from this node. A probe could be sent to the parent node requesting data on uptime and battery level to provide an insight into why the node has disconnected.
- If more than one alert is present, for each node with an alert we check if one of the other alerts concerns a parent. If so, it can be assumed that child nodes with disconnection alerts can be ignored until the parent is reconnected, as the child nodes may not have an alternative path.

If the issue is latency:

- Check the query node for its expected hops per packet. Compare this to the most recent packet's number of hops. If the expected number of hops does not equal the recent number of hops, a route topology change has occurred introducing extra delivery latency.

If the issue is packet loss:

- Check for congestion issues. Retrieve the number of packets received in total from the node profile (every time a packet is created and added to the node profiles list of packets the number of packets received is increased). Retrieve the number of packets received for a neighbouring node profile. Compare the two values. If the problem node has a higher value, the node is sending packets at a higher rate and causing congestion. If it is lower or equal, then congestion is not an issue. Since GinMAC is TDMA based all nodes should have a similar received packet count stored in the node profile, as all nodes are expected to transmit according to a shared schedule.
- Check for a node reboot. If a node reboots while in operation, the sequence number of its packets resets also. The difference between the last sequence and the first sequence after reboot is treated as a packet loss alert. A query is sent to the alert node with the uptime as a query. As the packet travels to the destination node, it broadcasts the query to nodes encountered along the path. These nodes return their uptime along with the alert node. The alert node's uptime is then compared to the average alert time of the parent nodes. If it is less, a node reboot is the diagnosis. The node uptime and parent average uptime are also presented to the user, so they may decide to continue the debugging process if desired.

Once recommendations have been made, the users are asked if they would like to continue debugging the node. If so, they are presented with the debug data once again and may send another probe. If not, the diagnosis will move onto the next alert and the previous alert is removed. The node associated with the alert is reset and monitoring begins again.

To fully automate the diagnosis process, a priority level system for alerts would have to be developed. This is to ensure that the problem being diagnosed is currently the most important issue affecting the network. For example, disconnection of nodes may be granted priority if the node is a parent node. However if a leaf node is disconnected and simultaneously a parent node close to the sink is flagged for massive packet loss, the algorithm must have a method to decide which issue to diagnose first. The leaf node, although disconnected, will not have a detrimental effect on the network as a whole, whereas a parent node suffering from massive packet loss could incur massive performance degradation for the entire network. Once the issue is chosen, the algorithm would proceed with the steps explained above to diagnose the issue.

When implementing the automated diagnosis process, the issue of automatic probe request generation is relevant. For each issue detected, a standard set of probe requests could be developed. With data returned from each probe request, non-causes can be eliminated, narrowing the possibilities of the cause. For example, a decision diagram approach could be applied. Taking the node reboot case from above as an example, the uptime for each node could quite easily be collected and compared. A new set of thresholds could be defined that indicate the amount the uptime of a node is allowed to deviate from the average of the nodes along its path to the sink. Automatic diagnosis and probe generation is ongoing work.

4.4 Probing Protocol

If an issue is detected by the system, a complete diagnosis may not be possible with the data available at the sink. A probe message can be sent to the affected node in order to obtain any additional information that is required. Since there is a limit on the size of the response that can be returned for a query and the nature of each performance problem requires different data for diagnosis, it is desirable to have a customisable query system. In order to achieve this we employ a bitmask to specify the desired result set. Each bit of the bitmask represents a particular element in the MIB, as shown in Table 1. Currently the bitmask uses only a single byte that is limited to specifying 8 elements. This size can be increased in the future to allow specification of a more comprehensive result set.

Query	Bit
Uptime	1
Neighbours	2
RSSI	3
Retransmissions	4
Packets Sent	5
Packets Received	6
Battery level	7
Hardware ID	8

Table 1: PerDB Probe Bitmask

The network operator specifies the desired result set for the given query. If an element of the MIB is included in the request then its corresponding bit is set to 1, otherwise it is set to 0. For example a query for Uptime, RSSI and Hardware ID would result in a bitmask of 10000101, as shown in Table 2.

HWID	Battery	Packets Received	Packet Sent	Retransmissions	RSSI	Neighbours	Uptime
1	0	0	0	0	1	0	1

Table 2: PerDB Probe Bitmask example

In addition to the required result set there are several other fields in the message format sequence. This sequence represents the action a node should take and is described below:

- 1 bit representing whether aggregation should be performed by nodes on a return probe response or not.
- 7 bits representing the repeat period of the probe response. The repeat period states whether the probe response should be returned at a set interval from a node. For example, if a node is experiencing intermittent interference causing latency delays, a single probe message may not return information that leads to a conclusive

diagnosis. However, if the RSSI levels of the node were queried and returned multiple responses at regular intervals, it may provide more insight to the issue.

- 2 bytes representing the address to which the probe message should be sent. The address can be that of an individual node with performance problems, one of its neighbours or a broadcast address can be used to query all nodes in the network. The broadcast request should be used sparingly or when no other option is available due to the amount of traffic overhead generated on the network in response.
- 1 byte representing a bit mask. This value of this represents which properties of the node should be queried.

When a probe message is received at a target node, the query data must be gathered and a response message sent to the sink. The node checks the value of each bit of the request bitmask and adds the corresponding MIB element to the response message if it is set (bit=1).

When a probe response is returned to the sink each forwarding node attaches its address to the response message payload. Since the bitmask is known, the size of the returned data in the payload can be calculated. The remainder of the payload is the path that the response message took back to the sink. This allows further directed probing of nodes along the path if additional information is required.

5. In-network Debugging of Performance Problems

We have researched and published work on the feasibility of using a node's external Flash to log information concerning the node and its neighbours. There are bandwidth and throughput constraints inherent in WSNs that limit the amount of meta-data that can be sent over the radio to the sink. Instead of disregarding this data, logging it to the node's onboard storage can help with debugging of performance problems. We investigate the feasibility of using such a system and compare the cost of using Flash to sending data over the radio. A separate investigation and modified file system are also presented in this section. The in-network debugging carried out for the Second Software Integration (Milestone 4) is also described.

5.1 Introduction

Environmental conditions, component failure and programming error can all cause problems in deployed sensor networks making them resulting in systems that are error-prone and often perform poorly. Hence, a variety of debugging tools have been developed to help researchers to understand faults. Such tools commonly require that state be sent to a sink, either at regular intervals or upon a query from a network operator.

During development providing a detailed logging layer can be very helpful in detecting and diagnosing problems with the system. However development and early testing takes place in in-house test beds where the events can be reported over a serial line. In contrast, real deployments are limited to delivering debug data over radio.

The problem with mixing debug packets and ordinary packets in the network is that 1) the packets may not get through in case of bad performance, 2) "heisenbugs" may be introduced, and 3) it is limited to low-fidelity data. Although frequent sampling and reporting gives a more accurate picture of the network, it requires extra traffic. In multi-hop networks, this cost can be considerable. Given a fixed and often tight energy budget for fault diagnosis, only a limited number of variables can be reported to the sink regularly, and only at moderate rates.

Instead of this periodic reporting approach we investigate the idea of keeping detailed logs for each node but storing them on the node's external Flash instead of sending over the radio. Using the onboard Flash along with Contiki's Coffee file system allows more comprehensive data to be kept than could possibly be transmitted to the sink.

By logging sensor values together with performance statistics, we enable correlation of arbitrary data sets, self-monitoring, and detailed diagnosis of performance anomalies. Earlier research has uncovered unexpected correlations between performance and different types of environmental characteristics, or hardware failures. Boano et al. [2] have shown that the temperature has a significant effect on the packet reception rate (PRR), even inside an office building. Finne and Eriksson observed that radio communication triggered sensor readings, causing superfluous alarms [3].

