# IRIS: A Flexible and Extensible Experiment Management and Data Analysis Tool for Wireless Sensor Networks

Richard Figura[1], Chia-Yen Shih[1], Songwei Fu[1], Roberta Daidone[2], Sascha Jungen[1], and Pedro José Marrón[1]

[1] Networked Embedded Systems, University of Duisburg-Essen, Germany
{richard.figura,chia-yen.shih,songwei.fu,pjmarron}@uni-due.de,
sascha.jungen@stud.uni-due.de
[2] Department of Ingegneria dell'Informazione, University of Pisa, Italy
r.daidone@iet.unipi.it

**Abstract.** Performing field experiments is a key step to validate the design of a Wireless Sensor Network (WSN) application and to evaluate its performance under various conditions. We present an experiment management and data analysis tool called *IRIS* that offers effective management of various configuration settings for WSN experiments. One special feature of IRIS is its extensibility. That is, IRIS allows the developer to define customized functions for application-specific data analysis and performance evaluation. Other main features include: enabling the interaction with the deployed WSN at runtime for fine tuning the experiments and providing graphical presentation for visualizing the collected data as well as the processed results. We highlight the advantages of IRIS for the WSN application development in different experiment phases. Furthermore, we demonstrate the usefulness of IRIS with two real-life WSN applications to show that IRIS can be integrated to develop an application and can greatly help in performing experiments more efficiently.

**Keywords:** Wireless sensor networks, Data processing, Experiment management, Data analysis, Data visualization.

## 1 Introduction

*Wireless Sensor Networks* (WSNs) offer a pragmatic option for acquiring physical parameter measurements, so many applications, e.g., habitant monitoring, surveillance and industrial automation, have considered deploying application-specific WSNs. Setting up a suitable WSN involves an iterative process of developing the necessary WSN application, deciding on the deployment topology and evaluating the application performance. The developer often conducts numerous empirical experiments or testing cases with various application parameter settings and network configurations. Once the experiment data is available, the developer needs to define how to process and analyze the data in order to evaluate the application performance.

M. Zuniga and G. Dini (Eds): S-Cube 2013, LNICST 122, pp. 94–110, 2013.

The tasks of handling experiment settings, evaluating the application performance in order to determine the *best-fit* WSN deployment is time consuming and error-prone. It is easy to see that effective management on experiment configurations and performance evaluation is the key to the success of WSN application development, especially when a complex application requires several WSN deployments. For this purpose, we offer an integrated solution that enables effective experiment management and data analysis for performance evaluation. Our work was motivated when preparing experiments in an EU project, PLANET [1], for a pollution monitoring application in the *Doñana Biological Reserve* (DBR) [2], Spain. The application requires a long-term WSN deployment in order to acquire physical measurements to monitor the target environment. The measurement data is either delivered by the connected WSN or collected by unmanned aerial or ground vehicles (UAVs or UGVs). With the time and hardware limitation in DBR, we needed to efficiently perform concurrent experiments with various network configurations and parameter settings. Therefore, we developed a tool, formerly known as IMAC [3], which provides a primitive mechanism for experiment management and on-site data analysis. The use of IMAC greatly helped us in accomplishing the experiment objectives.

We present in this paper the successor of IMAC, called IRIS[1], with many augmented features to IMAC. Our goal is to provide an integrated and flexible solution for experiment configuration management and performance data analysis in order to facilitate the WSN application development. We particularly address IRIS' main features in several aspects. First, IRIS provides a mechanism for managing WSN experiments. The developer can use IRIS to automate the application installation procedure, to iterate the experiment with different configuration and parameter settings, to create customized logs for different experiment purposes, etc. Second, for data collection and result analysis, IRIS emphasizes the extensibility by allowing the user to specify required data message formats and to flexibly define necessitated functions for data processing. The user can also develop the application by integrating IRIS and implement the program logic using functions. Third, during the experiment, the developer can use IRIS to interact with the deployed WSN in order to fine tune the parameter settings for higher performance or for debugging purpose. Finally, IRIS also includes graphical interface for visualizing the status of data collection as well as analyzed results. IRIS integrated JFreeChart [4] to generate figures with line charts and bar charts for clear data presentation. With these features, IRIS can support experiment tasks in different phases of experiments including *pre-experiment configuration*, *experiment runtime* and *post-experiment data analysis*.

The remainder of the paper is structured as follows. Section 2 describes the related work of experiment tools for WSNs; Section 3 gives an overview of the IRIS tool and elaborate the main features; Section 4 demonstrates the usage of the aforementioned features in different experiment phases; in Section 5, we show two case studies, in which IRIS helps in the process of the WSN development; finally, we conclude our work and discuss the future work in Section 6.

[1] After the Greek goddess IRIS for the meaning for messaging and communication.

## 2 Related Work

Much work has been devoted for enabling WSN application experiments. WSN testbeds, for example, offer hardware and software for WSN experiment setup, WSN application installation, node reprogramming and experiment execution for performance analysis. The TWIST [5] testbed deployed by TKN (Telecommunication Networks group at Technical University Berlin) enables indoor experiments with heterogeneous node platforms and network reconfiguration; the CONET testbed [6] includes a graphical software that allows an intuitive experiment configuration. Similar to IRIS, these testbeds allow managing the WSN experiments. However, such testbed infrastructure is setup in a specific environment, and they are not built for performing on-site experiments with the real environment nor for processing the application data.

