Janus: An Architecture for Flexible Access to Sensor Networks

Adam Dunkels*
Swedish Institute of Computer Science
adam@sics.se

Richard Gold, Sergio Angel Marti, Arnold Pears, Mats Uddenfeldt
Department of Information Technology, Uppsala University
rmg@it.uu.se, toch@ono.com, arnoldp@it.uu.se, mats@uddenfeldt.se

ABSTRACT

We present the design and implementation of the Janus[1] architecture for providing flexible and lightweight access to sensor network resources from Internet-type networks. Janus provides flexibility by focusing on *functions* of the sensor network rather the *data* that it contains. This allows us to perform service composition by dynamically combining functions together. In contrast to existing application-specific access techniques, Janus separates the access technique from the sensor network itself by inserting itself between the sensor network and the access network. This attribute allows for application-specific access techniques to be deployed dynamically without requiring the sensor network itself to be upgraded. Although Janus provides RPC-like semantics, unlike typical RPC systems, most of the functionality of Janus is present at the RPC client, which is located outside of the sensor network, in order to keep the RPC server inside the sensor network as lightweight as possible. We have implemented a prototype of Janus to interconnect a network of sixteen sensors in a typical environmental monitoring scenario to an Internet host.

Categories and Subject Descriptors: C.2.1 [Computer System Organization]: Network Architecture and Design

General Terms: Design

Keywords: Architecture, Sensor Networks

1. INTRODUCTION

Many wireless sensor network applications do not operate in isolation: there is a need to monitor and configure the sensor network, as well as to collect data sensed by the network. In some cases sensor networks are deployed in harsh or remote locations, where it is hard to physically access the sensors [10]. Being able to remotely access the sensor network not only reduces the complexity of the deployment, but can also be crucial to the long term viability of the network. One example of a sensor network deployment requiring remote access is the Great Duck Island habitat monitoring network [9]. In the words of Mainwaring et al.[9]:

> "Although personnel may be on site for a few months each summer, the goal is zero on-site presence for maintenance and administration during the field season, except for installation and removal of nodes."

In this paper we present Janus, a flexible architecture for remote access to sensor networks. Janus is designed both to support existing sensor network paradigms, and to be a flexible platform for future development both in terms of new functionality and new combinations of existing functionality.

Existing approaches to remote access to sensor networks are typically application-specific [2, 7–9]. Whilst these approaches each perform well in their own right, they necessarily limit the range of interactions permissible with a sensor network. This is because these access applications have to be embedded in the sensor network itself, typically at the gateway node of the sensor network. The goal of Janus is to remove these restrictions with the following techniques:

1. Provision of a flexible signaling mechanism which supports both passive and active access approaches

2. A generic RPC-like mechanism which matches the data-centric nature of sensor nets by providing an interface to invoke functions rather than forwarding packets to destination nodes. We chose not to follow existing RPC approaches which exchange abstract data representations between a client and a server. Instead our approach uses named function invocation to remove the need for API synchronization between Janus and the sensor network.

3. On-demand setup of access to sensor network resources of varying types. Unlike current RPC approaches which require synchronization of an API before usage, Janus allows for dynamic negotiation of an API for accessing the sensor network.

4. A lightweight approach where primitive functions on a gateway to the sensor network can be combined at an external host to provide service composition. This is in contrast to existing RPC approaches where the server contains comparatively more functionality than the client. We believe that it is critical for as little functionality as possible to be present in the gateway. This is because the gateway will usually be in a

*Authors names appear in alphabetical order of surname
[1]Janus is the two-faced Roman god of doorways

DIN'05, September 2, 2005, Cologne, Germany.

location which is hard to access physically and additionally since the gateway may be a sensor network node itself.

We have implemented a prototype version of Janus. To demonstrate the feasibility of the approach we used Janus to connect a typical sensor network to Internet hosts on a LAN. Our sensor network implements both *active querying* and *passive event notification*. A client application can request data from specific nodes, such as the sound sensor history of a particular node. Motion detectors on the sensors have been programmed to propagate an alert message to the gateway when the detected motion exceeds a given threshold. These basic functions are then combined to implement reactive data operations for regions of the network. We can also ask the network for active notification if there is a high level of activity in a particular area.