5.2 Design Overview

There are three distinct elements to the storage-centric approach we devised. Firstly, the storage itself, which relates to what data is stored, how frequently, and for how long it should be kept. Secondly, the computational capabilities of the motes are employed to perform some statistical analysis, allowing the nodes to detect and possibly even diagnose performance problems. Finally, data collection involves the methods available to the network operator for accessing the stored data, whether it is only a summary, a subset, or all of the information that is required.

5.2.1 Storage

A simple, yet essential point of our work is to exploit the node-local storage that exists in many types of motes. Unlike traditional debugging tools that mainly support queries for live data only, our approach supports analysis of detailed data collected over a long time. For this purpose, we use the Coffee file system [4] in Contiki to store vast amounts of sensor data and networking statistics in a circular log. Coffee is able to append data in files at a speed close to that of the underlying Flash driver, and uses a constant RAM footprint for each file, regardless of size.

The online logs accommodate arbitrary metrics that are relevant to the performance. Each mote is able to scan and analyse its own performance, sending only statistical summaries or warning messages back to the sink. After inspecting these summaries the network operator can decide to download more detailed data from a node that reports anomalous performance. The implication is that analysis of the network performance is significantly less likely to affect the behaviour of the network. Moreover, even without having some form of analysis online, the stored data is useful in post-deployment analysis of performance, effectively removing the need for any debug-data messages to be sent.

5.2.2 Statistical Analysis

Diagnosis of a performance problem requires analysis of the data available to determine the cause. Given the information stored on the motes and their computational capabilities, many techniques are possible. Examples include finding the maximum, minimum, or average of a variable–such as RSSI–over a specified period, or searching for deviations in the logs. Another option is to examine different sets of events looking for any correlation between them. In order to illustrate the type of online analysis possible, we have implemented two functions on the motes for calculating such correlations using standard statistical methods. The first function is Pearson's correlation coefficient r_{xy}, between the sets x and y as shown in Equation 1.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n} XY(iy_i - \overline{y})^2}}$$

Equation 1: Pearson's correlation coefficient

In this equation, x_i and y_i are the i-th measurements of two series of n measurements of the random variables X and Y. $\overline{x}$ and $\overline{y}$ denote the sample means of X and Y. Using a medium effect size of 0.3 and alpha error 0.05, an estimated 84 records or more are needed to detect a significant correlation (with 80% power).

Our implementation of this function retrieves the data series from the Coffee file system and computes the correlation. We implement Pearson's correlation coefficient as an iterative algorithm that reads one sample and computes one step. By only storing a small number of intermediate data samples, we are able to keep the memory footprint of this function low. We need to make two passes over all samples: the first one to compute the average and the second for the sums in the equations.

Pearson's correlation coefficient is widely used but cannot compute the correlation between a binary and a continuous variable. Therefore, we have also implemented an algorithm to compute the mutual information $I(X;Y)$ between a discrete variable X and a continuous random variable Y. $I(X;Y)$ is also a more generic correlation measure than Pearson's correlation coefficient.

$$I(X;Y)\frac{1}{160} = \frac{1}{2}\log(2\pi e\sigma^2) - \frac{1}{2}\sum_{x\in X} P(x)\log(2\pi e\sigma_x^2)$$

Equation 2: Mutual Information

We compute mutual information as depicted in Equation 2. In this equation, σ^2 is the variance of the continuous variable Y and σ_x^2 is the variance of Y given that $X = x$. The implementation of the algorithm is similar to the one for Pearson's coefficient described above in that we need to make two passes over the data. The first pass computes the averages and the second pass computes the variances.

5.2.3 Data Collection

Performance anomalies have many causes, as already outlined, that can result in a variety of problems. For example, congestion may result in increased delivery delays and intermittent connectivity, whereas a node failure will result in a disconnection from the affected node and possibly any nodes that use it to forward messages. As a result several methods for accessing the stored data are required.

When there is no physical access to a deployed network, the preferred method is remote querying. The network operator is able to request and receive data summaries from selected nodes in the network. Upon receipt of such a query, the node performs the requested calculation, returning the results required. The correlation procedures described in Section 5.2.2 are examples of this. Since only summaries are sent, this method has little impact on the network.

Alternatively the operator can take a mobile sink (e.g. a laptop or a tablet) into the field to query nodes directly. For partitioned networks this may be the only option available. This method has a smaller response delay and a lower overhead since all communication is single-hop, making it suitable for diagnosis isolated to a small section of the network.

To allow for deeper analysis than what is practical on the nodes, all stored data can be downloaded in a batch transfer over the radio or using a node's UART bus. While a UART download has no impact on network operation, it does require a physical connection to the node. A batch download over the radio is best done using a separate channel or a mobile sink to avoid forwarding the data over the network.

5.3 Feasibility Evaluation

To evaluate our storage-centric approach to performance debugging, we quantify the energy required to store information locally on a node in comparison to sending it via the radio. We also use the motes to efficiently calculate statistical correlation among events. Finally, we collect data from a tested to demonstrate typical observations that are possible using storage-centric performance analysis.

5.3.1 Energy Analysis

In this experiment, we compare the cost of logging data locally with that of transmitting it over the radio. Our setup consists of a set of emulated Tmote Sky motes in the cycle accurate Cooja/MSPsim simulator [5]. Each log record is 64 bytes, comprising debug information from several parts of the system.

We run the Contiki operating system on the motes, and use the Coffee file system and the unicast primitive in Rime [6]. The MAC protocol under Rime is X-MAC, configured with a 1.25% duty cycle. The duty cycle is selected using the default radio on-time of $TX_{\min}$ s, and an off-time of 0.5s, which is half of the average packet transmission interval. We send 1,000 packets and measure the per-packet energy consumption using Contiki's Compower library.

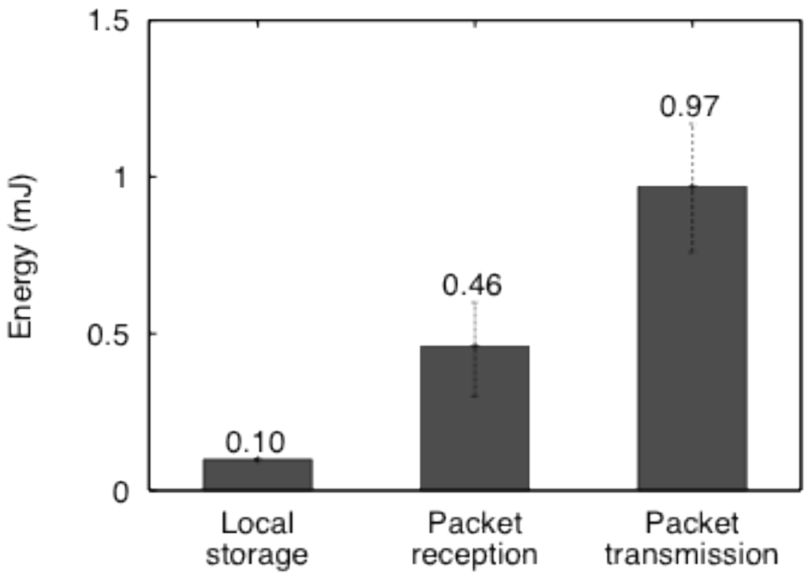


Figure 4: Storing performance data to the Flash requires significantly less energy than sending the data over the radio, even under optimal radio conditions.

Figure 4 shows that the energy consumption is considerably lower for storing the data locally. Whereas the packet costs vary, the cost of storing the data locally is the same since Coffee's optimised file append operation closely follows the performance of the underlying Flash device driver.

We have used an optimal setup for unicast transmissions: a single-hop network without interference. The total cost of transmitting the data from a mote to a sink in a multi-hop network is a multiple of the single-hop cost. Furthermore, the contention and the collision rate in the network will increase significantly if performance debugging packets are transmitted at high rates: our 1,000 packets sent in the network layer generated 3,762 packets in the link layer because of the strobing procedure in X-MAC.

Using Contiki's energy estimation utility, we also characterize the energy overhead of logging over a range of sample sizes [50, 200]. The average overhead is 0.435 μJ per stored byte, with maximum and average estimation errors of 5.3% and 2.1% respectively.