Several network analysis tools have been proposed to gather data from the physical environment to captures the network dynamics, SWAT [7] is a software tool that automates data collection and analysis of measurements for low-level wireless network properties. These properties allow a better understanding for the performance of protocols or applications in different environment. Other tools of this category are TRIDENT [8] and RadiaLE [9]. Similar to IRIS, these tools offer a user interface allowing users to interact with the testing nodes that gathers network parameters, to visualize the data packet and to process/analyze the data. The difference is that these tools gather raw data packet statistics such as received signal strength (RSSI), link quality indicator (LQI), noise floor, and define a fixed set of performance metrics, e.g., packet delivery temporal and spatial correlations and link asymmetries. IRIS does not limit the processing data type and allows the user to define application-specific processing function. We note that IRIS can share similar functionality if the metrics are defined as IRIS functions.

For analyzing application data, MATLAB [10] and Octave [11] are notable technical languages for performance analysis, algorithm development and model design. They also provide a rich set of built-in math functions for a wide range of applications such as communication, signal processing, computational biology, etc. Other tools such as SciDavis [12] and LabPlot [13] are free software for scientific data analysis and visualization. While these tools are powerful for data processing, they can not be used for run-time WSN data analysis. A set of tools such as MOTE-VIEW [14], SpyGlass [15], Nviz [16] and NetViewer [17] are commonly used for WSN data collection and visualization. However, they have limited capability for data processing.

Although above tools are specialized in either experiment management or data processing, they don't address the possibility to support both on-site WSN experiment and real-time application data processing. To the best of our knowledge, IMAC/IRIS is the first tool that provides an integrated solution for the above issues and offers a flexible mechanism for the user to define application-specific processing functions to meet different experiment purposes.

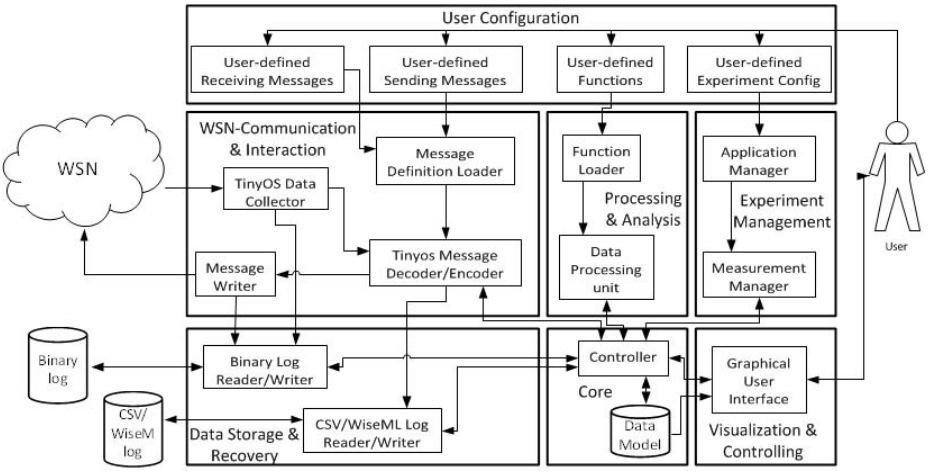


**Fig. 1.** IRIS modular architecture

## 3 IRIS

IRIS aims to provide an integrated solution for experiment management and on-site data analysis. The predecessor, IMAC, provides an environment for basic experiment journaling, data logging and visualization. IRIS enhances these features and especially puts emphasis on offering flexible data processing with customized functions. More importantly, IRIS provides an innovated method allowing to manipulate the collected data during or after an experiment.

### 3.1 Architecture Overview

We developed the IRIS tool in Java as a modular system, which is flexible and extensible. Each IRIS module carries out one or more main features of IRIS. Figure 1 shows an overview of the IRIS' design architecture.

The first module, *User Configuration* (UC), takes user input for configuring experiments, processing data and specifying messages used to interact with the WSN. This module also provides the user-configuration to other modules. To enable automated application installation and experiment measurement management, we introduced an *Experiment Management* (EM) module. The *Processing and Analysis* (PA) module carries out IRIS' capability of processing experiment data using user-defined functions, while the *WSN-Communication Interaction* (WCI) module enables the interaction between the user and the deployed WSNs running TinyOS applications. Other modules such as *Data Storage and Recovery* (DSR) module and *Visualization and Controlling* (VC) module are responsible for storing the log data and for visualizing the experiment output, respectively.

The core of IRIS is centered at the *Controller* component and its associated *Data Model*, which stores all the message structures and experiment data imported into IRIS. The Controller defines the logic for managing these data as

well as the interaction with all other components. Together all the modules carry out the main features of IRIS as described below.

### 3.2 Features

We highlight the main features of IRIS regarding experiment management, data collection, WSN interaction, data processing and data visualization.