2. ARCHITECTURE

Janus comprises two entities: an *engine*, which runs on the sink node in the sensor network, and an *agent* that communicates with the engine. The engine and the agent communicate using the *eXtensible Resolution Protocol* (XRP) [6], as shown in figure 1.

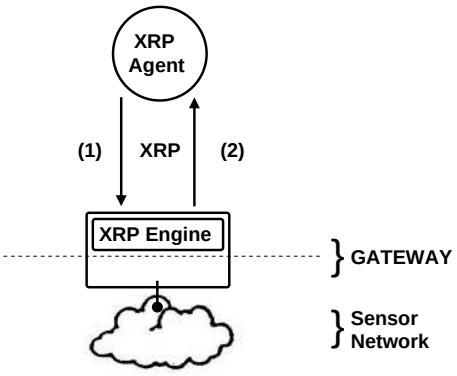

Figure 1: Architecture overview: (1) XRP requests sent by the agent (2) XRP replies sent by the engine.

Gateway : Sits between the sensor network and the access network and runs the XRP engine.

XRP Engine : Provides an RPC-style interface to sensor network functionality which the XRP agent uses to discover network resources and operations provided in the sensor network.

XRP Agent : Exchanges XRP messages with the XRP engine to query the sensor network and to receive events

Janus uses XRP as a basis upon which to construct an RPC-style interface to the sensor network. A more detailed discussion of how XRP is used is provided in section 3.5. Participating in each Janus XRP transaction are an XRP Agent and an XRP Engine. In the remainder of the paper we refer to the XRP Agent as the *agent* and the XRP Engine as the *engine*.

The agent uses a typical RPC client semantics and sends requests for function invocation to the engine. The engine executes tasks on behalf of the agent and returns the result to the agent from which the request originated. An additional mode of operation is when the agent registers at the engine interest for a particular event from the sensor network. When this event occurs, the engine then sends an XRP message to the agent informing it of this event.

XRP permits us to specify a extensible number of parameter fields for each packet. This makes XRP expressive enough to meet our needs. The XRP packet begins with an XRP command, which in our case implementation is either a data request (DREQ) or a data reply (DREP). The XRP command parameters are of variable length and their semantics is given by a *class* field, which names the parameter and a *class-type* which defines the parameter's data type. XRP parameters can appear anywhere inside the parameter block, thus are not bound to a specific position. We take advantage of this when we export the API of the gateway. See [6] for a more detailed description of the XRP packet format and its possible uses.

The agent and the engine exchange XRP messages with each other in order to:

1. Dynamically discover available sensor network resources. This enables the agent to ascertain which resources are available to it via the engine. This is useful in situations where the agent does not have explicit knowledge of the exact nature of the sensor network resources.

2. Send queries from the agent to the gateway concerning the state of sensor network. Examples of these involve querying the current temperature, light levels and motion sensing.

3. Send information from the gateway to the agent about the state of the sensor network. An agent can be notified by the sensor network when a specific event occurs. Examples of these include when motion is detected or when there is a significant change in temperature.

The gateway uses XRP to export the sensor network resources as functions which the agent can access. The agent can then invoke these functions by sending XRP messages to the gateway. The functions are named by *selectors*, which are opaque and uniquely map to functions at the gateway.

2.1 Supporting Multiple Sensor Network Types

Real-world sensor network deployments typically provided one of the following interfaces to either access collected data or to query the sensor network: directed data diffusion, database abstractions or passive monitoring. We discuss how each interface can be implemented with our architecture:

Directed diffusion [7] is a data-centric routing protocol for sensor networks. The protocol is based on a model in which a sink node first registers a *data interest* with the network. When the interest has been propagated through the network, the sensor network begins sending data in the reverse direction of the interest propagation. Directed diffusion fits well with the Janus architecture. The first DREQ message sent from the agent to the gateway causes a data interest to be registered with the network. Data from the sensor network will then be reported back from the gateway to the proxy with an DREP message.

TinyDB [8] is perhaps the most well-known example of a sensor network database abstraction. TinyDB makes the sensor network appear as a database that can be queried for data. Data queries are made using an SQL-like syntax, and data is transported through the sensor network back to a gateway node. Data aggregation is used to reduce the amount of communication in the network. The database abstraction model also works well with Janus. When the XRP gateway receives a DREQ message from the agent, a TinyDB query to be inserted into the sensor network. The results of the TinyDB query is then placed in an DREP message and sent back to the agent.