5.3.2 Computing Correlations

The purpose of this experiment is to determine if any correlation exists between the environmental conditions and the minimum transmission power required for successful communication ($TX_{\min}$). We use a data set that we obtained previously from an office testbed[2]. The data consists of $TX_{\min}$ along with samples collected from the nodes' temperature, humidity, and light sensors. Tmote Sky nodes are equipped with two light sensors; a photo-active radiation sensor (PAR) and an ambient light sensor (TSR). The data is measured over 48 hours, during which time-significant fluctuations in the data were observed.

To prepare the calculation, we transfer the data to a mote and store it in a file. We compute the correlation between $TX_{\min}$ and the temperature using Pearson's correlation coefficient using Equation 1. We then compute the mutual information (MI) using Equation 2. In contrast to Pearson's correlation coefficient, MI allows us to compute the correlation between a binary variable and a continuous one.

We use Contiki's power profiler to measure the energy consumption for reading the data from the file system and computing the correlations. We vary the amount of samples used as input to the correlation functions. We have also verified that the computed correlation is correct.

5.3.2.1 Pearson's Correlation

Figure 5 confirms our expectation that the energy consumption increases linearly with the sample size where one sample consists of a $TX_{\min}$ and a temperature value. The figure also shows the power consumption for computing the Pearson's correlation coefficient is low. In particular, the cost of reading the file from Flash is lower than the cost of the CPU activity.

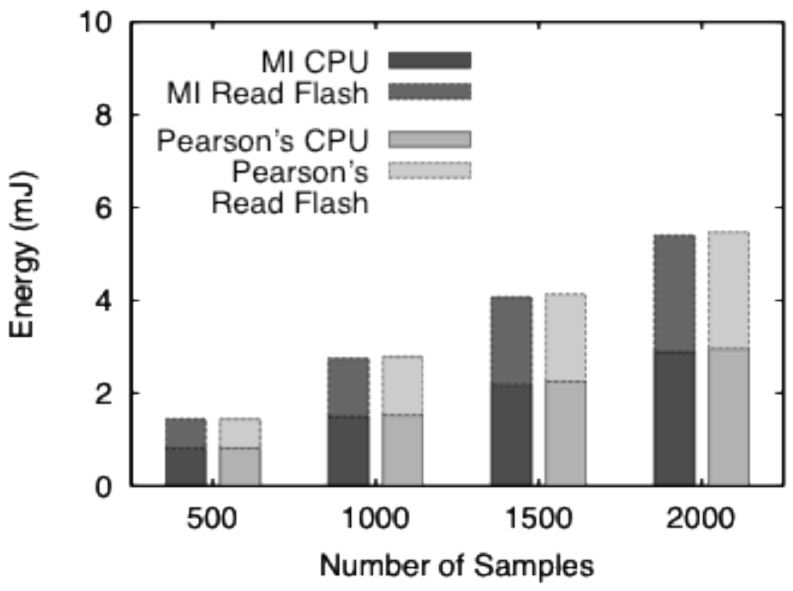


Figure 5: The energy consumption for computing a Pearson's correlation coefficient and mutual information (MI) from locally stored data.

The results of the Pearson's correlation coefficient calculation are shown in Table 3. This value is between 1 and -1, with values approaching ±1 indicating that there is correlation between the data sets. Values close to 0 denote that no correlation exists. The results show that $TX_{\min}$ correlates with the temperature, consistent with the findings by Boano et al. [2]. Incidentally for this deployment, we observe that $TX_{\min}$ also correlates with humidity and light.

Sensor	Sender	Receiver
Temperature	0.779	0.670
Humidity	-0.776	-0.689
TSR	0.641	0.608
PAR	0.640	0.590

Table 3: The calculated Pearson's Correlation Coefficient

5.3.2.2 *Mutual Information*

We use a modified version of the trace in Section 5.3.2.1 to calculate the mutual information. In the trace, we pick a transmission power TX and convert $TX_{\min} > TX$ into a binary variable by setting it to 0 if $TX_{\min} > TX$ and 1 otherwise. As in the previous section we compute the energy consumption. Figure 5 shows that the results are very similar. The reason is that most of the CPU power is used for reading the files and computing the sums. The calculation of mutual information uses slightly less energy because the logarithm is cheaper to compute than the square root.

Reading 2000 samples from a file and applying one of the two correlation functions takes less than 300 ms, and can hence be scheduled without affecting periodic tasks such as sensing and logging. If needed, there is also the option of yielding control of the processor at multiple points within the iterative algorithms.

5.3.3 Packet Transmission vs. Local Analysis

When comparing the results in Figure 4 and Figure 5, we see that reading 500 samples from Flash and computing a correlation requires the same amount of energy as transmitting and receiving one packet over a single hop. The energy consumption of transmitting and receiving four packets, in optimal conditions, is similar to reading 2,000 samples and computing the correlation. These results clearly demonstrate the applicability of our approach.

5.4 Multi Ring File System for Wireless Sensor Nodes

In Wireless Sensor Networks, data is often continuously obtained from sensors. However, due to energy constraints or network capacity limitations, data may only be sent off occasionally. Consequently, sensor data has to be buffered on nodes for later processing or sending as shown below. For this purpose, a ring file (system) that allows reading data from the beginning of a file and appending data to the end of it is helpful.

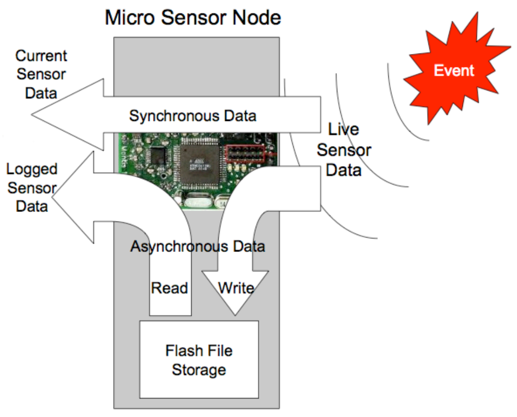


Figure 6: Buffering of continuous sensor data (figure taken from [7]).

Sensor nodes come with a varying amount of Flash memory that can be used for such purpose. The Coffee file system [4] that comes with Contiki supports Flash memory, uses the notion of files and allows read, write and delete operations on such. However, files can only be deleted completely, so that real ring buffer functionality cannot be realized using Coffee. Also, Coffee is not optimized for time-critical systems. In GinMAC, the slot length of 10 ms defines the typical time that is available for data processing and read/write operations.

To overcome the limitations of Coffee, we have created a multi ring file. It provides the ability to create, delete, append, read and write multiple files (rings). It is energy efficient and uses a wear-levelling schema to avoid using certain Flash pages too often.