**Experiment Management.** The first feature of IRIS is that it provides a set of useful utilities that allows performing experiments efficiently.
*Automated Application Installation* The WSN developers have a common experience, i.e., repeatedly installing the application onto many sensor nodes with unique IDs. IRIS' *Application Manager* automates the installation process, and makes the task of matching the hardware devices with their node IDs and required applications less erroneous, especially when the application has strict limitation on the hardware for installation. This feature is extremely useful and have greatly shortened the preparation time of our experiments in DBR.
*Measurement Management* When running a series of experiments, another issue is to match the recorded data to different experiment settings. In IRIS, an experiment can consist of several experiment runs, or "measurements", and for each measurement IRIS organizes a set of log files for incoming and outgoing messages in the binary, CSV and WiseML formats. For each experiment, IRIS generates a metadata file that specifies the general information and experiment statistics including an experiment ID, the start time, the hardware list, list of TinyOS applications, the number of measurements and their corresponding settings, etc. Moreover, for the applications that stores the sensor data in the flash, IRIS provides a utility for offloading the data (see Section 3.2). IRIS associates the offloaded data with its measurement and logs the flash data in a similar way.
*Customized Logging* IRIS provides a flexible and convenient mechanism allowing the user to customize the logging format. The user can define application-specific format for every incoming or outgoing message. Together with the Data Processing Unit, it is possible to define functions to directly manipulate collected raw data and to store the end result to a log file freely. This bypasses the step of raw data storing and therefore greatly increases the efficiency.

**Data Collection and WSN Interaction.** The second feature of IRIS allows user interactions with the deployed WSN in both directions, i.e., collecting data from the WSN and sending command messages to the network in order to control the experiment flow. Such interaction requires the knowledge of the message structures. IRIS allows the user to extend the message set by defining new message structures with the templates generated by the TinyOS *Message Interface Generator* (MIG) tool.
*Data Collection.* To collect WSN data, IRIS can be connected to one or several base stations via serial ports. Every base station collects messages and delivers those messages to the WCI module to handle the messages. The handled message

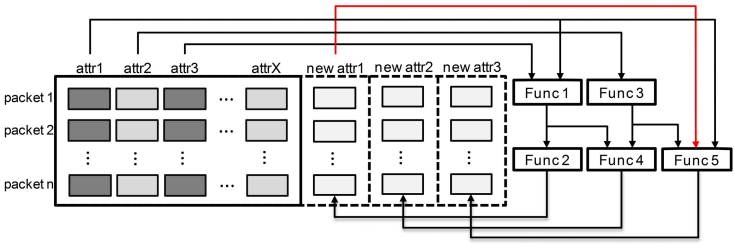


**Fig. 2.** Different types of composition functions in IRIS

are then stored by the DSR module depending on whether the message can be parsed. If so, the message fields are converted into IRIS attributes and the messages are stored as attribute values. The attributes and their values are stored in a human readable format, i.e., CSV or WiseML for further processing or result analysis. In addition to collecting data from the base station, IRIS also provides a TinyOS program called FlashReader for offloading the data from the flash memory. This application reads out the flash content of a node and sends the data over the serial link, through which IRIS parses and logs the data.
*WSN Interaction.* IRIS allows the user to control the experiment flow and to interact with the deployed WSN by dispatching the arbitrary user-defined AM messages. Such feature is advantageous for adjusting experiment parameters and for program debugging. To send a message to the WSN, the user needs to connect IRIS with one or more base stations, and to specify three parameters: the message template (type), attribute values and the base station for sending the message. When configured with the message template, IRIS creates its message instances and serializes them in the binary form for sending.

**Processing and Analysis.** IRIS offers an innovated way for extending the data processing capability of IRIS by flexibly defining application-specific functions to process data at the packet level. These functions can be applied to collected data during runtime or after the experiment. Such feature is especially useful for runtime data analysis, application debugging and customized logging. In IRIS, a function is created via the definition of a *function template*, which specifies the number of input attributes (ports) and the number of static configuration values. The user can initiate a function instance by *wiring* the target attributes to the input ports and deciding on the constant values during/after the experiment. Currently, IRIS provides a rich set of function templates for data processing. However, the user can extend it with user-defined templates. With such feature, the user can use IRIS not only merely as a data processing tool but as a building block for a WSN application (refer to Section 5.1 for an example).

It is worth noting that IRIS' function template has a unique feature that distinguishes IRIS from other data processing tools. That is, it is *composable*, meaning that the function template can take inputs as attributes that are either converted from the message template or created by other function templates. Figure 2 depicts the flexible composability of the function templates. For in-

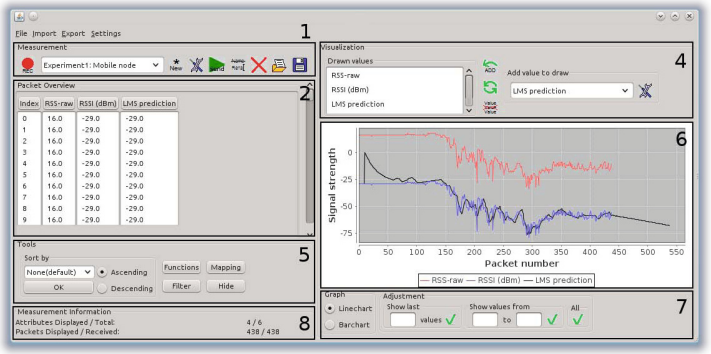


**Fig. 3.** The graphical user interface of IRIS

stance, a normal function template can take message attributes as inputs (Func 1); one can have a function output as input and creates a new attribute (Func 2); a template as Func 5 can take input a message attribute and a newly created attribute. With composable function templates, the user can implement a variety of data processing algorithms that directly access the data without redundant storing and retrieving for further processing, and thus can obtain the experiment result with less storage resources and significantly reduced time.