Many deployed wireless sensor networks have implemented a passive monitoring network (e.g., [2,9]), where data is transported

through a sensor network to a base station. The base station either logs the sensed data for later processing, or transmits the data to an external entity. Our architecture can be seen as a generalization of the gateway approach from such deployments.

2.2 Supporting Multiple Access Applications

In addition to supporting multiple types of sensor networks, Janus also provides support for multiple access applications. This scenario is depicted in figure 2.

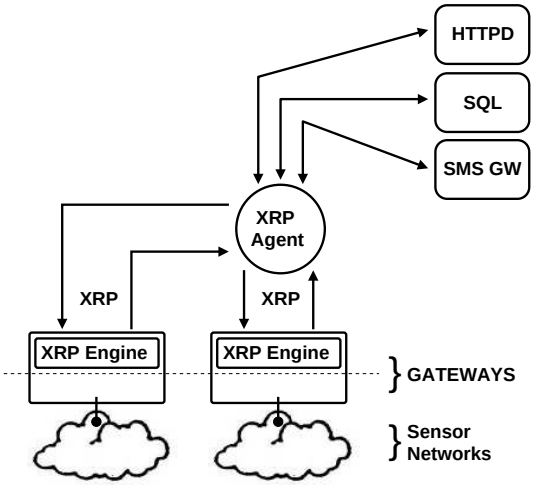

Figure 2: Extended architecture with access servers connected to multiple sensor networks via an XRP agent.

The goal of using application-specific access techniques is to enable access services such as HTTP, SQL or an SMS gateway to process the information gathered by the sensor network. As this information is quite simple due to the constrained nature of the sensor network itself we do not envisage this translation to be an impossible task. The advantage of this approach is to enable a whole class of deployed applications to interface with the sensor network. This would dramatically increase the ease-of-use of interpreting the sensor network data. In our scenario described in section 1, if the beach, the field and the forest of the island are different sensor networks monitoring different aspects of the environment we may wish to access the sensor network resources in different ways. We may care about specific events happening on the beach and therefore wish to be notified when a specific event occurs. One such notification mechanism could be via an SMS gateway. Subsequently we may care about the average of rainfall in the forest and wish to have this information presented to us on a regular basis either via a web page or by updating a database. As depicted in figure 2, these differing sensor networks can be connected through various gateways to one agent. This agent can then provide application-specific proxies for access applications to connect to. If we take this approach, we can keep the application-specific code on an entity *outside* of the sensor network. This makes it much easier to upgrade or add new functionality to support other access applications than if this application-specific code is located in or near the sensor network.

3. IMPLEMENTATION

We have implemented a prototype which comprises of a subsection of the architecture described in the previous section. Our prototype was written in C++ under Linux and we connected it to a sample sensor network using a USB connector attached to a sink node. We have successfully run tests with the agent and the gateway + engine on separate machines and with a sensor network of up to 16 sensor nodes.

3.1 Sensor Network

Our sensor network uses a standard equipment sensor board developed at Freie Universität, Berlin. These devices have an MSP 430 microcontroller, 2KB of RAM, 8KB of EEPROM, and a wide variety of sensors. The sensors can detect motion, light, temperature, vibration, and sound.

The sensor nodes run Contiki, a small operating system for tiny devices that allows us to implement the desired functionality. Our code has been written in C.

As an example of a real sensor network, we have implemented a scenario where nodes store sensed sound data, aggregate it and send it to a sink node in reply to a query. For our purpose, we have implemented the following functions in the sensor network:

Query current : When the sink node asks a specific node about the current sound level in one of its sensors, the node checks the current sensor state and sends back the result.

Query log : Nodes consult their sound sensor periodically, compute a mean of the sound data every 5 seconds and store the result in their memory to form a set of measurements over the last minute. When a node receives a *query_log* message from the sink, it will reply with the information that it has stored.

Event notification : Critical and/or interesting events in the network are reported to the sink for notification outside the network.

3.2 Gateway

The gateway sits between the sensor network and the access network and contains an engine. It handles incoming connections from agents and listens to incoming events from the sensor network. When an agent connects to the gateway it can issue XRP commands, which are interpreted by the engine at the gateway. All knowledge about the sensor network is situated at the gateway and all functions used to query the sensor network are implemented here. An API at the gateway can be exported as a list of the available functions and their corresponding selectors.

