




<b>Project Number:</b>	<b><i>INFSO-ICT-224282</i></b>
<b>Project Title:</b>	GINSENG-Performance Control in Wireless Sensor Networks
<b>Project Deliverable Number:</b>	224282/SICS/D2.5
<b>Contractual Date of Delivery to CEC:</b>	Month 42 – February 2012
<b>Actual Date of Delivery to CEC:</b>	
<b>Title of Deliverable:</b>	GINSENG D2.5: System Evaluation
<b>WP contributing to the Deliverable:</b>	WP2

**Abstract:**

This deliverable updates D2.3 and D2.4 including lessons learned from the second system integration and test (WP4). The algorithms developed in Task 2.1, 2.2, 2.3 and 2.4 are tested and evaluated and final improvements are implemented.

**Keywords:**

Operating system, node lifetime prediction, medium access control, performance debugging

INFSO-ICT- 224282	Deliverable D2.5 System Evaluation	
----------------------	---------------------------------------	---

## Executive Summary

The deliverable updates D2.3 and D2.4 with final refinements of WP2 components and respective evaluation results.

Various contributions are made in the four WP2 components, namely operating system, medium access control protocol, sensor node life time prediction, and performance debugging. Our operating system's realtime scheduling mechanism is extended with support for non-preemptable tasks. Our medium access control protocol is evaluated in the Sines refinery outdoor testbed and shows robust performance. Our sensor node life time prediction shows high accuracy over long test periods. Finally, our performance debugging mechanism successfully captures important communication faults due to external factors during the test runs in Sines.

In conclusion, we have developed a set of essential software components that together provide a solid foundation for wireless sensor network applications operating in harsh industrial environments.

## List of Contributors

Name	Company	Address	Phone/Fax/E-mail
Tony O'Donovan (deceased)	UCC		
James Brown	ULANC	Lancaster, UK	j.brown@lancaster.ac.uk
Wolf-Bastian Pöttner	TUBS	Braunschweig, Germany	poettner@ibr.cs.tu-bs.de
Zhitao He	SICS	Stockholm, Sweden	zhitao@sics.se

## Document Approval

	Name	Address	Date
Approved by WP Leader	Thiemo Voigt	thiemo@sics.se	30/03/2012
Approved by Project Coordination Committee Member 1	Vasos Vassiliou	vasosv@cs.ucy.ac.cy	04/04/2012
Approved by Project Coordination Committee Member 2	Paulo Gil	pgil@dei.uc.pt	04/04/2012

## Table of Contents

<b>1. INTRODUCTION</b> .....	<b>2</b>
<b>2. OVERVIEW ON WP2 COMPONENTS</b> .....	<b>3</b>
2.1 NODE OPERATING SYSTEM.....	3
2.1.1 <i>Scheduling non-preemptable tasks</i> .....	3
2.1.2 <i>Task execution time profiling</i> .....	4
2.2 MAC PROTOCOL.....	5
2.2.1 <i>GinMAC</i> .....	5
2.2.2 <i>GinLite</i> .....	6
2.3 SENSOR NODE LIFETIME PREDICTION.....	6
2.3.1 <i>Online Energy Estimation</i> .....	7
2.3.2 <i>Offline Power Prediction</i> .....	8
2.3.3 <i>Calibrating the offline power prediction</i> .....	8
2.4 PERFORMANCE DEBUGGING .....	9
2.4.1 <i>Sink-based detection and diagnosis tool</i> .....	9
2.4.2 <i>Management Information Base (MIB)</i> .....	9
2.4.3 <i>In-network debugging</i> .....	10
2.4.4 <i>Network monitoring and management</i> .....	10
<b>3. SYSTEM EVALUATION OF WP2 COMPONENTS</b> .....	<b>11</b>
3.1 NODE OPERATING SYSTEM.....	11
3.2 MAC PROTOCOL.....	12
3.2.1 <i>GinMAC Performance in the GINSENG Testbed</i> .....	12
3.2.2 <i>Conclusion</i> .....	16
3.3 SENSOR NODE LIFETIME PREDICTION.....	17
3.3.1 <i>Evaluation</i> .....	17
3.3.2 <i>Limitations</i> .....	18
3.4 PERFORMANCE DEBUGGING .....	19
3.4.1 <i>Sines Testbed</i> .....	19
3.4.2 <i>Testbed Configuration and Evaluation</i> .....	20
<b>4. CONCLUSION</b> .....	<b>23</b>
<b>REFERENCES</b> .....	<b>24</b>

## List of Figures

Figure 1: Average Task Submission-Invocation Delay Using Different Variants of Waiting .....	11
Figure 2: Average CPU Usage Percentages Using Different Variants of Waiting .....	12
Figure 3: Final Testbed Layout - Total End-to-End Loss .....	13
Figure 4: Final Testbed Layout - Total Time-Sync Failures .....	14
Figure 5: Final Testbed Layout - Total Energy Consumption.....	15
Figure 6: Relative Error of the Offline Energy Prediction. ....	17
Figure 7: Excerpt of the Relative Error of the Offline Energy Prediction. ....	18
Figure 8: Diagram of Network 1 Layout.....	19
Figure 9: Network 1 Initial Configuration Testing – Link Loss Rate .....	20
Figure 10: Network 1 Initial Configuration Testing – End-to-end Loss Rate .....	21
Figure 11: Network 1 Initial Configuration Testing – Time-sync Failures.....	22

## List of Tables

Table 1: Power consumption calibration values from testbed.....	9
Table 2: Test parameters .....	11

## List of Acronyms

ACK	Acknowledgement
API	Application Programming Interface
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
GUI	Graphical User Interface
MAC	Media Access Control
MCU	Microcontroller Unit
MIB	Management Information Base
PC	Personal Computer
RAM	Random-Access Memory
RX	Receive
SLIP	Serial Line Internet Protocol
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TX	Transmit
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
WSN	Wireless Sensor Network
XML	Extensible Mark-up Language

## 1. Introduction

Achieving predictable performance in network communication of wireless industry sensor and actuators is a challenging task, because of the need to allocate limited channel capacity among the devices as well as the need to provide extra redundancy to tolerate losses due to external noise factors. While a carefully designed medium access control protocol can guarantee rigorous data delivery deadlines are met and recover occasional data losses, coordinating radio communication with other operations across the networking stack as well as other sensing and control tasks requires dependable realtime task scheduling support in the node operating system. To scale up deterministic node-level operations across the whole industry sensor and control network, two essential system software services are further needed: accurate node life time prediction and comprehensive performance debugging of key metrics.

We have researched on these requirements and developed a set of software components to fulfil such requirements, as reported in previous deliverables D2.3 and D2.4. Based on the lessons learned from the second system integration and evaluation, we have made further improvements to the components we developed and conducted more complete evaluations.

The rest of this document is organized as follows. We provide in Chapter 2 an overview of each of the four WP2 components, namely node operating system, medium access control protocol, sensor node life time prediction, and performance debugging. System evaluation of each component is described in Chapter 3. Conclusive remarks about WP2 are made in Chapter 4.



## 2. Overview on WP2 Components

### 2.1 Node Operating System

Extending our previous work on providing reliable realtime task scheduling for sensor nodes, we turn our focus to providing a more complete scheduling solution that further includes scheduling of non-preemptable tasks and automatic profiling of the execution time of such tasks.

#### 2.1.1 Scheduling non-preemptable tasks

Over the development of GINSENG system, we have identified three categories of realtime requirements for concurrent tasks running on nodes:

- Epoch-synchronized tasks
- Preemptable tasks
- Non-preemptable tasks

Epoch-synchronized tasks consist of two main types of operations: time-slotted radio channel accesses, which are prioritized in order to ensure message delivery latency is strictly bounded within the application's specified requirement; periodic, supportive routines for data communications, such as topology control, queue management, network statistics, etc, which are tightly coupled with radio communication. These are convenient to be scheduled into pre-allocated processing time slots for each epoch. We have already shown in D2.1 that our development of Contiki's rtimer, a hardware timer interrupt-driven task scheduler, fulfils the requirement for our MAC protocol (GinMAC)'s time-slotted scheduling. The robust performance of GinMAC in the final integration test has proven the excellent accuracy and reliability of rtimer.