**Visualization and User Control Interface.** Visualization and user control interface are basic and yet important features for a tool like IRIS. The user interface of IRIS consists of several parts for the user input and for visualizing the experiment data. Figure 3 shows a snapshot of the main graphical user interface (GUI) of IRIS. Through the IRIS' GUI, the user can perform experiment management, including creating an experiment profile, initiating data collection, sending control messages, store/load the experiment data, defining the function instances for data processing, etc. For each measurement, IRIS first displays general information about the measurement. When the data is available, the Packet View displays the content of the messages based on their attributes and can be customized to only show required attributes. Additionally, IRIS is integrated with JFreeChart, a library for generating line charts or bar charts. It allows the user to zoom into the chart, to save pictures of generated charts and to change the scale of the view for a better visualization about any attributes.

## 4 Usage

With above features, IRIS supports the user to perform tasks in the different experiment phases: *pre-experiment*, *runtime* and *post-experiment*.

### 4.1 Pre-experiment Phase

In this phase, the user can use IRIS to perform the following four tasks.

*Automatically Installing Applications* To automate application installation, the user needs to provide two configuration files, which specifies a list of application node IDs with the TinyOS applications to be installed, and a list of the IDs with their hardware addresses, respectively. IRIS also provides utilities to generate these lists easily. Once both lists are available, the user can attach any number of nodes to the USB ports and start application installation.
*Defining Message Templates* IRIS requires the message templates in order to access the content of incoming and outgoing messages. Therefore, the user needs to define application-specific templates as described previously. If an incoming message template is missing, IRIS will treat the collected messages as a binary stream. In contrast, the outgoing message structure must be defined, otherwise the message values cannot be serialized and therefore cannot be sent. With the message template, each message field is converted to an attribute, which stores the name of a message field together with all of its values. The definition or the type of an attribute is opaque to IRIS. However, such definition can be important for data processing e.g. indexing the packets by their source ID. Thus, IRIS allows *attribute mapping* to assign attributes with the type information for an experiment. These typed-attributes can then be used in the functions, e.g., a filtering function based on the node ID as a type. IRIS has already defined a set of mappings, e.g., the sequence numbers and the source node-ID, and the set can be easily extended by the user.
*Implementing Customized Function Templates* IRIS allows the user to process the experiment data by defining customized function templates, which can be categorized into three types. The first type is the *monitoring* function, which requires no output value but only examines the attribute values of incoming messages, e.g., an alarm function, which displays a warning message when an attribute value reaches a certain threshold. The second type is the *scalar* function, which only outputs a scalar value. This function is normally used to generate an aggregated result such as PRR. The last type of functions outputs a new attribute. Most data processing functions fall into this category. Such functions are normally defined to process the original message values in order to generate a new value, e.g., filtering functions and transformation functions. The newly defined function template must be implemented in Java and the compiled class must be placed in a specific folder so that it is available to IRIS. IRIS defines a set of base classes for customized function definition. The user defines the function logic by overriding the method computeData(), which is invoked every time when a new packet is received. Listing 1.1 shows an example of a snipped definition of a CC2420 RSSI conversion function template.

**Listing 1.1.** A User-Defined CC2420 RSSI Conversion Function Template

```
public float[] computeData(float[][] val,float[] set){
  float[] res = new float[val[0].length];
  for(int i=0; i<val[0].length; i++)
    res[i] = val[0][i] - 45;
  return res;
}
```

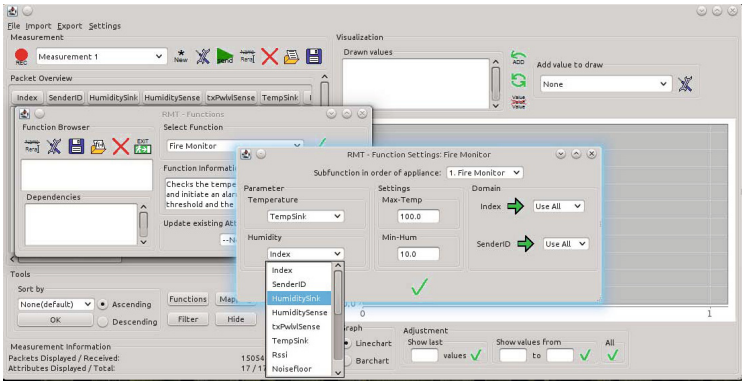


**Fig. 4.** Creating a fire monitoring function instance using attribute mapping

*Creating Experiment Profile* To initiate an experiment in IRIS, the user can create an experiment profile, which records logistic information specified by the user regarding performed experiment. For each experiment, IRIS creates a directory to hold all generated related files including the application files, log files, etc. If no log file is specified, IRIS automatically stores the binary and CSV formats of every incoming and outgoing message.