3.3 XRP Engine

The engine is in charge of the agent interaction on the gateway. This interaction takes the form of XRP messages sent back and forth as seen in figure 3. When an agent has connected it begins to issue XRP commands. These commands are interpreted by the engine. The agent can discover the available sensor network resources on this gateway by sending a DREQ for the API. The engine processes this request and composes an XRP packet with the API. This XRP packet is then sent back to the agent as a DREP message. A function is invoked on the gateway when the engine processes a QUERY style DREQ for a selector bound to a local function. The local function communicates with the sensor network and returns a result. The DREP is then built with the returned result as the payload.

3.4 XRP Agent

The role of the agent is to exchange XRP messages with an engine to query the resources of the sensor network and to receive

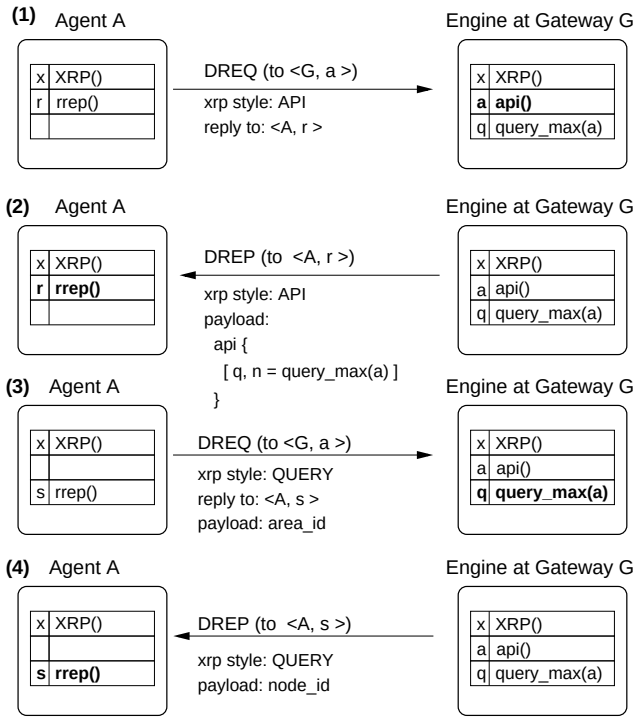

Figure 3: Signaling: The left column in the table represents function addresses and the right table column represents function names. (1,2) Exporting the API of the sensor network (3,4) Issuing an RPC by addressing a selector bound to a function together with the packet payload as an argument.

events generated inside the sensor network. The agent needs to request the API of the gateway to be able to actively query the sensor network. After the agent has received the API it can proceed to invoke functions at the gateway in a way similar to RPC. In an extended architecture the agent could be invoked by separate access servers to provide user interaction and present data. In our implementation we attach a simple text-based client to the agent.

3.5 Signaling between Agent and Engine

All XRP messages in our implementation are transmitted via UDP between the agent and the gateway. All the functions to interact with the sensor network are implemented at the gateway. We provide both low-level functions to query individual nodes and high-level functions to allow for more advanced queries, like the mean value inside a given area.

The gateway listens to events that originate inside the sensor network and waits on incoming connections to reach its engine. Once an agent connects, it negotiates with the gateway to retrieve an API of functions to reach the sensor network. After the API has been exported to the agent, it can issue requests by invoking functions at the gateway.

The signaling is achieved using a combination of XRP DREQ and DREP messages as shown in figure 3. The figure shows how XRP packets are sent between the agent and the engine at the gateway. A selector to handle the reply is always installed prior to sending a request and we will refer to this selector as the *reply selector*. Each DREQ contains the reply selector along with the address of the sending node. This information is used by the gateway to target its reply.