5.4.1 Coffee Performance Evaluation

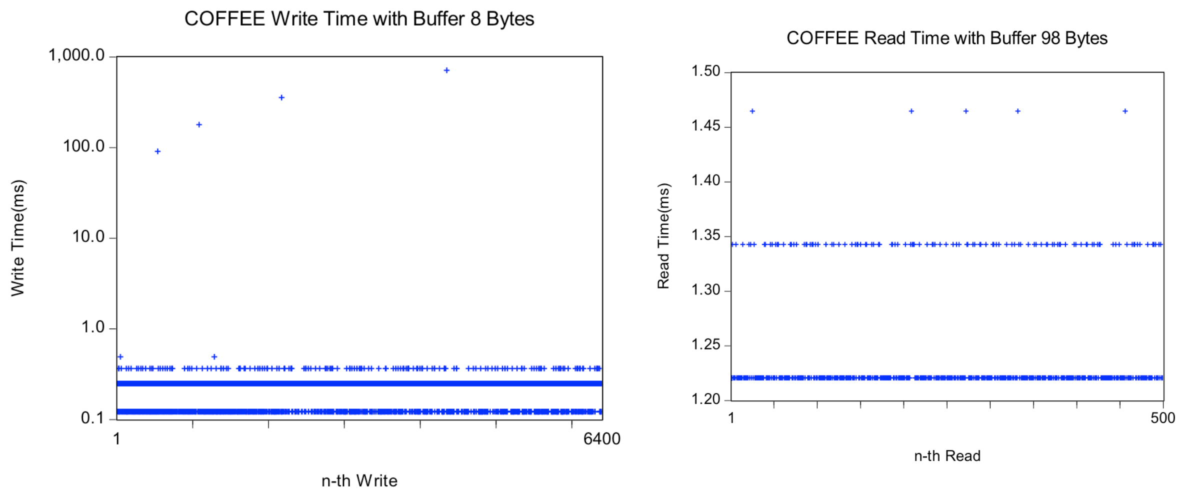


Figure 7: Coffee performance evaluation [8]

As baseline, we have evaluated the performance of Coffee in a typical scenario. Therein, a node repeatedly appends 8 bytes of sensor data to a file. Subsequently, the data is read back in blocks of 98 bytes (one GinMAC packet payload).

The result is plotted in Figure 7. The x-axis shows the number of the read/write operation and the y-axis shows the time it took to complete the operation. Please note that the y-axis of the write graph is logarithmically scaled. The number of read and writes operations differ, because the total amount of data (i.e. the file size) that was written or read is comparable.

The read performance is stable and well within bounds that are acceptable for GINSENG. However, the write performance has some outliers that are caused by a speciality of Coffee. It creates files with a certain size. Whenever the actual file size exceeds that limit during a write operation, Coffee will create a new file with twice the size and copy all data from the old to the new file. This process takes significant amounts of time and exceeds the time that is available during a typical epoch of GinMAC.

5.4.2 Ring File System Design

The multi Ring File System (RFS) is based on concepts published in [7]. It overcomes some limitations of Coffee and provides a similar interface to the application developer. However, by its nature of a ring it can be used as local, temporary data buffer if the amount of incoming sensor cannot be send to the sink in time. Either the data can be sent later (in case of network disruptions) or the data can be pre-processed (and possibly aggregated) on the node to send a summary to the sink after a certain period. This section outlines the design of RFS.

5.4.2.1 Data Organisation

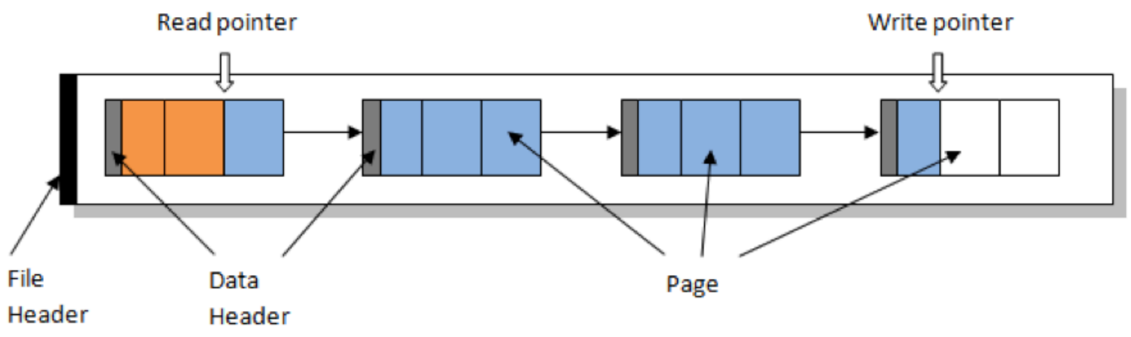


Figure 8: RFS Data Organisation [8]

Physically, Flash is organised in pages that can only be read, written and deleted at once. Overwriting a page is not possible; it has to be deleted first. Consequently, RFS aggregates a configurable amount of pages into one virtual data block. Each data block has a header, which contains meta-information such as the pointer to the next data block. One block is likely to span over multiple pages in Flash. This technique is used to limit the amount of meta-data overhead.

In RFS, a file has a file header that contains meta-information such as the file name, the read and write pointer and also a pointer to the first data block. The actual data of a file may be split into multiple data blocks wherein each data block references the following data block in the file structure. While data block headers are stored together with the blocks in the Flash, the headers of all files are stored centrally in one data block.

The file headers are read from Flash whenever a file is opened. Subsequently, the information is stored in RAM until the file is closed. Then, the information is persistently written to the header block in Flash to survive node reboots.

5.4.2.2 Application Programming Interface

The API of RFS uses the same interface as Coffee in Contiki. Normally, applications that are already using Coffee should also be able to use RFS. However, due to the special characteristics of a ring file system, RFS offers an additional read function. This function will mark data as obsolete after reading, so that the Garbage Collection will eventually free the occupied space. This additional read function should be used when using RFS as a ring buffer. Consequently, the sensing application can use the conventional append function to write data to a file. The processing application can use the ring buffer read function to get data out of the buffer and to process it. However, after reading data from the file once, it is deleted.

5.4.2.3 Garbage Collection and Wear-Levelling

When an application asks to delete a file or when the special ring read function is used, RFS does not delete the Flash page right away. Only when additional space is requested by a write operation and no free pages are left in the Flash, the garbage collection is called. Pages previously marked as obsolete are now freed and can then be used again. To further reduce the risk of long write operations, the garbage collection may also be called proactively whenever the node is idle.

In order to avoid ruining the Flash with too many delete and write operations, RFS uses a simple but effective wear-levelling schema. The last page that was freed by the garbage collection is always referenced in a pointer. Whenever a new page is requested, the referenced page will be used. This allows use of different pages when overwriting old pages.

5.4.3 RFS Performance Evaluation

For RFS performance evaluation, we have used the same scenario as for the evaluation of Coffee.

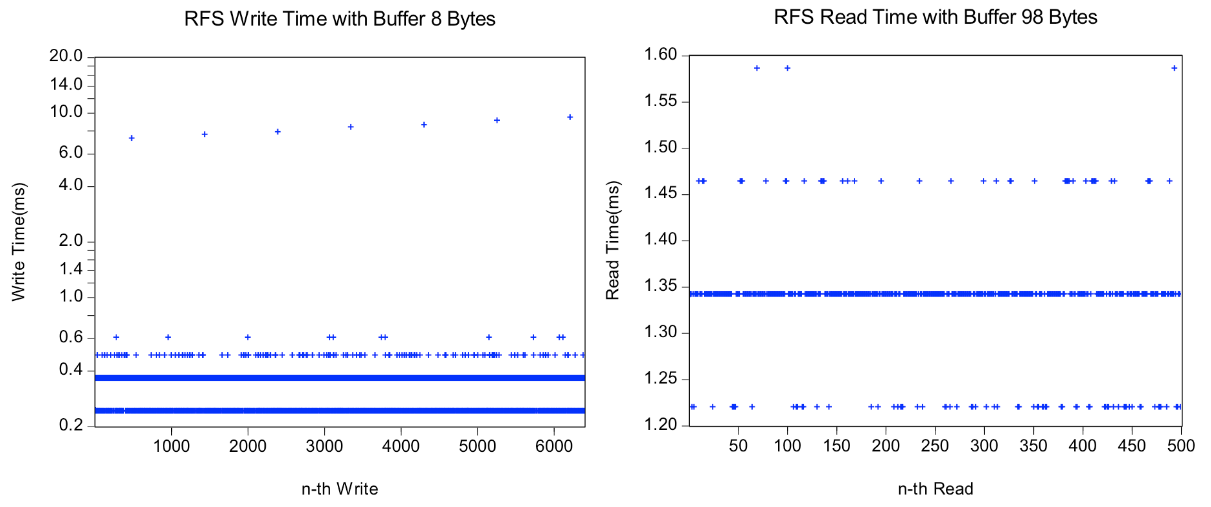


Figure 9: RFS performance evaluation

Figure 9 shows the read and write performance of RFS. As before, 8 byte blocks are appended to a file and subsequently 98 byte blocks are read from the file. It can be seen, that the write performance also has some outliers when new data blocks have to be reserved. Also, outliers occur when the garbage collection has to free obsolete pages. However, none of the outliers for writing data is above 10 ms and therefore well within the timing restrictions of GinMAC. Read performance is comparable to Coffee, although some outliers implicate that reading over data block boundaries sometimes slows down reading.