### 4.2 Experiment Runtime

Once IRIS is configured with required templates for messages and functions, and with required experiment information, during the experiment runtime, the user mainly uses the IRIS GUI to collect/process/visualize the sensory data and to interact with the WSN. After the user starts the data collection process, the incoming messages are first logged in binary, CSV or WiseML format. In addition, the message attributes are automatically displayed in the Packet View (see Area 2 of Figure 3) if the messages can be parsed.

For interacting with the WSN, IRIS allows the user to flexibly send messages in three different ways. First, the user can create a message using the GUI and send it to the network if the message template has been defined. The second option is to define a function template for sending messages by using the "IRIS_Mote" class provided by IRIS. The last method is useful when the user likes to introduce a series of messages. IRIS provides a scripting language for specifying sending commands. To perform runtime analysis, the user can create a function instance by selecting the required function template from the GUI (see Area 5 in Figure 3). The selected function can then be configured by mapping the input ports to the desired attributes and by defining the constant settings of the function. Figure 4 illustrates an example of creating a fire monitoring function with the specified minimum and maximum temperature thresholds for triggering the alarm.

For visualizing the runtime data, IRIS displays attribute values with a line graph or a bar chart. The user can choose the attribute from a pull-down menu

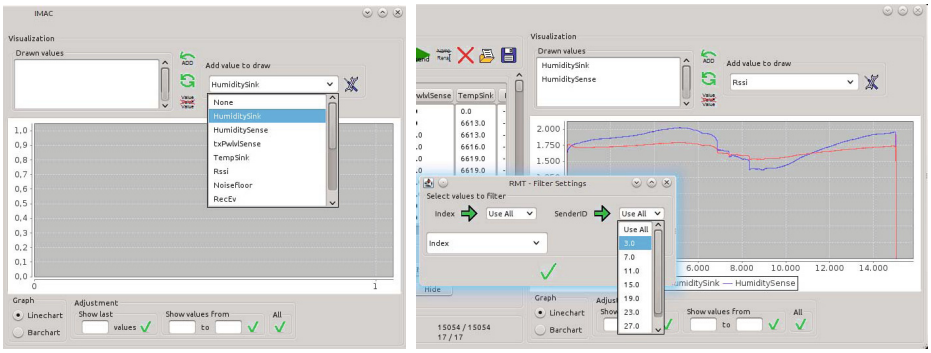


**Fig. 5.** (a) Adding the attribute "HumiditySink" to the line chart, (b) Applying a filter on the RSSI values for node with ID 3

as showed Figure 5a to view the change of the attribute values in real time. Furthermore, it is possible to filter the values by the attribute types. Figure 5b shows an example to only display the RSSI values from the node with the ID 3.

### 4.3 Post-experiment: Analysis and Management

When the experiment is complete, the user can apply previously defined functions to the collected data in a similar way as it for runtime processing. If the application requires the data stored in the flash, IRIS can automate the process of reading/erasing the flash by specifying the data structure and the volume-partitions. The default read-flash applications reads the whole flash content as a single volume and sends it via the serial port to IRIS. The offloaded data and any logged data can be visualized and processed within the Java GUI. Data can be loaded into different measurements using the following formats: binary, CSV or WiseML. Moreover, it is possible to load the data of two different measurements for comparing the measurements outcomes.

## 5 Case Study

In this section, we address the usability and extensibility of IRIS with two study cases: the first use case demonstrates an IRIS-integrated WSN application, while the second case uses IRIS for experiment management and data processing.

### 5.1 WSN Failure Detection and Diagnosis System

Wireless sensor networks are especially susceptible to unexpected environmental factors, radio interference, battery depletion and hardware vulnerability. To improve the robustness and reliability of WSNs, we have developed a *Failure*

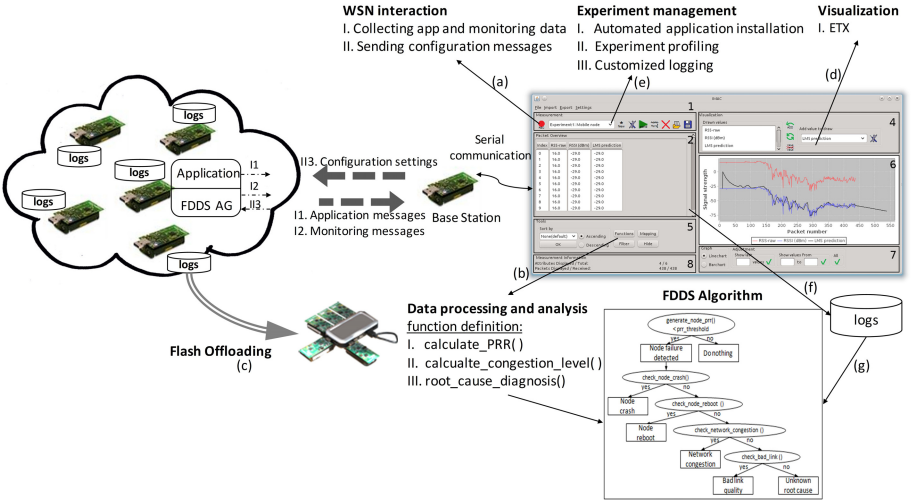


**Fig. 6.** Case 1: The FDDS network integrated with IRIS

*Detection and Diagnosis System* (FDDS) to provide analytical monitoring information regarding the presence of failures with their possible root causes. The implementation of FDDS integrates IRIS to carry out required operations. Additionally, to evaluate FDDS' performance, we conduct a set of experiments using IRIS to interact with the network with various parameter settings. Figure 6 illustrates the IRIS-integrated FDDS and IRIS' support for the experiments.