When an agent wishes to retrieve the API from a gateway, it installs a reply selector *r* and issue **(1)** a DREQ for the API to the gateway. The engine running at the gateway processes the request and responds **(2)** with a DREP to the *r* selector on the agent. This reply contains the API represented as XRP parameters inside the packet. In figure 3 we show the *query_max()* function along with its selector. This function takes an area as the argument and returns the node with the maximum sensor value. After the API has been exported, the agent can invoke functions at the gateway by addressing selectors on the gateway with a QUERY style message. The agent installs a reply selector *s* to issue **(3)** a DREQ with the function argument as the payload, in this case the requested area inside the network. The engine running at the gateway processes this request by invoking the corresponding local function. The gateway issues individual requests for the sensors within the given area to find the node with the highest sensor value. The engine proceeds by building a DREP containing the returned value as an XRP parameter and sends **(4)** this to the reply selector at the agent.

4. DISCUSSION

We now discuss some of the issues that arose whilst implementing Janus. We discuss some of the features that XRP has, show how Janus can support existing sensor network approaches and present some ideas for future work.

4.1 XRP Properties

The XRP protocol has the following properties which are crucial for providing flexible access to sensor networks: **Expressiveness** : XRP messages need to be interpreted in order for the correct functionality to be executed. By using the XRP protocol, a query is made for some information or data from the sensor network. This XRP query is then processed by the engine in order to return the appropriate result. This process of interpreting the XRP query is how XRP provides expressiveness. Rather than using a self-describing packet format, XRP performs signaling to set up functions on the engine that perform the tasks that the agent has requested. The reason for this design choice is that it is easier for a signaling protocol to express a new piece of functionality through its instruction set than a packet header format to do likewise. **Modularity**: By inserting a flexible architecture between the two different networks, it is possible for both networks to fundamentally change as long as they adhere to the API that Janus provides. Alternative access applications may be deployed on-demand to meet new system requirements. For example a sensor network which provides, via a gateway, an SMTP access method may wish to additionally deploy an SMS gateway to provide event notification. Different types of sensor networks are useful for different scenarios and it is our intention to not restrict how the sensor network is implemented, but rather to use Janus to allow them to be used in a consistent manner.

The usefulness of these properties can be explored through the discussion of *invariants* [1] in network architectures. Invariants are pieces of an architecture that cannot be changed without stopping an architecture from functioning correctly. The authors of [1] claim that all network architectures contain invariants, a claim that we agree with. Examples are the IPv4 address in the Internet and the SIM card in mobile phones. Since the access application and sensor network in our scenario are potentially quite disparate and variable, we believe that we should not chose either one to be the invariant for our system. We propose that Janus which is inserted between the access applications and the sensor network should be the explicit invariant of our system. We note here that this does not mean that the functionality remains static, but rather that the XRP protocol format is adhered to.

4.2 Future Work

A promising direction for future work is to completely integrate Janus with the Contiki Operating System to provide an active networking platform for sensor networks. Our goal is to use the Janus function table to store programs which have been shipped in XRP packets. The function table presented in figure 3 shows pre-loaded programs being accessed by XRP queries. We are currently implementing an XRP interpreter and a function table for the sensor nodes used for the implementation presented in this paper. We will then build the functionality to allow programs to be sent inside XRP packets and subsequently loaded dynamically into the Janus function table running inside a sensor network node. Through our work with microLUNAR [11] we know that it is feasible for an XRP interpreter and a function table to be implemented in less than 3.5KB.

5. RELATED WORK

Existing solutions for querying networks such as SNMP provide a standard interface for accessing the resources of a network. However the expressiveness of SNMP is a concern. It provides a get/set operation for a Management Information Base (MIB). This provides the ability for both event-driven and on-demand types of query/responses from the sensor network. However, it is not straightforward to send specialized queries into the sensor network from outside of the sensor network. With SNMP, we are limited by what information and events that are defined in our MIB at the time of deployment. Furthermore, we cannot see a clear migration path from SNMP to an active network platform as discussed in section 4.2. SNMP could potentially be useful for querying a sensor network if we could make the assumption that the the gateway node to the sensor network will never be a sensor network node itself. Since SNMP uses ASN.1 for representation, it is not suitable for implementation in severely memory-constrained devices. From our work on LUNAR for embedded systems [11] we know that it is feasible to implement an XRP parser in an embedded environment.

Similar concerns about implementation in constrained environments apply to existing RPC-style systems such as SUNRPC, Java RMI, CORBA and WSDL. The representation systems used such as XDR or XML were not designed for devices which are extremely memory-constrained such as sensor network nodes. For the Janus approach where we wish in the future to support the gateway to the sensor network being a sensor network node and also to migrate the Janus function table into the sensor network itself, we must be mindful of the resource consumption of the tools that we choose.