As a direct consequence of the pervasive use of the GinMAC time slots for task scheduling, a node's CPU state timeline is divided into a series of fixed-length time slots bounded by periodical interrupts generated by the underlying rtimers. Tasks implemented as ordinary event-driven Contiki processes, such as debug prints, can be preempted by rtimer interrupts in unpredictable moments but are usually able to tolerate the resultant context switching delays. On the other hand, I/O accesses to certain industrial peripheral sensors or actuators might fail if a command sequence is interrupted in the middle of the execution. Integrating these usually infrequent I/O accesses into the normal GinMAC epochs would comprise the flexibility of our system, since the timing requirements of each peripheral is hardware-dependent. Alternatively, putting such kind of I/O accesses into critical sections with disabled interrupts would tamper the rtimer mechanism, which would in turn cause the realtime tasks to miss deadlines. We instead decide to go for a less aggressive strategy by putting off the entrance time of any non-preemptable code block until after an imminent rtimer task is finished, so that safe I/O accesses can be scheduled in the nearest future gap between two rtimer interrupts. The safety comes at a cost of higher average response time. To allow programming of such tasks as a normal Contiki process, we develop a new Contiki process macro *WAIT\_FOR\_NON\_PREEMPTION (duration)*, which is to be placed ahead of a non-preemptable block to reserve a length of continuous CPU time. The macro hides communication with the underlying rtimer that determines whether the current process must give way to an imminent rtimer task and at what future time to resume execution.

The implementation can have a few variants, which all share the following common loop structure:

```
while((t = rtimer_pending()) != NULL &&  
      RTIMER_TIME(t) - RTIMER_NOW() < duration) {  
    //yield and wait  
}
```

The "yield and wait" statements should schedule a future continuation inside the loop before yielding CPU control. There are two main implementation considerations here, which are conflicting: the loop should to be exited as soon as possible to minimize the delay of the suspended

process, but the checking of the exit condition should be as sparse as possible to save CPU usage. We have implemented four different “yield and wait” mechanisms, which represent different trade-offs between time delay and CPU overhead.

The first variant consists of an empty “yield and wait” statement, so it never yields but just busy waits for the loop condition to become invalid.

The second consists of just a one-line statement:

```
PROCESS_PAUSE();
```

It is a Contiki process macro that is equivalent to *PROCESS\_YIELD()* followed by *process\_poll(current\_process)*. The resulting loop represents a best-effort, self-polling policy, which normally results in multiple iterations of yielding and checking until the loop exits. It is equivalent to checking the status of a spinning lock controlled by the rtimer.

The third variant uses a Contiki etimer to reschedule the current process to run at a later time:

```
etimer_set(&_wait_et, rtimer_time_until_next()/(RTIMER_SECOND / CLOCK_SECOND) + 1);  
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&_wait_et_));
```

If the etimer’s timeout value is set to a conservative estimate, then the obstructive rtimer task must have already completed when the etimer expires and the current process resumes. No multiple loop iterations are needed, so CPU overhead is lower than the previous method.

The fourth variant requires extensions made to the rtimer C struct definition as well as the *rtimer\_run\_next()* function, so that a callback process can be appended to an rtimer. As the rtimer task has finished, the callback is executed to pass control back to the suspended process:

```
rtimer_pending()->callback_process = PROCESS_CURRENT();  
PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_POLL);
```

Using a callback process call issued by the obstructive rtimer task to resume the suspended process guarantees minimal latency.

We have added a fifth variant since D2.3, which is a broadcast version of the fourth, whereby an rtimer task always notifies all suspended Contiki processes about its completion by sending a special broadcast event, so that any process in need of a non-preempted CPU duration can wait for such an event. This removes the need for any callback registration, but increases the CPU overhead due to the use of event broadcast.

## 2.1.2 Task execution time profiling

Taking the execution time of a code block into consideration is beneficial for more efficient task scheduling design. In Contiki, the normal practice for execution time measurement is enclosing a code block with two timestamps and then computing their difference. The resolution of the timestamps is however rather coarse-grained due to the need for economical use of hardware timer interrupts. For GINSENG, the timestamps’ tick interval is 1/8192 second. In order to amortize relatively large measurement errors due to limited time resolution, repeated executions of the same code block are usually needed to achieve a good estimate.

When we introduce the *WAIT\_FOR\_NON\_PREEMPTION (duration)* macro for scheduling non-preemptable code in the previous subsection, we assume the user provides a reasonably accurate guess of the “duration” parameter, based on his or her knowledge about the execution time of the critical code block. Nevertheless, if the value is too optimistic, i.e., smaller than the actual execution time, the risk that the process becomes interrupted later by the rtimer becomes rather high. On the other hand, a too pessimistic estimate causes unnecessary delay. A self-learning code execution time profiler would come in handy for setting the duration parameter to an appropriate value.

We therefore have implemented such a profiler to measure the execution time of a code block in terms of rtimer clock ticks. The users first initialize the profiler by calling a function called

*cprofile\_init(duration, margin, runs)* with three parameters: an initial, conservative guess on execution time, a safe margin, and number of runs required to replace the initial guess value with measured time. The code block to be measured is enclosed by a pair of *cprofile\_start()* and *cprofile\_end()* calls. The profiler automatically updates the measured time after each execution of the code block, and eventually settles at a level corresponding to a worst-case estimate, replacing the initial guess. When scheduling a non-preemptable code block, the profiler can be used to extract an accurate duration value to supply to *WAIT\_FOR\_NON\_PREEMPTION (duration)*, to ensure sufficient CPU time is reserved for the code block.


## 2.2 MAC Protocol

This section presents a final conclusive description and analysis of the Medium Access Control protocol (GinMAC) that is designed to detail its capabilities and characteristics when used in a real world deployment. Previously, in D2.3 a comprehensive study of the performance of GinMAC in a controlled laboratory environment was presented. This provided an insight into the possible potential of the GinMAC protocol; however, it is common knowledge that real world deployment of cutting edge research technologies can create unexpected results because of the unpredictable nature of the surrounding uncontrolled environment. Therefore, in order to suitably demonstrate GinMAC's capabilities we present here a follow-on study from that in D2.3 using data acquired about its performance that has been solely gathered from an extended sample taken at the Petrogal testbed in Sines. In addition, we also provide an overview of the work carried out to develop a more lightweight, generic GinMAC design and implementation that is intended to be more portable. By stripping out functionality from GinMAC that was intrinsically linked to the GINSENG system and by streamlining the overall protocol design, we have been able to develop a final release that can be more easily incorporated into other systems. We call this development GinLite.

### 2.2.1 GinMAC

The GinMAC protocol is the TDMA based Medium Access Control protocol designed for use in the GINSENG system. It is a key component in ensuring the reliable delivery of data across the wireless network infrastructure that the GINSENG model implements and provides attachment points for other GINSENG system components such as topology control, traffic management or performance debugging. By exercising a fine grained control over the timing of all wireless transmissions carried out in a GINSENG deployment, GinMAC attempts to eliminate data loss that can occur from collisions when multiple nodes transmit data simultaneously. This functionality ultimately ensures that a GINSENG network can offer deterministic data transmission properties that are integral to supporting the stringent data delivery requirements of many industrial processes, including those found in oil refineries. In particular, the GinMAC protocol consists of the following three main features:

1. **Offline Dimensioning:** Through prior analysis, traffic patterns and channel characteristics are established for proposed GINSENG networks before deployment. Thus, complex protocol operations such as the calculation of the transmission schedules are performed off-line and before the physical network deployment occurs.
2. **Exclusive TDMA:** A TDMA schedule with exclusive slot usage is utilised to overlay network-wide control over node transmissions in order to achieve deterministic data delivery.
3. **Delay Conform Reliability Control:** The protocol is designed to support delay bounds of  $D_S$  and  $D_A$  while achieving very high data transport reliability. Thus, all available flexibility

INFSO-ICT- 224282	Deliverable D2.5 System Evaluation	
----------------------	---------------------------------------	---

in transport delays is used to improve reliability (assuming energy consumption targets permit this).