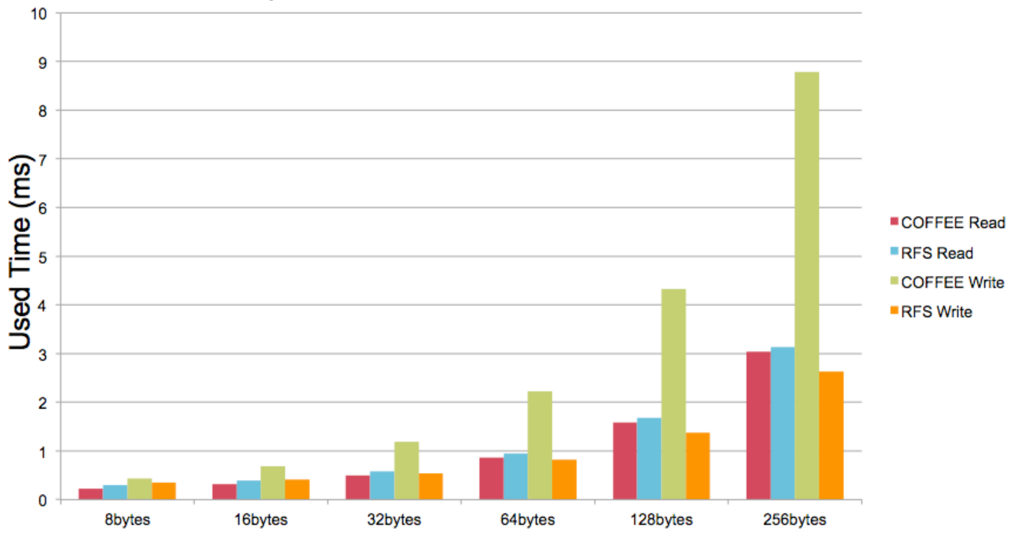


Figure 10: Coffee and RFS sequential read and write performance for different block sizes

Figure 10 shows a comparison of the performance of Coffee and RFS when reading and writing different block sizes. The x-axis shows the write block sizes while the y-axis shows the average of all read/write durations for that specific block size. It can be seen that the read performance of RFS is not as good as Coffee's performance while the difference is app 0.1 ms. However, the write performance of RFS is significantly better for all block sizes. The outliers that are visible in cause this significant difference.

5.5 Second Software Integration (Milestone 4) Evaluation

For the Second Software Integration (Milestone 4) the objective is to reduce the amount of performance monitoring traffic by moving some of the monitoring and analysis functionalities from the sink onto the nodes and into the network, providing a more balanced and efficient solution than was implemented in the First Software Integration (Milestone 2).

The frequency of performance messages sent by a node affects the accuracy of the representation of the network held at the sink. A high update rate provides a clearer picture of the network's current state for the operator but comes with overhead associated with additional network traffic, a low rate means less traffic but the condition of the network available to the operator may be out of date and of little use. The desirable scenario is a low update rate without the loss of accuracy.

It is possible to achieve this goal by getting the node to check for anomalous traffic behaviour based on stored data and send supplementary performance messages if an anomaly is encountered. Given that GinMAC is TDMA based and the application has a periodic nature it is to be expected that the number of messages sent and received by a node in a given period should not vary considerably from one period to the next. Any sudden increase or decrease in traffic in such a system is a key indicator of performance problems or network changes. Interference, disconnected nodes, topology changes all result in a changed message sent/received rate. With this in mind it was decided to periodically log the number of messages sent to and received by a node to Flash and use this log as a basis for the node's usual traffic conditions. Any large deviation from this norm generates an exception and a supplementary performance message to update the sink.

5.5.1 Implementation

While we wish to reduce the overhead associated with sending performance monitoring data to the sink some state information is required at the sink so low-rate periodic updates are still necessary. With this in mind the frequency of performance message transmission is reduced, with the new lower frequency sending only 1 message for every 5 messages normally sent. To ensure the sink is aware of changes in traffic conditions a check for anomalies is performed and supplementary messages are sent if necessary.

Upon initialisation a file is created on the node's Flash using the Coffee filesystem and a ginseng_perfd_fs_epoch event is scheduled. The ginseng_perfd_fs_epoch event updates the file with the number of slots, the number of messages sent and the number of messages received since the last time the event was triggered. A circular array is used to capture a recent snapshot of the node's traffic so bursts can be detected easier. The standard deviation and average number of messages sent and received are then calculated based on the records in the file stored in Flash. Finally a check is performed to see if the current traffic conditions are consistent with those of the stored data. Equation 3 is for messages received:

$$2SD_r < Abs(A_r - C_r)$$

Equation 3: Check for exception in number of messages received

where SD_r, A_r and C_r denote the standard deviation, average and current number of messages received in a period respectively. Similarly exceptions in the rate of messages sent is checked using Equation 4

$$2SD_s < Abs(A_s - C_s)$$

Equation 4: Check for exception in number of messages sent

where SD_s, A_s and C_s denote the standard deviation, average and current number of messages sent in a period respectively.

If an exception is encountered then an exception message is generated and added to the queue. Exception messages are marked so the sink can distinguish easily between exceptions and regular messages sent at the lower frequency.

5.5.2 Evaluation Setup

This evaluation was carried out using a laboratory testbed in University College Cork. The testbed uses the same topology, configuration and software as the testbed used to evaluate the second software integration (Milestone 4) at Sines. Apart from the physical environment the laboratory testbed is as close a representation of Sines testbed as possible.

It consists of 16 TelosB nodes running a version of Contiki 2.4 that has been heavily modified for the GINSENG project including GinMAC and overload control etc. A PC running Ubuntu Linux is connected to one of the nodes and is used as a sink. It also runs the Dispatcher and Monitor applications described in Sections 3.1.3 and 3.2 respectively.