Delay Tolerant Networking (DTN) [5] is a network architecture that is designed for environments with intermittent connectivity, scheduled transmissions, and possibly long propagation delays. DTN provides a *convergence layer* abstraction that allows running DTN both on top of TCP/IP and a sensor network network protocol. DTN can be used for connecting sensor networks and the Internet [4] by inserting a DTN gateway at the interconnection point, but can also be used within the sensor network.

Buonadonna et al. [3] use a special purpose gateway called the Sensor Network Appliance (SNA) running Linux. The SNA has a radio module that can communicate with the wireless sensors, an Ethernet connection, as well as support for satellite or cellular connectivity. The SNA runs an ODBC-compliant database management system and a HTTP server that both can be used for remote access of the sensor network. This is different from our approach, in that the Janus places the user interface—the HTTP server and database management system—at a proxy rather than on the gateway. This makes modifications to the interface easier after deployment of the sensor network. Janus also enables new services to be added after deployment simply by setting up additional agents and/or access applications.

6. CONCLUSIONS

This paper presents Janus, an approach to providing flexible access to sensor networks. Janus allows clients to access sensor network resources without requiring detailed knowledge of the sensor network implementation structure. The approach is lightweight, flexible and extensible. We have demonstrated its ability to support Internet client interaction with a prototype sensor network implementation for both simple and complex operations on the target sensor network. We have found our initial implementation to be promising and plan to integrate several different types of sensor networks as well as several access mechanisms in the near future.

Acknowledgment

This work was partly sponsored by SSF, the Swedish Strategic Research Foundation, VINNOVA, the Swedish Agency for Innovation Systems, and the European Commission under contract IST-004536-RUNES.

7. REFERENCES

[1] AHLGREN, B., BRUNNER, M., EGGERT, L., HANCOCK, R., AND SCHMID, S. Invariants: A new design methodology for network architectures. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA)* (2004).

[2] BECKWITH, R., TEIBEL, D., AND BOWEN, P. Report from the field: Results from an agricultural wireless sensor network. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors* (Tampa, Florida, USA, Nov. 2004).

[3] BUONADONNA, P., GAY, D., HONG, J. H. W., AND MADDEN, S. TASK: Sensor Network in a Box. In *Proceedings of the Second European Workshop on Sensor Networks* (2005).

[4] DUNKELS, A., VOIGT, T., ALONSO, J., RITTER, H., AND SCHILLER, J. Connecting Wireless Sensornets with TCP/IP Networks. In *Proceedings of the Second International Conference on Wired/Wireless Internet Communications (WWIC2004)* (Frankfurt (Oder), Germany, Feb. 2004).

[5] FALL, K. A delay-tolerant network architecture for challenged internets. In *Proceedings of the SIGCOMM'2003 conference* (2003).

[6] GOLD, R., GUNNINGBERG, P., AND TSCHUDIN, C. A virtualized link layer with support for indirection. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA)* (2004). http://user.it.uu.se/~rmg/pub/fdna05-gold.pdf .

[7] INTANAGONWIWAT, C., GOVINDAN, R., AND ESTRIN, D. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobile Computing and Networking* (2000), pp. 56–67.

[8] MADDEN, S., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. The design of an acquisitional query processor for sensor networks. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 2003), ACM Press, pp. 491–502.

[9] MAINWARING, A., POLASTRE, J., SZEWCZYK, R., CULLER, D., AND ANDERSON, J. Wireless sensor networks for habitat monitoring. In *First ACM Workshop on Wireless Sensor Networks and Applications (WSNA 2002)* (Atlanta, GA, USA, Sept. 2002).

[10] PADHY, P., MARTINEZ, K., RIDDOCH, A., ONG, H., AND HART, J. Glacial environment monitoring using sensor networks. In *Proceedings of the REALWSN'05 Workshop on Real-World Wireless Sensor Networks* (Stockholm, Sweden, June 2005).

[11] TSCHUDIN, C., GOLD, R., RENSFELT, O., AND WIBLING, O. LUNAR: a Lightweight Underlay Network Ad-hoc Routing Protocol and Implementation. In *Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN'04)* (2004).