Reliability throughout the network is ensured by the provisioning of redundant slots within the schedule for retransmission, the number of redundant slots required is determined during the pre-deployment offline dimensioning measurement phase. This core functionality coupled with additional performance improving capabilities combine to produce a MAC protocol that is ideally equipped to support reliable and deterministic data transfer over wireless networks in challenging industrial environments.

### 2.2.2 GinLite

GinMAC is a protocol that has been specifically tailored to meet the requirements of specific industrial process automation scenarios. However, in addition to industrial process control, many other domains also require reliable time-critical data delivery. With the lack of available open source off-the-shelf systems to support these requirements, commencing research in these areas is difficult. Whilst the GINSENG system could be used to support such research, it may be considered excessive and too complex as it potentially provides many unnecessary features, which have been tailored to particular industry settings. Instead it was believed that a more generic system was required to help facilitate further research in the area of reliable and deterministic wireless sensor networks. We call this optimised system GinLite, a derivative of the original GinMAC protocol that is designed to only provide the essential minimum features and to support modification to adapt the protocol to a variety of real-time research tasks.

GinLite is implemented for the Contiki sensor network operating system. GinLite supports a number of advantages and improvements over GinMAC such as reduced resource use, a simpler implementation, and better integration into Contiki OS. These features make the system more suitable as a reliable research platform. The system requires less than 5KB of flash memory, which is comparable to other Contiki MAC protocols that do not have to deal with the complexity of supporting real-time communication. With regards to RAM the system requires 1.2KB of RAM, which includes support for a frame queue of 8 frames. This queue partially replaces the default queue of 16 packets held in the Contiki RIME system. With a modest Rime based application, only 23KB of flash and 3KB of RAM are used leaving 25KB of flash and 7KB of RAM available on a commonly used Tmote Sky mote for system extension. The implementation of GinLite has a simple structure with the entire MAC protocol contained in a single file in a similar structure to other Contiki MAC protocols. Additional components such as Topology Control, Overload Control or Performance Debugging can be attached. However, the basic system only includes minimal implementations of these add-ons. For example, the system utilizes a purely static topology with pre-computed and static TDMA schedule. Nevertheless, if a research project requires, a complex topology management component can be added. The system provides mechanisms to transmit additional information piggybacked on data transmissions which allows researchers to construct complex performance monitoring components without disturbing real-time communication. The basic system supports a simple FIFO queue but this can be replaced by sophisticated queue management if the targeted research requires this. GinLite has been designed to be tightly integrated with Contiki OS, implementing the standard Contiki MAC interfaces. This allows the system to be used in the same way as other built-in MAC protocols and allows the system to be used with ease, with the provided Contiki OS example RIME based applications and with some small modifications with 6lowpan applications.

## 2.3 Sensor Node Lifetime Prediction

In order to predict the lifetime of a sensor node, the remaining capacity of the battery as well as the average power consumption of the node are important metrics. However, due to the proper-

ties of batteries, accurately measuring the remaining capacity is not possible with off-the-shelf sensor nodes [2].

Fortunately, GINSENG is using a TDMA MAC Layer with exclusive slot usage that eliminates idle listening by design almost completely. GinMAC usually uses most available slots by generating dummy data whenever no user data is ready to be transmitted. Consequently, the power consumption of a node running GinMAC is predictable which allows for offline life predictions of nodes without the necessity of live data assuming that the nodes behave as designed.

As outlined in Deliverable D2.4, Contiki's Online Energy Estimation (Energest) allows estimating the power consumption of a node. By measuring the time that is spent in different power states, GinEST estimates the actual power consumption of each node. Information about how much time is spent in which power state is transported to the sink via Performance Debugging and from there on to the dispatcher. The dispatcher uses numbers about current consumption of different components to calculate the actual power consumption per power state and sums this up, producing a sum of the power consumption of a node.

However, for various reasons, nodes behave different than designed and the power consumption calculated by the receiver does not match with the predictions. In effect, this allows comparing the predicted and the estimated power consumption to detect nodes that are not behaving as designed. In fact, on several occasions nodes in the Sines testbed have shown significantly higher power consumption than other nodes in the same tree tier. This implies that said nodes are misbehaving as far as the application is concerned.

Reasons for such misbehaviour may include but are not limited to the following:

- Problems with the radio hardware may make operations take longer or shorter
- Problems with the CPU may make the node sleep more or less often
- Problems with the CPU may make certain computational operations take longer or shorter
- Problems with the radio hardware may produce more TX or RX failures or retransmissions

The mechanisms presented in the following sub-sections can be used to detect power consumption of nodes that is significantly different than the predicted consumption. However, at the current stage these mechanisms do not allow to locate the cause of this mismatch. As we will explain later, due to the limited accuracy of this approach only significant mismatches can be detected.

### 2.3.1 Online Energy Estimation

As outlined in Deliverable D2.4 we already have means to estimate the power consumption of a node based on Contiki's Online Energy Estimation module. The following states of power consumption are recorded:

- Radio RX
- Radio TX + transmission power level
- CPU Active
- CPU Low Power Mode
- Onboard Flash Read
- Onboard Flash Write

Since transmitting all those values in one performance-debugging packet is not possible, we are now using an aggregation approach for the different TX power levels. By aggregating the power



consumption of three transmission power levels we increase communication efficiency by sending fewer packets. On the other hand, we are limiting the resolution and accuracy of the power estimation, while this should not be a problem in practice.

### 2.3.2 Offline Power Prediction

GinMAC divides time in slots whereas a number of slots are aggregated into an epoch. Epochs are repeated continuously whereas nodes will likely behave more or less the same in all epochs. The allocation of slots to nodes is static which also makes the communication activity of each node static. Since communication is widely accepted to be the greatest contributor to power consumption, we can also expect the power consumption per epoch to be almost static.

In addition to the communication activity that is mainly related to the power consumption of the radio, the CPU of a node also influences the total power consumption. But also for CPU activity it is a reasonable assumption that in most epochs the power consumption will be almost the same.

This can be broken down even further. From a radio perspective, the slots within an epoch are either inactive, receive or transmit. With a certain inaccuracy we can assume, that the power consumption in each slot of a specific type is equal. This observation allows predicting the power consumption of each node in the network by extrapolation based on the schedule. For this task a formal specification of the schedule in the network is required. Furthermore, information about the power consumption in each slot type is required. With this knowledge, we can calculate the power consumption  $P_{E_i}$  of node  $i$  in a single epoch using Equation 1. For this equation we have to know, how many TX or RX slots ( $S_{tx_i}, S_{rx_i}$ ) the node has per epoch. In addition, we need to know the number of active and sleep slots ( $S_{Active_i}, S_{Sleep_i}$ ) the node has. Of course, the sum of these four slot counts can exceed the total number of slots in an epoch since the CPU of a node is likely in active state for RX and TX slots. In addition, the CPU may be active in even more slots due to local computation or sensor sampling. Finally, we have to measure the power consumption for each specific slot type ( $P_{tx}, P_{rx}, P_{Active}, P_{Sleep}$ ).

$$P_{E_i} = S_{tx_i} \times P_{tx} + S_{rx_i} \times P_{rx} + S_{Active_i} \times P_{Active} + S_{Sleep_i} \times P_{Sleep}$$

**Equation 1: Calculate the power consumption for a single epoch for node i.**

Since we assume that the power consumption is similar each epoch, summing up the total power consumption of a node can be done by multiplying the power consumption per epoch with the number of active epochs  $E_i$  with the current schedule according to Equation 2.

$$P_i = P_{E_i} \times E_i$$

**Equation 2: Calculate the total power consumption of node i.**

### 2.3.3 Calibrating the offline power prediction

To obtain realistic values for the Power consumption in different slot types, we have used results from a lab testbed in Braunschweig. We have used a network of 16 nodes in a 3-2-1 topology with a fixed schedule. We have recorded the power consumption using Contiki's Energyest mechanism and summed up the consumption for the four individual types of power consumption that we have identified earlier. After 3.16 hours of experiment duration, we have di-

vided the total energy consumption per type with the number of epochs and slots of a specific node according to Equation 3.

$$P_{tx} = \frac{P_{tx_{total}}}{S_{tx_i} \times E_i}$$

**Equation 3: Obtaining the TX power consumption per slot.**

We have done this for 7 nodes in the testbed. As expected, we have found slight deviations in the per-slot power consumption between the different nodes. This can be caused by variations in the behaviour in the different slots (Unicast transmission receiving an ACK or not) and also by the limited accuracy of the Energest mechanism. While this limits the accuracy of the presented approach, it is not a principal problem and can be overcome by using an average over a significant number of nodes and slots. Table 1 shows the values recorded in the testbed where the standard deviation for RX and TX slots is 2.00 %.