**IRIS-Integrated FDDS.** FDDS consists of two parts: (1) a set of 20 TinyOS agents (running on the TelosB platform), which periodically send monitoring information, and (2) a central control server (running on the PC), which analyzes the data for hardware/network failures and their root causes. FDDS requires collecting application and network monitoring data from the WSN. Thus, FDDS integrates the IRIS' data collection routine to gather these messages (see Figure 6a,I). Once the data is available, the FDDS control server performs real-time data processing for failure detection, and monitors parameters such as PRR, ETX [18] and the congestion level ($cl$) [19], which is defined as $cl = n(bp)/n(gp)$, where $n(bp)$ and $n(gp)$ are numbers of bad and good packets, respectively (A *good*/bad packet is the packet received without/with a CRC error). Each FDDS agent records the values of ETX, $n(bp)$ and $n(gp)$ in its flash during the last monitoring period, and includes these values in the monitoring messages to the control server. The server is implemented with three IRIS processing functions (see Figure 6b). The first two functions are *calculate_PRR()* and *calculate_congestion_level()*, which output the PRR and $cl$ attributes, respectively. *calculate_PRR*() needs one attribute as input (sequence number), while *calculate_congestion_level*() takes two attributes, $n(bp)$ and $n(gp)$, to calculate the value of $cl$. The third function, *root_cause_diagnosis()*, implements the fail-

ure detection and the root cause diagnosis algorithms. It takes three attributes, PRR, ETX and *cl*, as input, and outputs the root cause of the failure as a new attribute if a failure has occurred.

*Discussion.* Integrating with IRIS makes the development process of FDDS very efficient. FDDS draws support from IRIS in several aspects. First, with IRIS' data collection utility, we only need to define the required message types without additional code, and the gathered data is recorded in a customized format for later processing. Since IRIS has covered the typical and yet tedious data collection task, we could focus our efforts on optimizing the FDDS algorithm. Second, flexible IRIS function definition especially allows us to easily design the above functions to carry out the operations of FDDS. Third, to retrieve data stored in each agent's local flash, we use the IRIS' flash offloading utility to automatically download the flash data from all the nodes connected to the USB ports (see Figure 6c). Moreover, the flash data is stored in the customized log in the same format for the runtime messages. Without such feature of IRIS, we would have to manually download the flash data and to write a program for parsing the data and storing it in the correct format. Last but not least, FDDS uses IRIS' GUI for the user to view the network condition during runtime (see Figure 6d). This feature not only frees us from the GUI implementation and but also helps us in debugging and verifying FDDS' operations in different failure scenarios.

**FDDS Experiments.** The aim of the experiments is to evaluate the FDDS' performance on failure detection and root cause analysis. We first would like to know the accuracy of FDDS in identifying these failure causes. Therefore, we simulated 4 root causes of the failures at the nodes: battery depletion, bad link, node crash and node reboot. Moreover, we also study the impact of two different message sending rates (for both application and monitoring messages) on the responsiveness and the performance of FDDS. The experiment runs on the above IRIS integrated FDDS and uses IRIS' utility for introducing messages to the WSN in order to reconfigure different parameter values for both sending rates (see Figure 6a, II). IRIS manages the experiment profile and stores the collected experiment data in a particularly customized format, sorting with the timestamp and the parameter setting (see Figure 6e,f). For each experiment run, the result of the response time and the root cause is logged along with the corresponding parameter setting and the measurement profile.

*Discussion.* It is easy to see the advantages of IRIS with its effective environment for experiment management. Without this feature, the user typically needs to manually record the experiment logistics, to organize all collected data sets by placing them in proper directories, to associating them with the measurement settings and to store them in the customized format, etc. In the FDDS experiment, the experiment data is collected and stored based on the experiment profile and organized in the customized format. In addition, IRIS allows us to retrieve stored logs and to visualize the data flexibly in order to focus on

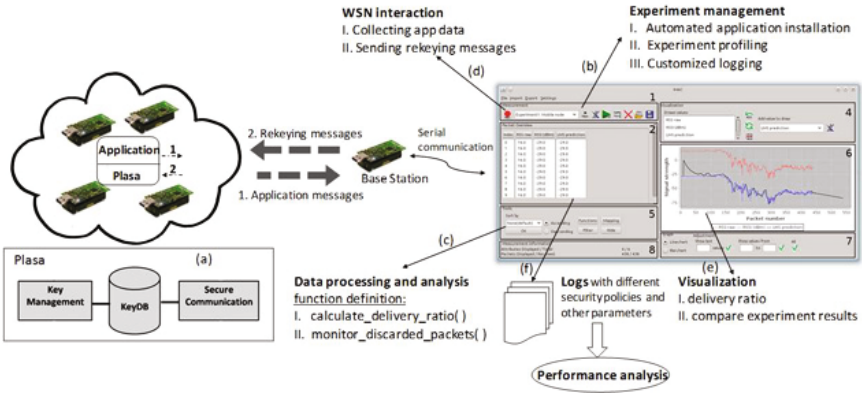


**Fig. 7.** Case 2: Secure communication experiments using IRIS

the monitored information. For instance, FDDS monitors the changes of *cl* of each node for failure detection. For post-experiment analysis, we apply the same FDDS function filtered by the sensor ID in order to calculate the *cl*. Furthermore, using the IRIS visualization tool, we can compare different experiment results using a combination of functions and filters and view the results in the graphical charts.