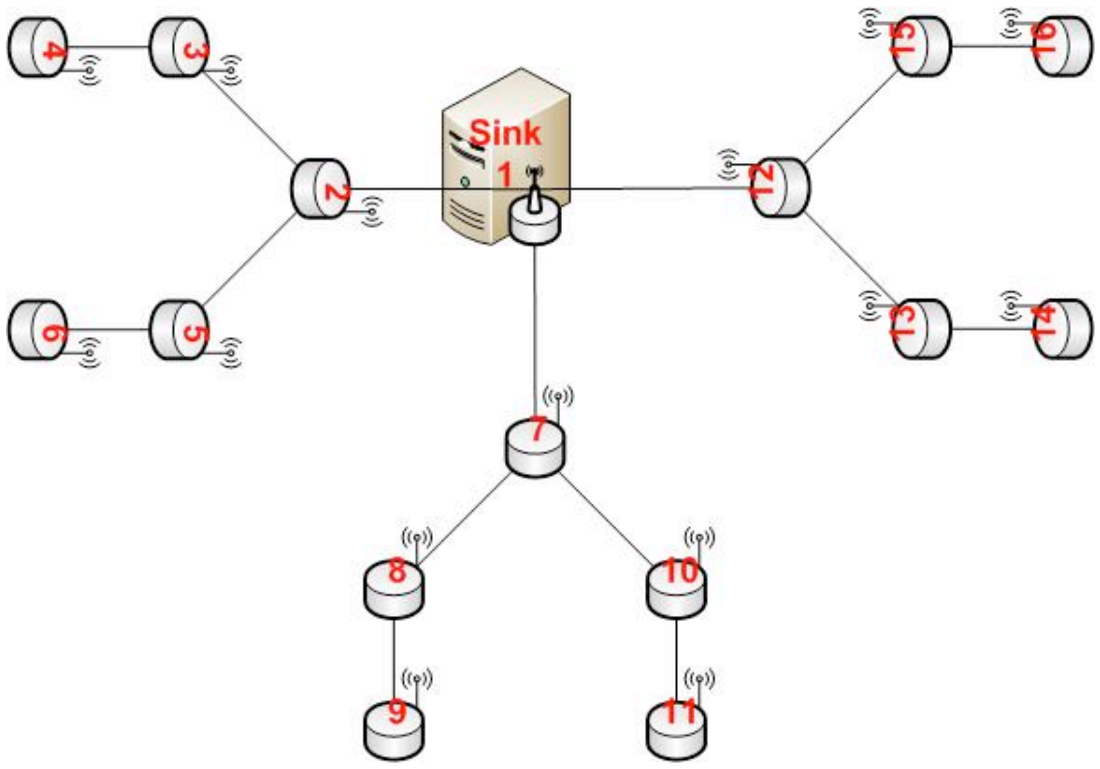


Figure 11: 3-2-1 Network Topology

The other 15 nodes make up a network with a static 3-2-1 topology as shown in Figure 11. Each node generates an application message once per second and a performance messages once every 25 seconds at the default high frequency. One hour is the duration of the experiment. The low frequency for sending performance messages is 1/5th of the high rate, i.e. one message every 125 seconds.

Several GINSENG modules are being evaluated for Milestone 4 using the same set of experiments, this has imposed some constraints on the experiment design as only a limited number of experiments can be run. For this reason all performance messages are sent but are marked as high, low and exception.

5.5.3 Evaluation Results

Of interest in this experiment is the cost of sending performance messages as well as the accuracy of the data at the sink. Data collected was consistent across the 15 nodes in the network; we chose node 8 (Figure 11) to illustrate our findings. Firstly the cost is determined by comparing the number of messages sent by node 8 as shown in Figure 12. We compare the number of messages sent at the high frequency (all performance messages), the low frequency and the low frequency including exceptions. There is a clear and obvious benefit to reducing the frequency in terms of energy consumption and congestion. A significant saving is possible even with the exceptions included.

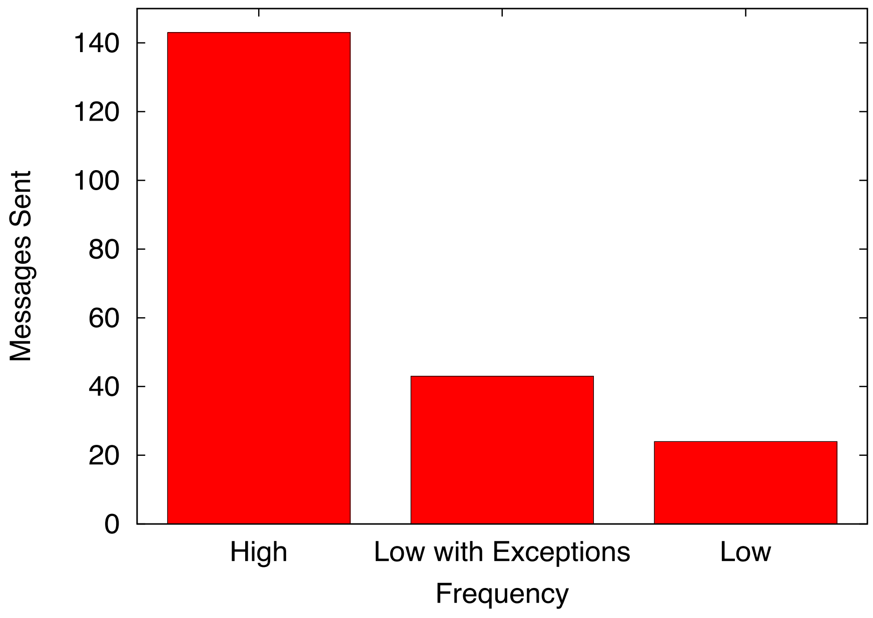


Figure 12: Performance Messages Sent

As stated previously there is a trade off between accuracy and cost. We want to determine how much accuracy is lost by reducing the frequency of sending periodic performance messages. To achieve this measure the accuracy of the data at the sink for messages sent at the low frequency and the low frequency including exceptions against the high frequency. The number of messages sent during each period was normalised against the number of messages sent during each period at the high frequency and plotted in Figure 13.

Several variations from the norm are apparent for the low frequency where bursts in traffic occurred. Variations exist for the Low with Exceptions configuration as well. However these only appear early in the experiment while state on traffic conditions is being collected. Once enough state has been collected to allow calculation of average and standard deviations etc. the data is very accurate and matches the norm exactly.

These results confirm the applicability of using Flash to store data for in-network performance debugging in evaluation in Section 5.3 and show that with the use of Flash memory and the Coffee file system can be used for in network performance debugging and allow significant energy savings to be made without any loss in accuracy of the data held at the sink.

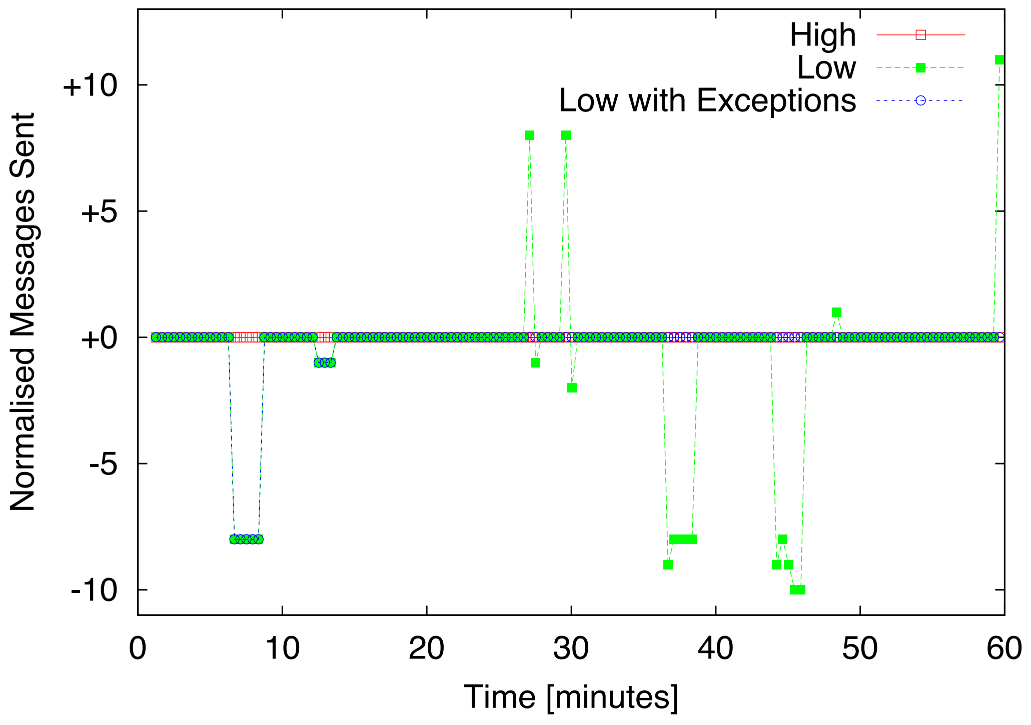


Figure 13: Accuracy Evaluation

6. Support for Experimental Evaluation

In many cases, the experimental evaluation of performance and debugging protocols is non-trivial. The experiments should be repeatable but the results are affected by the environmental conditions that are not constant. An example is the impact of packet loss on protocol performance. In wireless sensor networks, packet loss is often caused by external interference and hence it is beneficial to test the performance of WSN protocols under interference. This, however, requires proper support so that interference patterns can be easily created in a precise and repeatable way.