	Arith. Mean [uJ]	Minimum [uJ]	Maximum [uJ]	Standard Deviation [uJ]
<b>Slot RX</b>	225.56	219.22	235.10	6.76
<b>Slot TX</b>	64.39	61.97	66.54	1.93
<b>Slot Active</b>	4.43	4.28	4.61	0.11
<b>Slot Sleep</b>	0.34	0.33	0.34	0.00

**Table 1: Power consumption calibration values from testbed.**

## 2.4 Performance Debugging

Performance debugging is a crucial GINSENG component that collects and analyses performance data from sensor nodes and performs fault detections. Key contributions include a sink-based detection and diagnosis tool, a management information base, an in-network debugging tool, and network management software including a dispatcher program and a monitor program.

### 2.4.1 Sink-based detection and diagnosis tool

Our centralized version of performance debugging PerDB relies on the sink collecting periodic updates from individual nodes and analyzing anomalies from the collected performance data. By checking packet losses and delivery latency against specified performance thresholds, the PerDB backend raises exception alerts to the network operator. In addition to flagging faulty nodes with under-performing metrics, PerDB also attempts to provide hints at the possible cause of a problem, such as disconnection, congestion, node reboot, etc, by matching the metrics with a set of heuristic criteria. The operator may perform further investigation on specific nodes by requesting more details to be reported from them, by sending a probing command to pull specific parameters from on-node MIBs.

### 2.4.2 Management Information Base (MIB)

A Management Information Base (MIB) resides on every GINSENG node's RAM, which maintains a comprehensive repository of performance data. Four types of data are maintained: node status, packet statistics, neighbour status, and scheduling and timing. Information from the MIB is periodically sent to the sink to augment information embedded in data packets. Moreover, the

MIB also provides valuable information to GinMAC, topology control and overload control to assist decision making.

### 2.4.3 In-network debugging

In order to reduce the number of periodical performance updates by PerDB, so as to save bandwidth and energy, we move two of its most important anomaly detection logics from the sink to the nodes. Leveraging the Coffee file system running on Tmote Sky's Flash memory, we log onto each node the counts of messages sent and received in past time windows to a file, and keep track of their average value and standard deviation. An anomaly is detected when the latest time window observes a large deviation in either the number of sent or received messages from respective historical means, and an exception message is generated and reported to the sink. We have shown in D2.4 that an 80% reduction of periodical updates can be achieved without loss of detection accuracy.

### 2.4.4 Network monitoring and management

The Dispatcher software has been developed to maintain a reliable and synchronized interconnection between the sensor network and backend components. Performance messages, along with data messages and other control messages collected from the sensor network by the sink node are sent in SLIP encoding over the sink node's USB connection to the dispatcher running on a sink PC. The dispatcher performs CRC checking and detection of duplicates on incoming messages, before transforming them into time-stamped XML data records, which in turn are accumulated and stored as log files for offline analysis.

Two backend components, the middleware and the network monitor, query current status of the sensor network by connecting to the dispatcher via TCP. The network monitor includes a GUI to allow the network operator to inspect node-specific parameters as well as network-wide parameters such as topology, so that potential problems can be identified. It also offers the option to send raw data directly to the sensor nodes.



### 3. System Evaluation of WP2 Components

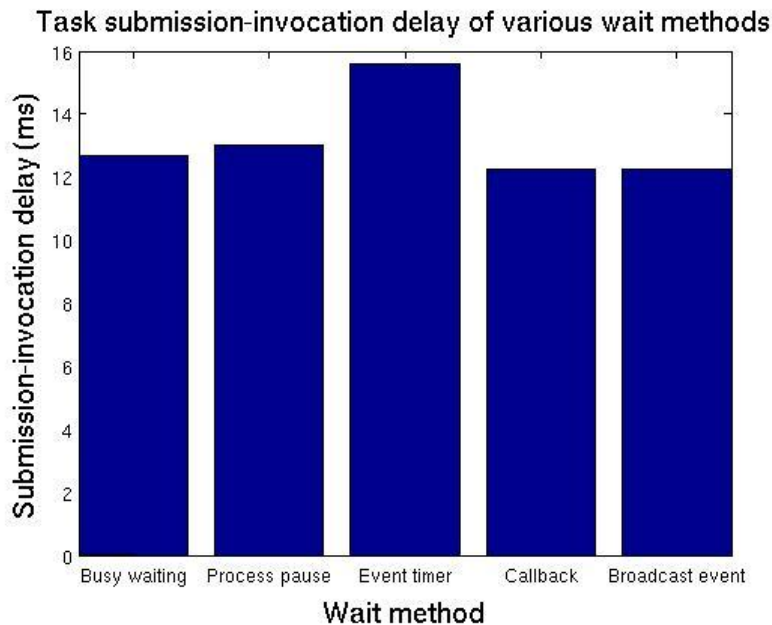
#### 3.1 Node Operating System

We have previously shown the effectiveness of our non-preemption macro in D2.3. We perform a more complete evaluation here, which uses the code profiler to learn the duration of the I/O access and measure the CPU usage of each implementation variant, including the newly added one based on Contiki event broadcast. The experiment lasts for 900 seconds to ensure a sufficiently large number of I/O tasks are generated. Each instance of I/O access is scheduled using a periodical Etimer, which inherently incurs a small phase drift at each invocation, therefore the phase difference between the I/O accesses and the rtimer interrupts are randomly distributed. Table 2 shows our test parameters.

I/O interval (ms)	Rtimer interval (ms)	I/O duration (ms)	Rtimer duration (ms)
333	50	7.5	5

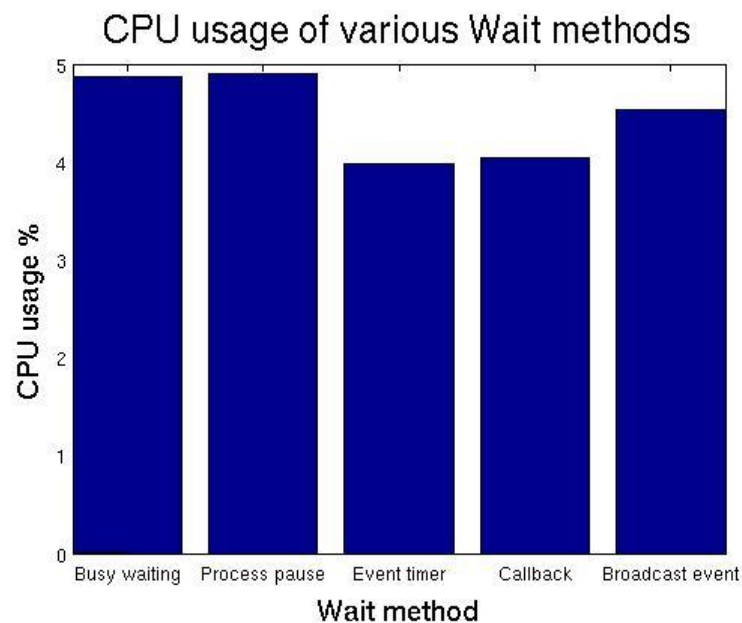
**Table 2: Test parameters**

Figure 1 shows the average delay between submission of an I/O task and its invocation, using the five WAIT\_FOR\_NON\_PREEMPTION implementation variants that differ in the wait method. The Etimer variant yields significantly larger delay the others, due to the coarse Etimer clock ticks that it uses for the estimation of invocation time. The callback and broadcast event methods have lowest delay, because their waits are promptly terminated by the completion of rtimer task.



**Figure 1: Average Task Submission-Invocation Delay Using Different Variants of Waiting**

Figure 2 shows the average CPU usage percentages using the macro. The Etimer method is the least CPU intensive, followed closely by the callback method. The event broadcast method incurs a larger CPU overhead than the callback method, because the Contiki scheduler implements event broadcasts by iterating through each process in the system.