### 5.2 Secure Communication

In many realistic WSN applications it is fundamental to guarantee confidentiality and/or authenticity of messages exchanged within the network. The issue is that the developer needs to assure confidentiality, integrity or both, while preserving performance of a network of resource constrained sensor devices. In addition, it is a prudent cryptographic engineering practice to periodically refresh cryptographic keys in order to avoid cryptanalysis attacks. Thus, it is important to distribute and refresh cryptographic keys in an efficient manner and to lower the overhead for network performance [20,21]. To tackle the above issues, we implemented a security architecture to be used in the PLANET project [1]. The *PLANET Security Architecture* (*PLASA*) is composed of three modules: secure communication, key management and a keyDB for storing cryptographic keys (see Figure 7a). With these modules, PLASA is used as a transparent layer between the application and the remaining communication layers to secure communications and manage cryptographic keys. PLASA transparency relies on the secure communication module transparency. This module secures communications according to a *security policy*, which can dynamically change over time. Further details about this module can be found in [22]. In this study case, we use IRIS as an experiment tool to evaluate the performance of PLASA with different security policies and to study the impact of the rekeying frequency on the number of packets which are discarded because of authentication mismatches.

**PLASA Experiment Using IRIS.** The setup of the PLASA experiment includes 2 to 10 sensor nodes that periodically send a fixed amount of packets to the base station connected to IRIS, when the secure communication is enabled, PLASA secures the packets before transmitting them, while the base station unsecures the received packets and authenticates them. The performance evaluation of PLASA first involves in reconfiguring the node with many different settings to consider various security policies, and thus the experiment replies on IRIS for effective experiment management. We also heavily used IRIS to automatically installing the application linked with PLASA and specified security policy for each measurement (see Figure 7b). Different security policies influence performance with: (1) processing overhead due to security computations, and (2) communication overhead due to extra bytes added to the packet to allow the adversary to recognize the security policy, or because of the *Message Authentication Code* (*MAC*) appended to the payload. To know the impact of different policies on WSNs with different number of nodes, we define a processing function called *calculate_delivery_ratio*() to calculate the delivery ratio, which is defined as the ratio between the amount of secured packet transmitted by a sender and the amount of packets correctly received by the base station. By knowing the changes of this ratio, we can derive the overhead introduced by the security policy. Moreover, we defined another function called *monitor_discarded_packets*() to evaluate the impact of rekeying (see Figure 7c). During the experiment, we use IRIS to send different rekeying messages (Figure 7d, II) and to evaluate PLASA's performance with different policies and to monitor the number of discarded messages using the functions defined previously. The collected messages as well as the changes of the metric values, e.g., the delivery ratio, can be viewed on the IRIS' GUI (Figure 7e). Finally, the messages are stored in the customized logs for post-experiment data analysis (Figure 7f).

*Discussion.* The PLASA experiment demonstrates the IRIS' capability regarding effective management with a large set of experiment data, runtime WSN interaction, real-time data processing and visualization for result analysis. The experiment requires the sensor nodes to be reconfigured with various security policies. With the IRIS' installation tool, we only need to define a script program with a few lines of code and connect the nodes to the USB hubs. Similar to the FDDS experiment, we spend little efforts on collecting experiment data and managing data logs, and only need to define the required message types. The requirement of the PLASA experiment strongly highlights the usefulness of IRIS in runtime data processing. IRIS can calculate the delivery ratio of each node at runtime and provide plots of network performance over time. Note that the impact of the processing overhead on the delivery ratio cannot be evaluated with simulations. Collecting and analyzing a huge amount of experimental data without a tool like IRIS might be very complex. It is worth mentioning that the implementation of the two processing functions only counts for less than 100 lines of code in total. When performing rekeying, IRIS is extremely useful because it allows periodically injecting a rekeying message to the network without reinstalling the sensor node programs or forcing sensors to change their behaviour

to send rekeying messages. Thus, we force the refresh of the cryptographic key and observe the number of packets that cannot be correctly unsecured during the network transient state, in which nodes do not share the same key. Finally, IRIS allows storing the experimental results to compare them over time, or to collect statistics offline. This is very important because the feature makes it possible to have a deep evaluation on the impact of security policies.