6.1 Interference Generation

6.1.1 Use of continuous-mode interferer

In Deliverable 2.2 and in previous publications [10][11] we have presented a method to create controllable and repeatable interference. We have shown that many ChipCon radios, such as the popular CC2420 radio chip can be set into different transmit test modes. In the test mode, the CC2420 broadcasts a continuous carrier, which is useful for performance evaluation or lab testing (see page 54 of the CC2420 manual). We have presented experimental results that have demonstrated that using the continuous carrier enables us to overcome the problems of the packet-storm based method without requiring any additional hardware. Our results have further shown that it is possible to have fine-grained control over the interference created by adapting the TX power as well as turning on and off the continuous carrier. This approach makes it also possible to interfere with a specific IEEE

802.15.4 [9] channel, in such a way that the adjacent channels are not affected, i.e. if we interfere channel 23, neither channel 22 nor 24 are affected.

We have successfully used this method of generating interference using the CC2420 test mode when evaluating the BurstProbe mechanism [12], also described in Deliverable 2.3.

We have recently extended our work and developed JamLab, a low-cost infrastructure to augment existing WSN testbeds, to include repeatable interference patterns [12]. JamLab supports the recording and playback of interference traces in WSN testbeds, as well as the customizable generation of typical interference patterns resulting from WiFi, Bluetooth or microwave ovens. To ensure a low-cost and hence widely applicable solution, we also use off-the-shelf motes leveraging the test mode of the CC2420 as described above and in Deliverable 2.2. In this way, a fraction of the already deployed nodes of a testbed can be used for interference generation with the overhead limited to the simple uploading of the appropriate software.

6.1.2 Limitations of continuous-mode interferer

A limitation of the interferer is lack of packet selectivity. In order to block a link partially for purposes such as protocol debugging, a continuous-mode interferer has to duty-cycle its transmissions, or tweak its transmission power level to degrade the targeted link's quality to a certain intermediate level. Both methods require iterative calibration of the interferer, and the resulted partial link tends to fluctuate between good and bad states over time in an uncontrolled manner. One way to achieve better packet selectivity is embedding protocol knowledge into the interferer, and using time synchronization to schedule interference transmission at exact points in time, thus blocking specific packets. The cost is increased software complexity.

A side effect generated by a continuous-mode interferer is its large spectrum footprint onto the targeted channel, which can be captured by a spectrum analyzer or by a dedicated eavesdropping node. Persistent use of a full duty transmitter on an ISM band will degrade all devices' performance, and is incompliant with spectrum administration regulations.

6.1.3 Reactive interferer

To achieve better packet selectivity and bandwidth efficiency than the continuous-mode interferer, we conceived a reactive interferer that detects an IEEE 802.15.4 packet header being transmitted over the channel and then sends out a burst jam signal to destruct the payload. In its default state, the interferer silently listens to the channel, only switching to transmission state temporarily when a target packet appears. Successful detection of a target packet depends on that the received signal strength is at a level above the sensitivity of the interferer (-95 dBm for CC2420); whereas successful destruction of the target packet depends on that the signal-to-noise and interference ratio (SNIR) at the receiver is below the receiver's co-channel rejection (-3 dB for CC2420).

For this type of listen-before-jam operation to react quickly, we need combined fast header detection, RX-to-TX switch and signal transmission. We experimented with two hardware-assisted packet detection modes in CC2420, one based on sampled received signal strength indicator (RSSI) values and the other based on decoding of the start-of-frame delimiter (SFD). We found that the SFD-based mode is superior to the RSSI-based mode. SFD decoding is faster (32 μs) than RSSI sampling (128 μs). We also found that SFD decoding is more reliable than RSSI sampling in terms of probability of neglected packets (false negatives) and fake packets (false positives).

Figure 14 shows the timeline of a packet detected and then destroyed by an SFD-detection based interferer. The sender transmits an IEEE 802.15.4 packet, which begins with a 4-zerobyte preamble followed by a constant SFD 0x7A. Both the receiver and the interferer synchronize with the preamble automatically and then detect the incoming packet when the SFD is successfully decoded. The interferer then switches from RX to TX state and transmits a jam signal for a short period of time. On the other hand, the receiver proceeds to decode the payload length field and then fills the payload into its packet buffer. Because a portion of the payload is contaminated by the jam signal, the receiver CRC check eventually fails, and the whole packet is discarded.

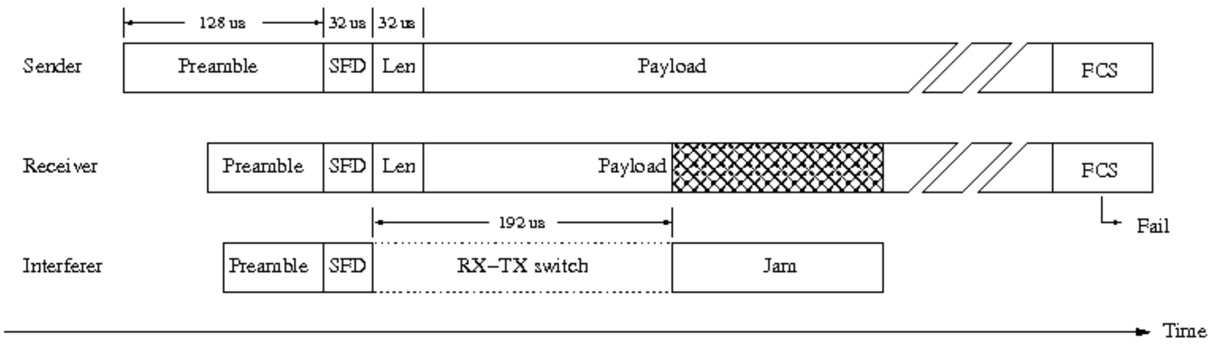


Figure 14: SFD detection-based interferer

6.1.4 Implementation of the reactive interferer

In order to implement the reactive interferer, we modified the Contiki CC2420 radio driver. We enable CC2420's SFD-mode clear channel assessment (CCA) by setting CC2420's MDMCTRL0 register's CCA_MODE field to "2". We enable the MCU's interrupt vector for the CCA pin, which calls a jam generation function. We reuse the continuous-mode interferer's jam generation function, but only enable the transmission mode for a period of about 312 μs, which turns out to be sufficient to destruct a packet. To allow selectivity, we insert a decision function between the SFD detection and jam generation, which may be tailored by the user to achieve a certain random distribution of packet loss. Compared with the software complexity of a normal sensor node, the code and memory usage of the interferer is almost negligible.

6.1.5 Evaluation of the reactive interferer

We summarize the key parameters of the reactive interferer in Table 4.

Minimum PHY payload length of target packet	3 bytes
Sender – interferer delay	352 μs
Minimum effective jam burst length	312 μs
Interferer TX power margin over sender	5 dB

Table 4: Key parameters of the reactive interferer

We want to construct a lossy link of certain packet loss rate and burstiness from a perfect link, by using the reactive interferer to block some of the transmissions. Our test set-up consists of 3 Tmote Sky nodes: a sender, a receiver and an interferer. The nodes are placed apart from each other at certain distances, so that both the receiver and the interferer observe a near-perfect link from the sender, with average signal strength of -82 dBm. The sender uses the Contiki Rime stack's identified broadcast primitive to send out a sequence of 50-byte packets, at 128 Hz rate. The interferer transmits at a power level 5 dB higher that the sender. We implement 3 different jamming decision functions to emulate late different link burstiness: periodic, periodic with a burst of 2 packets, and pseudo random. For illustration purpose, we set packet loss rate to 25% for a test. Figure 15 shows the link statuses over time observed by the receiver. Compared with the continuous-mode interferer, finer-grained control of both packet loss rate and channel burstiness can be obtained with the reactive interferer.