**Figure 2: Average CPU Usage Percentages Using Different Variants of Waiting**

In conclusion, our multitasking node operating system supports concurrent tasks of hybrid real-time requirements with high performance and low overhead, and provides to the application developer a flexible, easy-to-use programming interface.

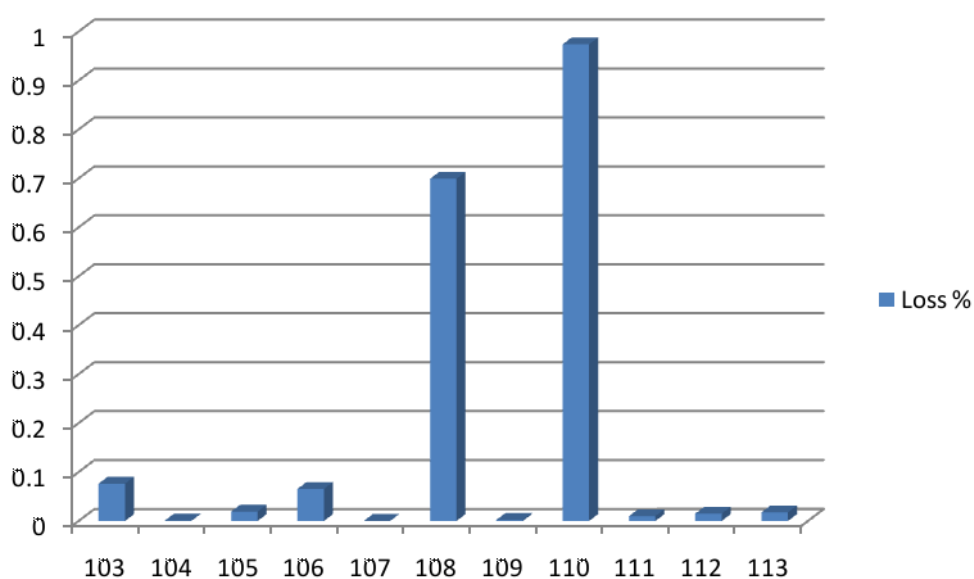
### 3.2 MAC Protocol

With the final layout of the GINSENG testbed decided upon, members of the GINSENG project were able to upload their different respective software components to monitor their operation in the real refinery environment for prolonged periods. The remote availability of this unique platform proved to be extremely beneficial and permitted bug-fixing and operational analysis to be carried out on a bigger scale than possible before. An iterative process of improvement whereby code would be uploaded, issues would be identified and solutions would be implemented ensued, and once considered completely stable, GinMAC was left to run for a number of weeks. We present here the data collected between a starting point of just after 9:20am on December 20<sup>th</sup> 2011 and shortly after 2:30pm on the 31<sup>st</sup> of December 2011, a continuous data collection period of over 11 days. We use the data to highlight some of the key characteristics of the GinMAC protocol and its performance.

#### 3.2.1 GinMAC Performance in the GINSENG Testbed

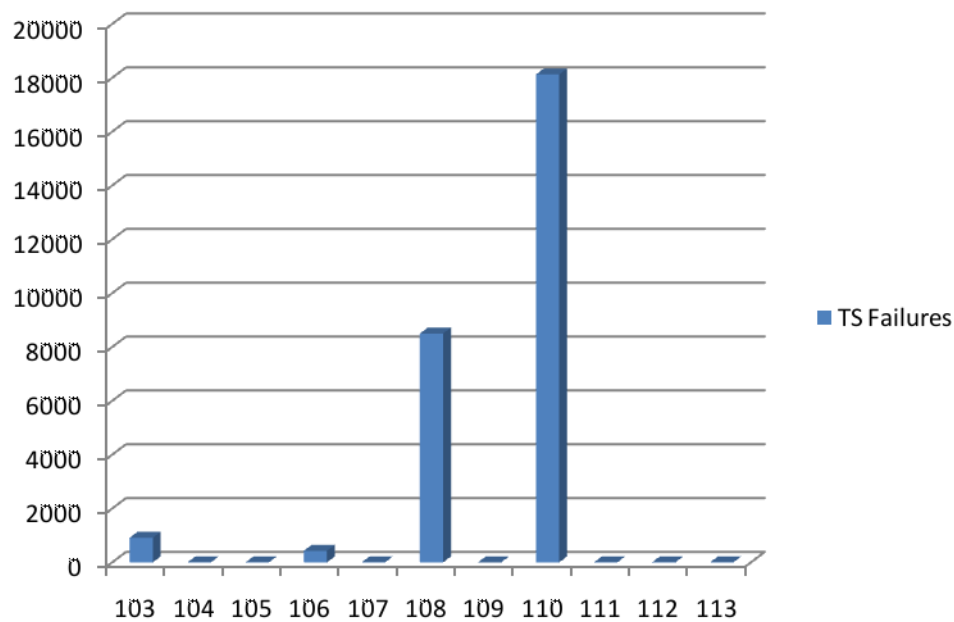
Over the course of the prolonged GinMAC testing period analysed, each node transmitted just over 1,048,300 packets. In Figure 3 we present the total end-to-end loss experienced by each node over that time. The first important point to highlight from this data is the units of measurement indicated on the Y-axis and more pertinently, the fact that none of the nodes experienced average loss above 1% of their total transmissions. Without further analysing the data this could be regarded as something of an achievement, however our confidence in GinMAC's capabilities has grown such that we are now far more critical of its performance than at the beginning of the project. With this in mind there is an obvious anomaly that gives us some cause for concern. Node 108 and Node 110 in the network experienced far higher levels of loss than any of the other nodes. Both of these nodes were connected to Node 107 as part of Branch 1 of Network 1, with both nodes being leaf nodes of that branch.

Apart from the problems experienced in the isolated case of the bottom of Branch 1, the results from the other remaining nodes are extremely encouraging. All of the remaining nodes lost less than 0.1% of their packets, with a number of nodes (Node 104 and Node 107) experiencing very negligible loss whatsoever and in the case of Node 107, zero loss. One of the key factors that this shows is GinMAC's ability to offer relatively high levels of reliability to nodes located all over the testbed network layout. In particular it is able to support nodes that are a number of levels (or wireless hops) deep into the network in the same way it supports those nodes that are directly connected to the Sink. Whilst localised incidents of packet loss may still occur (which is something that increased use and deployment experience would help to resolve) the ability of GinMAC to support nodes throughout the tree in many ways show that it is capable of achieving its intended goals of providing reliability too wireless sensor nodes in configurations where it would not otherwise exist.



**Figure 3: Final Testbed Layout - Total End-to-End Loss**

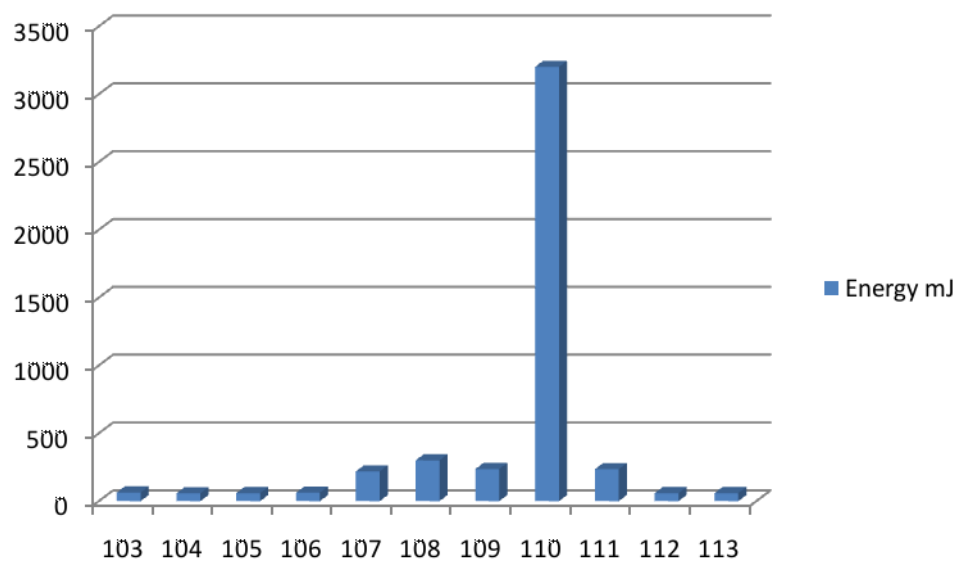
The total end-to-end packet loss results obviously point to problematic links between Node 108 and Node 110 and their parent Node 107. As with the testing we performed when initially configuring the network, we again analysed the occurrence of time-sync failures to identify if they correlated with the end-to-end packet losses experienced. The total number of time-sync failures experienced by each node is presented here in Figure 4. As can be observed, once again the incidence of time-sync failures directly correlates to the levels of packet loss experienced.