### 5.3 The Limitation of IRIS

While aforementioned case studies show the main advantages of IRIS, several factors can hamper the efficiency of IRIS. We notice that IRIS, with one processing function, can process approximately the first 50,000 packets at the constant rate: e.g., average 50 ms per packet with the sending rate of 50 ms per second. After that, the processing time increases linearly and IRIS becomes less responsive. However, the actual packet number depends on the number of functions applied on the incoming data, the number of packets to be visualized, the number of attributes for processed messages and the number of active graphs. Among them, the number of functions used has a significant impact on IRIS' performance. We notice that the total processing time for the same amount of packets increases superlinearly as the number of applied functions increases. This is due to the data structure we use to organize the collected packets for data processing. We expect higher performance of IRIS with a new data structure. However, IRIS supports message caching. If the data needs not to be processed at the arrival of every packet, IRIS can be configured to activate the function after a specified number of receiving packets. This can greatly reduce the processing time.

## 6 Future Work and Conclusion

IRIS is a flexible and effective tool to support the WSN application for experiment management, data collection, real-time data processing, WSN runtime interaction, customized logging and visualization. The novelty of IRIS is its capability of managing the experiment data and its flexibility of allowing the user-defined functions for data processing. Other features like WSN interaction and visualization can also greatly help in experiment reconfiguration, data analysis, program debugging and performance evaluation. We have demonstrated the usefulness of IRIS with two real-life applications. We believe that the development of such a tool is valuable and the user can benefit from it when developing the WSN application.

While IRIS being a powerful tool, it can be enhanced in several aspects. First, IRIS' function currently can only process data as the floating-point type, so we plan to extend the data model to keep the type information of collected data so that more data types can be handled. As for the performance issue of IRIS, we are designing an optimal data structure. Finally, we would also like to evaluate IRIS' performance with more complex composition functions.

**Acknowledgments.** This work has been partially supported by PLANET, Platform for the Deployment and Operation of Heterogeneous Networked Cooperating Objects, funded by the European Commission under FP7 with contract number FP7-2009-5-257649(www.planet-ict.eu) and by TENACE, Protecting National Critical Infrastructures From Cyber Threats, funded by the Italian Ministry of Education, University and Research, under the PRIN Framework with contract number 20103P34XC (http://www.dis.uniroma1.it/˜tenace/)

## References

1. Planet, `http://www.planet-ict.eu`
2. Doñana biological reserve, `http://www.ebd.csic.es`
3. Figura, R., Jungen, S., Soleymani, R., Shih, C.-Y., Marrón, P.J.: Demo Abstract: IMAC, Enabling Flexible Configuration and Result Analysis for Diverse Wireless Sensor Network Experiments. In: EWSN (2012)
4. JFreeChart, `http://www.jfree.org/jfreechart/`
5. Handziski, V., Köpke, A., Willig, A., Wolisz, A.: TWIST: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In: REALMAN (2006)
6. Jiménez, A., Martínez-de Dios, J.R., Sánchez-Matamoros, J.M., Ollero, A.: Towards an open testbed for the cooperation of robots and wireless sensor networks. In: Conference on Mobile Robots and Competitions (2010)
7. Srinivasan, K., Kazandjieva, M.A., Jain, M., Kim, E., Levis, P.: Demo abstract: Swat: enabling wireless network measurements. In: SenSys (2008)
8. Chini, M., Ceriotti, M., Marfievici, R., Murphy, A.L.: Demo: TRIDENT, Untethered Observation of Physical Communication Made to Share. In: SenSys (2011)
9. Baccour, N., Koubaa, A., Jamâa, M.: RadiaLE: A framework for designing and assessing link quality estimators in wireless sensor networks. In: Ad Hoc Net. (2011)
10. Matlab, `http://www.mathworks.com/`
11. Octave, `www.gnu.org/software/octave/`
12. Scidavis, `http://scidavis.sourceforge.net/`
13. Labplot, `http://labplot.sourceforge.net/`
14. Turon, M.: MOTE-VIEW: a sensor network monitoring and management tool. In: EmNetS-II (2005)
15. Buschmann, C., Pfisterer, D.: SpyGlass: a wireless sensor network visualizer. Sigbed (2005)
16. Dinh-Duc, A.V., Dang-Ha, T.H., Lam, N.A.: Nviz - a general purpose visualization tool for Wireless Sensor Networks. In: ECTI-CON (2012)
17. Ma, L., Wang, L., Shu, L., Zhao, J., Li, S., Yuan, Z., Ding, N.: NetViewer: A Universal Visualization Tool for Wireless Sensor Networks. In: GLOBECOM (2010)
18. Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., Levis, P.: Collection tree protocol. In: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, SenSys 2009, pp. 1–14. ACM, New York (2009)
19. Ramanathan, N., Chang, K., Kapur, R., Girod, L., Kohler, E., Estrin, D.: Sympathy for the sensor network debugger. In: Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, SenSys 2005, pp. 255–267. ACM (2005)

20. Dini, G., Savino, I.M.: LARK: A Lightweight Authenticated ReKeying Scheme for Clustered Wireless Sensor Networks. ACM Trans. Embedded Comput. Syst., 41 (2011)
21. Dini, G., Tiloca, M.: HISS: A HIghly Scalable Scheme for Group Rekeying. Comput. J. 56(4), 508–525 (2013)
22. Daidone, R., Dini, G., Tiloca, M.: STaR: Security Transparency and Reconfigurability for Wireless Sensor Networks programming. In: Proceedings of the 2nd International Conference on Sensor Networks, SENSORNETS (2013)