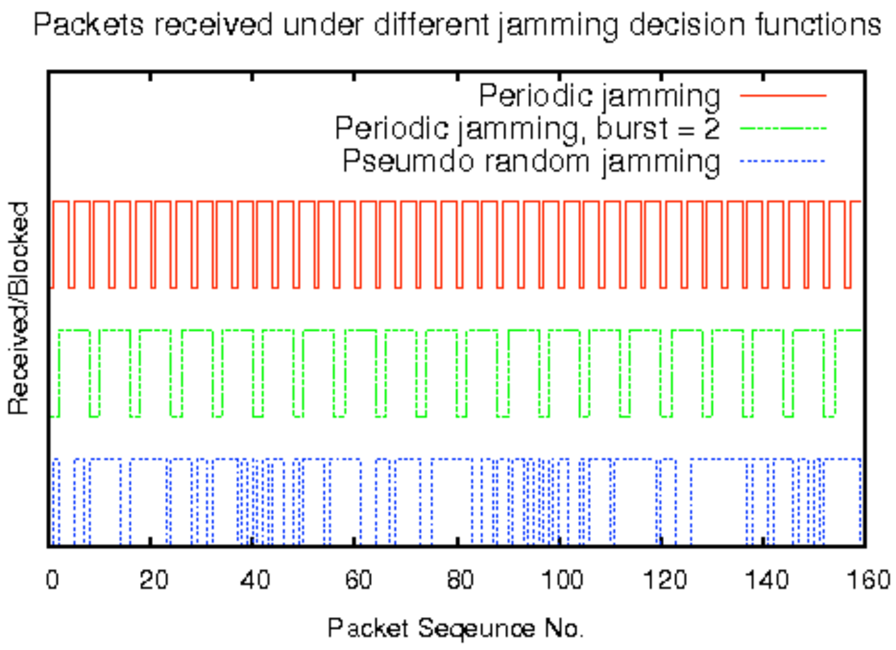


Figure 15: 3 lossy links with 25% packet loss rate, obtained by instructing the reactive interferer to enable jamming based on different decision functions

6.2 Remote Network Reprogramming

As outlined in Deliverable D4.2, the GINSENG project has two central testbeds that are used in addition to smaller testbed at each partner's venue. One is located in Coimbra and operated by FCTUC while the other one is installed in the Sines refinery and operated by Petrogal. The refinery testbed allows testing and evaluating software components in a real industry setting and therefore produces the most realistic results. However, having only the Wireless Sensor Network to obtain data from the nodes is a problem for network evaluation and especially evaluation and verification of Performance Debugging.

Since the capacity of the WSN is tightly limited and since too much in-band information would disturb the normal network operation, the need for an out-of-band signalling solution arose. Also, powering the nodes off batteries may lead to differing conditions when evaluating different components in the refinery. Also, the process of manually flashing nodes is error prone and time consuming. To solve these issues, we are currently in the process of implementing means to

- Flash software onto nodes from a central point
- Provide constant power supply to the nodes
- Access the serial log output of the nodes

The envisioned solution is to connect embedded PCs running the Linux operating system to each node. These boards are connected to a central server using standard Ethernet cabling and also receive power over the Ethernet cable. With custom software, this enables us to receive much higher volumes of information from each node. Also, this method will provide us with time synchronised log messages from all nodes, thereby allowing us to estimate the latency of the network. More details on this solution will be outlined in the upcoming deliverable D4.5.

7. Future Work

The evaluation of in-network debugging and comparison with alternative approaches will be repeated in the deployment at the Sines refinery with a view to validating our laboratory results in a real world setting and allowing a more thorough investigation of its efficiency and effectiveness.

There are also plans to add more data, such as RSSI, to that being stored on a node's Flash for in network debugging. This will allow more comprehensive analysis and diagnosis on the nodes.

The in-network debugging implementation for the GINSENG system would be improved with the multi-ring file system presented. This will be added to the system and evaluated.

8. Conclusion

Performance debugging is a crucial component for WSN applications in critical systems environments. Any failure in the network can have potentially catastrophic consequences for the operation of the system involved as well as those in the vicinity. This document examines the work done in order to allow performance debugging in WSN applications and provide suite of solutions. Results from laboratory testbeds were presented, together with a description of the role of performance debugging in the project's Second Software Integration and preliminary evaluation (Milestone 4). The performance debugging is currently deployed on-site in the Sines refinery and the upcoming WP4 deliverable D4.5 will include related preliminary evaluation and analysis.

References

[1] J. Romkey. A Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP, IETF RFC 1055, Jun. 1988.

[2] C. A. Boano, N. Tsiftes, T. Voigt, J. Brown, and U. Roedig. The Impact of Temperature on Outdoor Industrial Sensornet Applications, In IEEE Transactions on Industrial Informatics, Vol. 6, No. 3, Aug. 2010.

[3] N. Finne, J. Eriksson, A. Dunkels, and T. Voigt. Experiences from two sensor network deployments self-monitoring and self-configuration keys to success, In Proceedings of 6th International Conference on Wired/Wireless Internet Communication (WWIC), May 2008.

[4] N. Tsiftes, A. Dunkels, Z. He, and T. Voigt. Enabling Large-Scale Storage in Sensor Networks with the Coffee File System, Proceedings of the ACM International Conference on Information Processing in Sensor Networks (IPSN), Apr. 2009.

[5] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, and T. Voigt. Accurate Network-Scale Power Profiling for Sensor Network Simulators. In Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Feb. 2009.

[6] A. Dunkels, F. Österlind, and Z. He. An adaptive communication architecture for wireless sensor networks. In Proceedings of 5th ACM International Conference on Embedded Networked Sensor Systems (SenSys), Nov. 2007.

[7] H. Dai, M. Neufeld, R. Han. ELF: An Efficient Log-Structured Flash File System For Micro Sensor Nodes, Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SenSys) Nov. 2004.

[8] W. Tang. Ring Storage System in Wireless Sensor Network, Bachelor Thesis, Technische Universität Braunschweig, 2010.

[9] IEEE 802.15.4, http://www.ieee802.org/15/pub/TG4.html

[10] C. Boano, Z. He, Y. Li, T. Voigt, M. Zuniga and A. Willig. Controllable radio interference for experimental and testing purposes in wireless sensor networks. Fourth IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp), Oct. 2009.

[11] C. Boano, K. Römer, Z. He, T. Voigt, M. Zuniga and A. Willig. Demo Abstract: Generation of Controllable Radio Interference for Protocol Testing in Wireless Sensor Networks. Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems (SenSys), Nov. 2009.

[12] J. Brown, B. McCarthy, U. Roedig, T. Voigt and C. Sreenan. BurstProbe: Debugging Time-Critical Data Delivery in Wireless Sensor Networks. In Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Feb. 2011.

[13] C. Boano, K. Römer, T. Voigt, C. Noda, M.Zuniga. JamLab: Augmenting Sensornet Testbeds with Realistic and Controlled Interference Generation. Proceedings of the ACM International Conference on Information Processing in Sensor Networks (IPSN) - SPOTS Track, Apr. 2011. To appear.

[14] V. Pejovic, C.J. Sreenan. PerDB: Performance Debugging for Wireless Sensor Networks. Poster Proc. of 6th European Conference on Wireless Sensor Networks (EWSN), Feb. 2009.

[15] T. O'Donovan, N. Tsiftes, Z. He, T. Voigt, C.J. Sreenan. Detailed Diagnosis of Performance Anomalies in Sensornets. Proc. of Sixth ACM Workshop on Hot Topics in Embedded Networked Sensors (HotEmNets), Jun. 2010.

[16] T. O'Donovan, C.J. Sreenan, N. Tsiftes, Z. He, T. Voigt. Poster Abstract: Storage-Centric Debugging of Performance Problems in Sensor Networks. Poster Proc. of 7th European Conference on Wireless Sensor Networks (EWSN), Feb. 2010.

[17] I. Stokes. Performance Debugging in Wireless Sensor Networks. M.Sc. Thesis, University College Cork, Sept. 2010.