**Figure 4: Final Testbed Layout - Total Time-Sync Failures**

In addition to the reliability of the GinMAC protocol we also analysed the overall energy consumption of each node in the testbed over the testing period specified. The energy consumption results are presented in Figure 5 and again identify Node 110 as being a significant anomaly. In this chart we can see Node 110 experienced massively greater levels of energy consumption than all of the other nodes, with Node 108 experiencing the second highest level, but still ten times less (300.547mJ compared with 3201.438mJ) than its problematic neighbour. From further analysis of the breakdown of energy consumption on Node 110 we were able to identify that the bulk of the energy was used by the radio being placed in a receiving state.

As with the performance results, the energy consumption levels of the nodes not related to the problematic link in Branch 1 produced very encouraging results. All of the other nodes behaved as expected; the nodes that were not directly connected to the Sink (and therefore don't have to forward the traffic on behalf of any child nodes) only used on average around 60mJ of energy during the testing period analysed. In addition the nodes which were directly connected to the Sink (namely nodes 107, 109 and 111) consumed around 230mJ of energy.



**Figure 5: Final Testbed Layout - Total Energy Consumption**

The results taken from the prolonged testing period at the GINSENG testbed clearly show an uncontrolled anomaly occurred that affected nodes on Branch 1. From analysing the data we gathered further and from our experience of testing in the Sines refinery environment we believe that these nodes (Node 108 and Node 110) were negatively affected by an uncontrolled source of radio interference (i.e. not by other nodes in the GINSENG network) in the specific area that they were located in. However, whilst this interference did degrade the performance of the two nodes specifically affected, it also highlighted the GINSENG network's ability to continue operating effectively even in the presence of such an adversity. As our performance and energy results show, the GINSENG system continued to support very high levels of reliability using minimal levels of energy throughout the rest of the network. This demonstrates that the GinMAC protocol offers an encouraging level of resilience when all aspects of the network are not functioning as desired. This can be seen as a significant achievement by a MAC protocol alone since it cannot do anything if the physical link it is operating on is damaged beneath it.

### 3.2.2. Conclusion

The GinMAC protocol is one of the key components of the GINSENG system and from the beginning it had a clearly defined aim, to bring reliability and deterministic data transmission to otherwise unreliable wireless sensor networks. Over time the complexity of GinMAC increased and more and more features were added to improve its overall performance. The features and capabilities of GinMAC have been presented and documented in a number of places (most significantly in D2.3). Since that point the two most significant threads of development that have been undertaken with the GinMAC protocol have been:

1. An ongoing effort of bug fixing to try and produce the most stable piece of resulting software possible.
2. The development of an optimised version of the GinMAC protocol, expressly aimed at improving its accessibility and usability for other researchers and engineers wishing to carry out further work in this area, GinLite.

Our focus therefore has been on how well GinMAC is able to achieve its original aims now it is in its final most stable state of implementation, and also to provide some background of how GinLite has evolved as a result of the further evolution of the GinMAC implementation.

From carrying out extensive testing across the GINSENG testbed we believe we have demonstrated GinMAC's ability to operate in a stable and optimal manner on real deployment hardware platforms in a real and harsh operational environment. The GinMAC protocol is born out of innovative new research and designs for wireless sensor network operation and has in turn fostered a collection of new and unique ideas to the wireless sensor network research community. Given this fact it is therefore a significant achievement to then not only implement and demonstrate the capabilities of these new ideas but to do so across a relatively large real world remote testbed. Our evaluation results show that GinMAC is still susceptible to problems and that further work could be carried out to try to deal with external sources of interference (or additional research could look into the suitability of other alternative low power radio interfaces). However, given the progress that the GinMAC protocol has seen since its inception as a research concept, we are very pleased with its demonstrated capabilities and its current status as a stable and well performing proof-of-concept.

As it stands the GinMAC protocol is very closely coupled with the requirements of industrial process automation scenarios and in particular, those of oil refineries. This is unsurprising given the nature of the GINSENG project and its focus on supporting the requirements of Petrogal. The clear focus brought about by this partnership has been an important benefit over the duration of the GINSENG project and has ensured that the resulting solution is of very high quality. However, at the final stages of the project it is important to consider how it may be most possible to capitalise on the important groundwork that has been carried out by the GINSENG project and therefore provide a means to allow other interested parties to lead on from where GINSENG will finish. We believe that a key factor in making this goal a reality is to ensure that the GinMAC protocol implementation is made more accessible by making it more generically applicable to other scenarios. GinLite achieves this by removing the aspects of GinMAC that are inherently linked with the specific industrial automation scenarios it was originally designed for. Also, by modularising its architecture we have made it possible to simply "plug-in" different features whenever they are required.

All of this will hopefully increase the impact that the GinMAC protocol and the GINSENG project in general and also increase the likelihood that the outcome of the project will be adopted and taken further forward.

### 3.3 Sensor Node Lifetime Prediction

#### 3.3.1 Evaluation

Using a different measurement in the lab testbed in Braunschweig, we have evaluated the accuracy of the offline power prediction compared to the online energy estimation. For this purpose we have used the calibration data presented earlier and have predicted the power consumption for each incoming sample about the estimated power consumption. We have calculated the difference between those two and calculate the relative error.

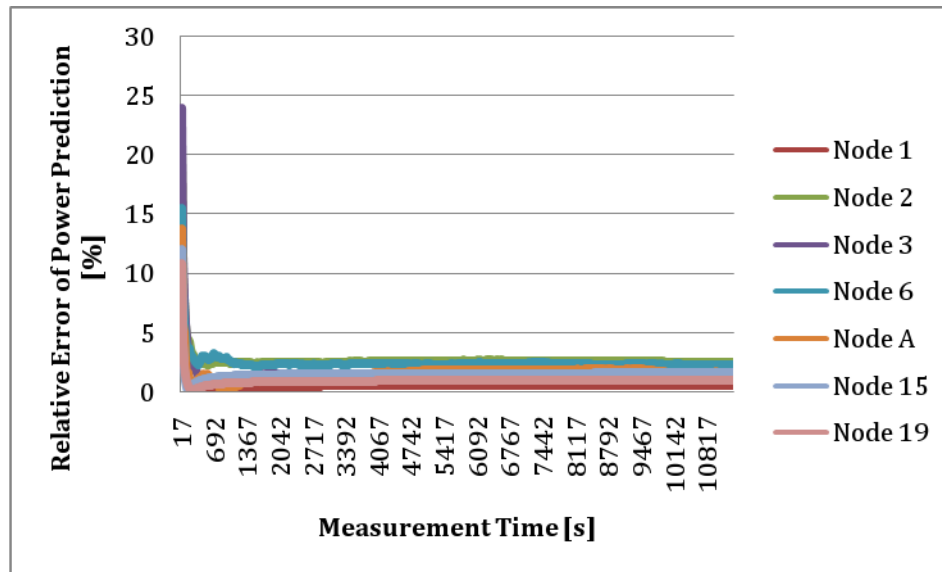


Figure 6: Relative Error of the Offline Energy Prediction.

Figure 6 shows the result from a 3.16 hour experiment. While we can see that the prediction has an error of up to 25% within the first 120 s of the experiment, the error consistently stays below 3.5% for all nodes within the rest of the experiment. Figure 7 shows an excerpt of the previous figure.

We can see here, that different than expected, the fluctuations of the error are independent of the tier in the GINSENG tree. Nodes 1, 2 and 3 as well as nodes 6 and A and nodes 15 and 19 are each at the same tier level. This confirms that the power consumption prediction does not suffer from principal problems but rather from microscopic differences in the behaviour of nodes.

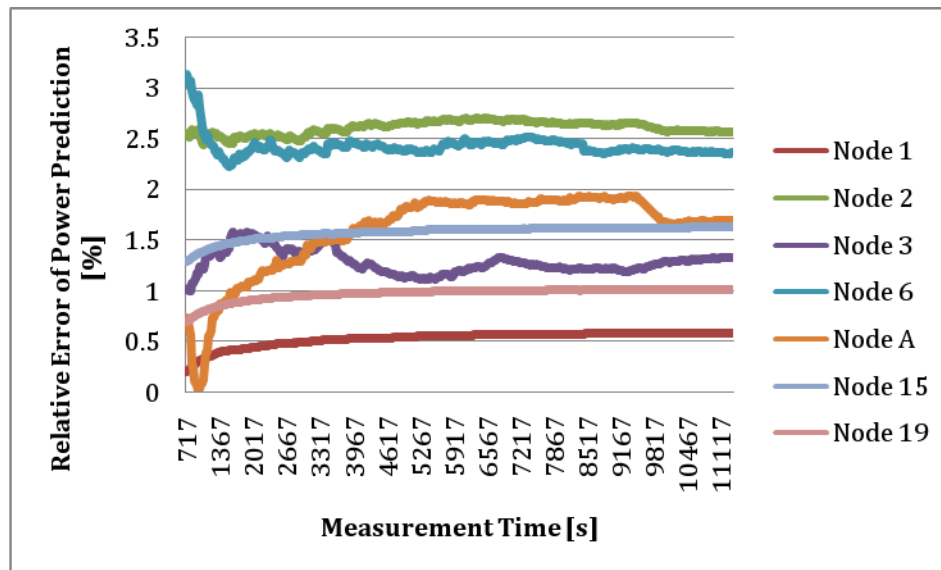


Figure 7: Excerpt of the Relative Error of the Offline Energy Prediction.

### 3.3.2 Limitations

Detecting misbehaving nodes using software-based power profiling has a number of limitations.

Most importantly, this mechanism can only detect certain errors. Short circuits on the board that may lead to significantly increased power consumption cannot be detected. Detecting such faults would require hardware support for online current and voltage measurements that is not present in the T-Mote Sky. Newer sensor nodes such as INGA [2] feature such hardware and implementing the presented mechanisms on said hardware may improve accuracy and resolution.

Furthermore, the accuracy of the presented mechanism is only within 3.5% of the reality. While this may be refined in the future, there will always be a certain inaccuracy. Events such as packet loss, etc. are not predictable and power consumption prediction will always suffer from such phenomena. However, integrating online feedback about such events into the prediction may allow a higher accuracy.

Due to the limited accuracy, misbehaviour has to significantly impact the power consumption of a node in order to be detected. Deviations of significantly less than 3.5% cannot be detected due to the noise that is connected with this prediction method. Further refinement of this approach may lower the bar, so that deviations of less than 3.5% may be detectable in the future.



### 3.4 Performance Debugging

We have presented in D2.4 the evaluation result of performance debugging of a 15-node indoor testbed. Here we present an evaluation based on the test runs conducted in the Sines testbed, with a focus on network performance. Several components of performance debugging approaches researched are included in the deployment on the Sines testbed. These include MIB, data piggybacking, and dispatcher.

#### 3.4.1 Sines Testbed

Deployed at Petrogal’s oil refinery in Sines, the GINSENG testbed consists of two separate networks, Network 1 and Network 2. Both networks connect wireless sensor nodes that are located at strategic points around the refinery up to a single portable office. For the purpose of illustrating the GinMAC protocol performance in the refinery deployment we chose to highlight the data acquired from Network 1, the layout of which is illustrated in Figure 8:

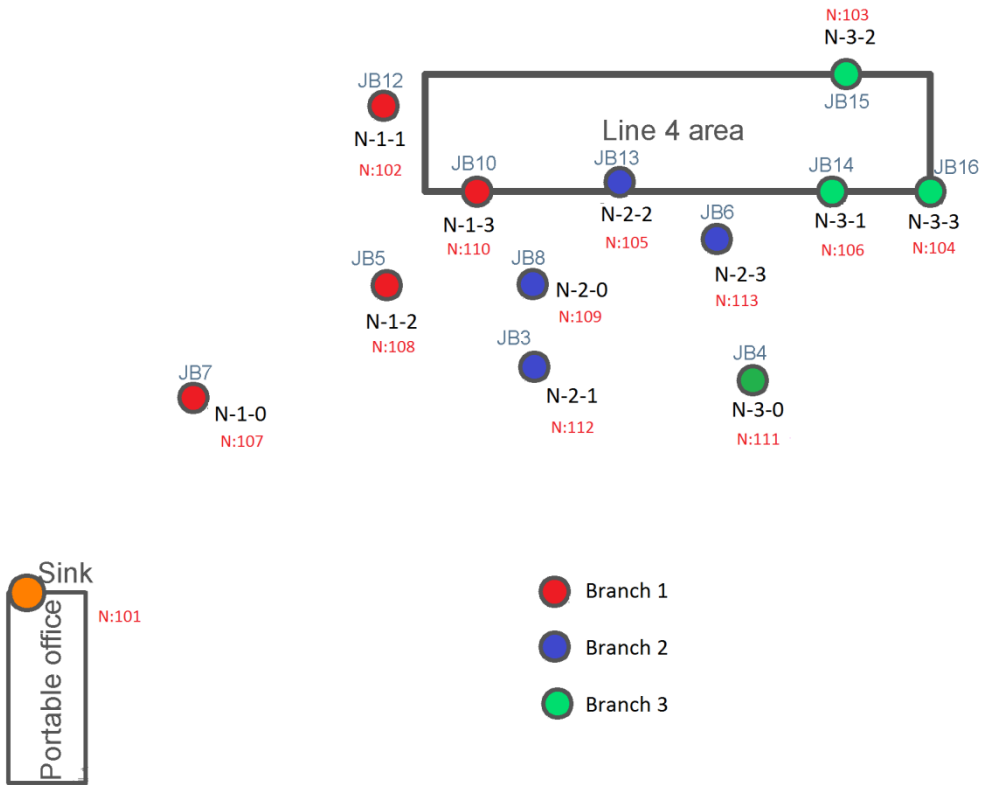


Figure 8: Diagram of Network 1 Layout

The locations of the nodes in Network 1 are illustrated to scale in Figure 1. As can be seen from the diagram, Network 1 has a compacted shape which allows for a number of different topologies to be supported. The final topology selected was 3-3, this topology allows all of the nodes to be at most two hops from the sink, which in turn ensures that less forwarding is required and therefore the epoch can become shorter.

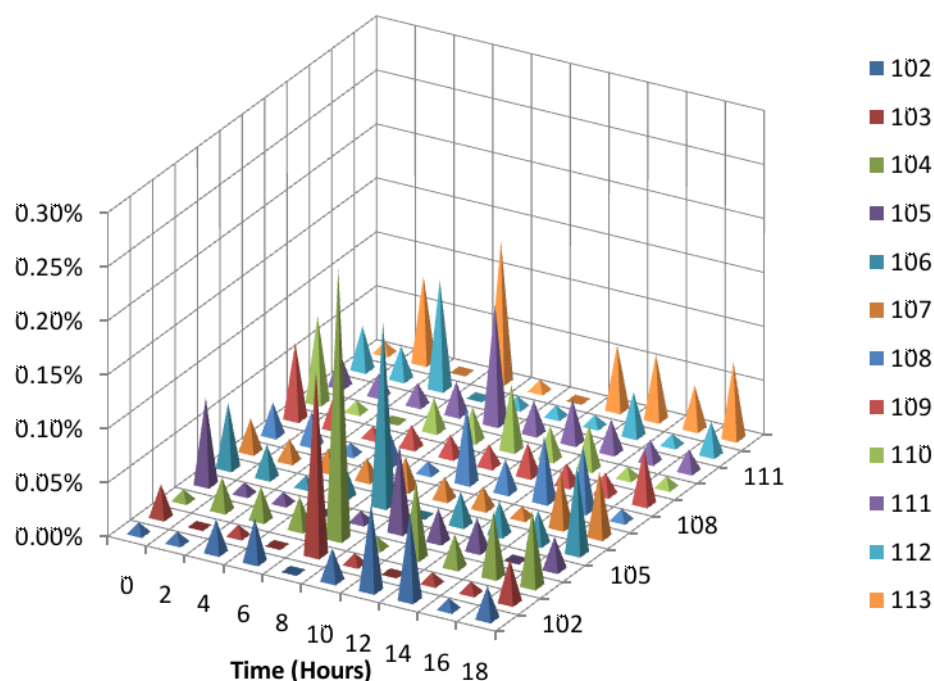
In order to gather information about the performance of the GinMAC protocol, a number of performance related messages were implemented throughout the system. We used 3 different message types in particular to acquire the results we present here:

- Node Message
  - This message type reports information related to data transmissions and is produced by each individual node.
- Energy Message
  - This message type reports detailed information related to energy consumption and is also generated by each individual node.
- Data Message
  - A separate message of this message type is generated by the dispatcher for each node, allowing end-to-end characteristics such as packet loss to be measured.

### 3.4.2 Testbed Configuration and Evaluation

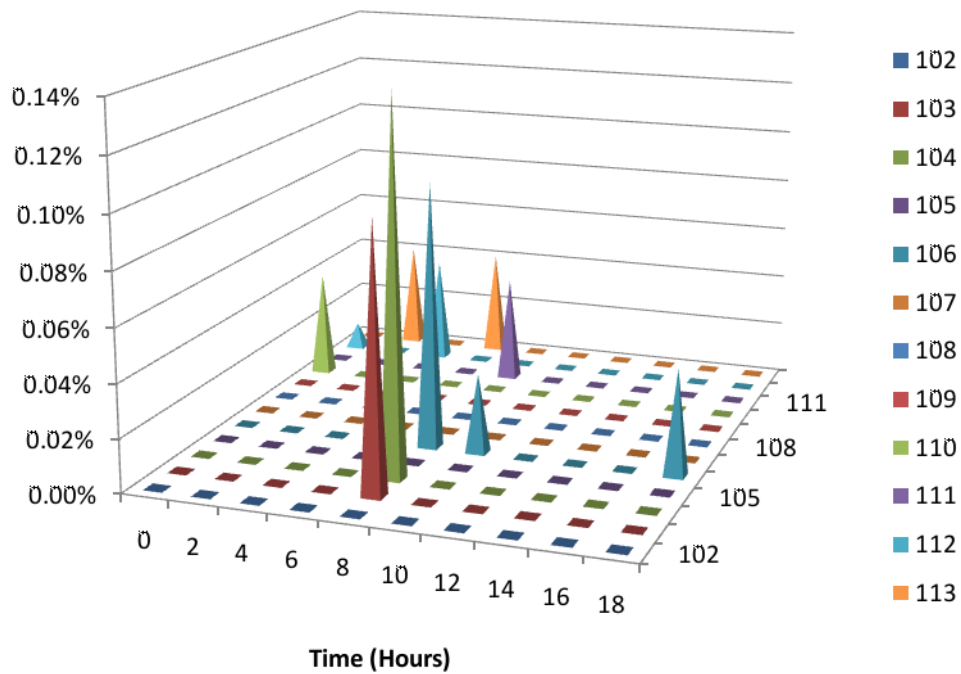
In the summer of 2011 the GINSENG testbed at the Sines refinery was configured for use in a final static state that would remain as the testbed layout for all subsequent evaluations (i.e. when Network 1 and Network 2 were configured). At this time a number of experiments were performed to measure the link loss rate, end-to-end loss rate and time-sync failures that occurred using GinMAC in its state at that given time. These performance results were captured from a 20 hour test run performed on the 22<sup>nd</sup> of August 2011 using channel 14 for all wireless communications.

All the links in this period of testing performed in a similar way with regards to link loss. Figure 9 presents the link loss rates encountered and shows that the highest loss experienced was of 0.25% over a 2 hour window between node 104 and 111 which equated to a loss of 18 packets out of a total of 7201 transmissions. The highest number of packets lost on any particular link was between node 111 and the sink with 32 packet losses after 28799 transmissions between the 8<sup>th</sup> and 10<sup>th</sup> hours of the evaluation.



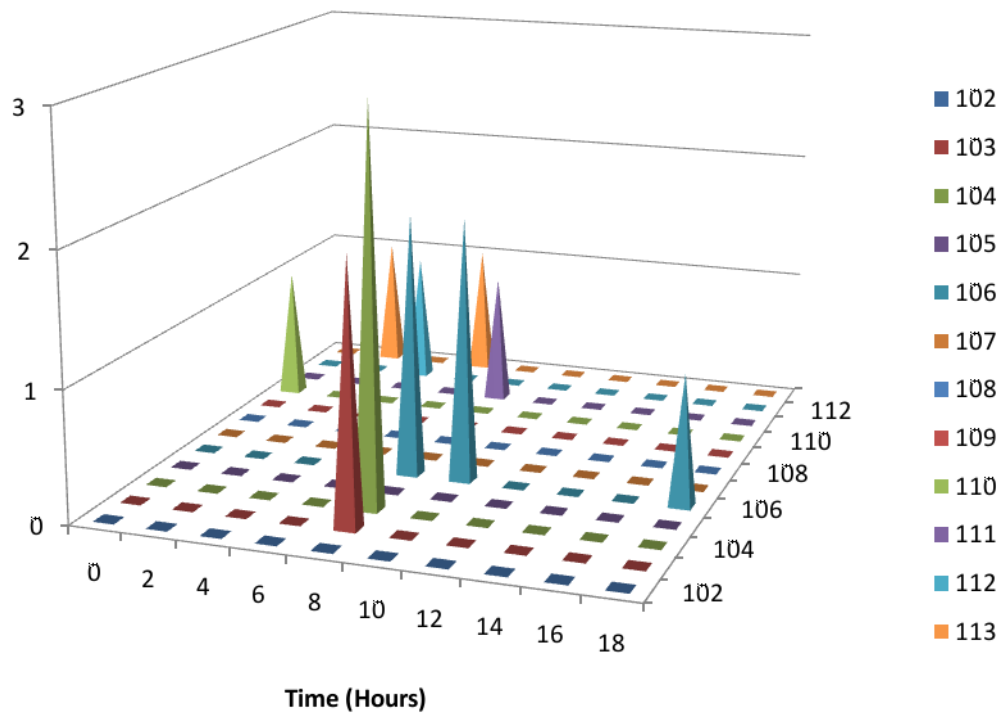
**Figure 9: Network 1 Initial Configuration Testing – Link Loss Rate**

Figure 10 shows the end-to-end losses that occurred in this testing period, with the highest being 0.14% for node 104 again between the 8<sup>th</sup> and 10<sup>th</sup> hours. These losses can be correlated to the high link losses of its parent node 111 and the Sink as mentioned above.



**Figure 10: Network 1 Initial Configuration Testing – End-to-end Loss Rate**

In Figure 11 we highlight the time-sync failures that occurred during our testing period. In these results we can see that the highest concentration of these failures again occurred between the 8<sup>th</sup> and 10<sup>th</sup> hours of operation again, which can be used to further explain the high end-to-end losses seen in this time window. From further examining these results we came to the conclusion that time-sync failures were the main cause of end-to-end losses and not individual link losses as these are mostly solved through our strategy of providing one transmission slot for each scheduled transmission.



**Figure 11: Network 1 Initial Configuration Testing – Time-sync Failures**

## 4. Conclusion

System evaluation of the final versions of WP2 components have led us to the satisfactory conclusion that they together provide a solid foundation for wireless sensor network applications operating in harsh industrial environments.

## References

- [1] Felix Büsching, Ulf Kulau and Lars Wolf, "Demo: INGA - An Inexpensive Node for General Applications", in Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, Sen-Sys '11, Seattle, WA, USA, ACM, 2011.
- [2] Da Cunha, A.B., de Almeida, B.R., da Silva, D.C., "Remaining Capacity Measurement and Analysis of Alkaline Batteries for Wireless Sensor Nodes", IEEE Transactions on Instrumentation and Measurement, Volume 58, Issue 6, pp. 1816 - 1822, June 2009.