# Design of a Decoupled Sensor Network Architecture Based on Information Exchanges

Eli De Poorter    Ingrid Moerman    Piet Demeester
Ghent University - IBBT, Department of Information Technology (INTEC)
Gaston Crommenlaan 8, Bus 201, 9050 Ghent, Belgium
Email: eli.depoorter@intec.ugent.be

## Abstract

***Sensor networks are used for simple monitoring applications, but also for complex applications, such as wireless building automation or medical assistance. Current layered architectures do not support the dynamic and heterogeneous nature of these networks. Therefore, we present an alternative architecture that decouples protocol logic and packet representation. Using this system, multiple information exchanges are automatically combined in a single packet. In addition, the system dynamically selects the most optimal network protocols and supports system-wide quality-of-service. Thus, our architecture is better suited for next-generation applications. We illustrate our architecture with several code examples, and prove that our architecture is much more scalable, in terms of memory requirements, energy requirements and processing overhead, than tradition system architectures.***

**Keywords: wireless sensor networks, sensornet architecture; system architecture; QoS; energy efficiency; heterogeneity; protocol selection**

## 1 Introduction

Sensor nodes are small and cheap devices that can monitor their environment. They are equipped with a simple radio to communicate with other sensor nodes. Due to their low cost, many sensor nodes can be distributed over an area. Over the last years, many wireless sensor networks (WSNs) have been deployed to monitor nature and office environments [2].

Due to these successes, other application domains have expressed a strong interest in using WSNs for more complex tasks. More sophisticated applications, such as process monitoring and control, wireless building automation, medical monitoring, disaster intervention or asset tracking, also benefit greatly from the use of many cheap sensor devices. However, the number of successful deployments of these applications is far less [3].

These next-generation applications impose many network requirements which are not found in traditional WSNs.

- To provide sufficient end-user support, a WSN must be easy to update and maintain. *Run-time addition* of new services and network protocols should be supported.
- WSNs will become *heterogeneous* [4], containing both simple nodes (such as light switches) and more complex nodes (such as heating controllers).
- Additionally, *QoS requirements* can no longer be ignored [5]. Medical, security and surveillance applications require that each application has its own set of specific QoS requirements.
- Since future applications will require even smaller nodes, new ways have to be found to ensure that network protocols have a very *small memory footprint*.
- Since most sensor nodes are battery powered, *energy efficiency* remains very important.

At present, there is no architecture that supports all of these challenges. As stated by Culler et. al: "*the primary factor currently limiting progress in sensornets is not any specific technical challenge but is instead the lack of an overall sensor network architecture*" [6].

Therefore, in this paper, we present an information driven architecture ('IDRA') for wireless sensor networks. This framework is specifically designed to support next-generation WSN applications. It takes into account the heterogeneity of the sensor nodes and supports energy-efficiency and QoS at an architectural level. What's more,

This article extends the paper '*An Information Driven Sensornet Architecture*' [1]. It adds illustrative code examples, it elaborates on several design choices and it provides additional evaluation results.

the proposed architecture can be made fully compatible with existing legacy sensor networks.

The remainder of this paper is organized as follows. In Section 2, we discuss the design philosophy behind our information driven system architecture. Section 3 clarifies how such an information driven architecture can be implemented. Next, in Section 4, we demonstrate how this architecture can be used to support energy efficiency, QoS, heterogeneity and legacy networks. In Section 5, we discusses possible shortcomings of the architecture. We describe our implementation experiences in Section 6 and evaluate the performance of our system in Section 7. Next, in Section 8, we compare the architecture with existing sensornet architectures. Finally, Section 9 concludes this paper.

## 2 What is an information driven architecture?

The main responsibility of a network protocol is to ensure that information is relayed to the correct destination. However, in practice, a network protocol has many responsibilities that are not directly related with its main function. Each protocol layer must *(i)* define a message format (including header and trailer fields), *(ii)* provide buffers to temporarily store packets and *(iii)* gather information from other nodes.

We argue that this approach is very inefficient. It makes no sense that every individual protocol layer has to bear the burden of gathering information, providing buffers and implementing header manipulations. Common sense indicates that such functions, which are implemented multiple times in the system, should be implemented in a shared library. Furthermore, the same information is often gathered by multiple protocols, each time resulting in an additional information exchange.

Therefore, we propose a system in which protocol design is based on ‘information exchanges’. The role of a network protocol is simplified to its 2 main tasks: sending information and interacting with the relayed information. Packet creation and buffer provisioning are delegated to the architecture. This way, network protocols are simpler and require less memory.

## 3 Design of an information driven architecture

In this section, we describe how such an information driven architecture can be implemented. A conceptual representation is given in Figure 1.

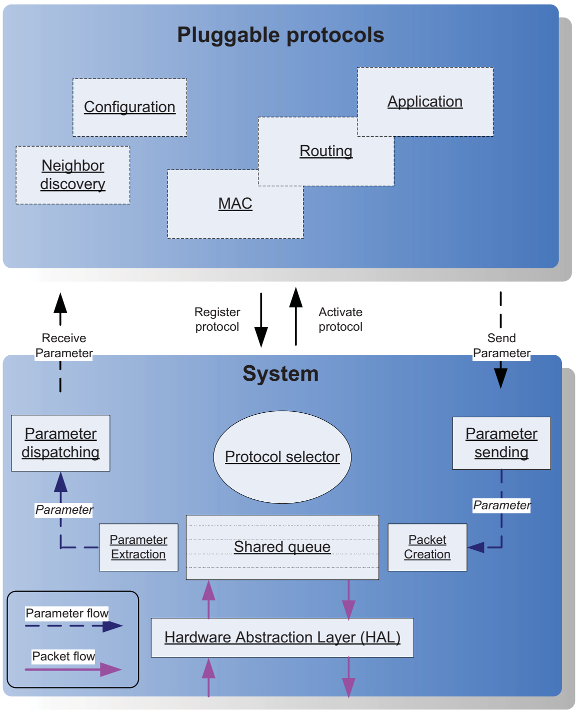

**Figure 1. In an information driven architecture, protocols exchange information with the system, and rely on the system to create and send packets.**

### 3.1 Information exchanges

Network protocols often exchange information with a remote node. Typical examples of exchanged information are:

- measured data values, such as the local temperature;
- status updates, such as the remaining battery capacity;
- or control information such as a route-request.

Using our information driven approach, network protocols do not create a new packet to send these types of information to a remote node. Instead, they rely on the system to send and receive information.

To send information to a remote node, the protocol hands over an information parameter to the system, together with the required destination (Fig. 1, ‘Parameter sending’). The system will create a new packet and ensures that the parameter is encapsulated into this packet.

To receive information from other nodes, network protocols indicate to the system what type of parameters they are interested in. Whenever a packet arrives at its final destination, the system decapsulates the information parameters and distributes them to the interested protocols (Fig. 1, ‘Parameter dispatching’).

The main advantages of transferring the creation of packets to the system are: *(i)* the system can ensure that redundant control information is sent only once; *(ii)* protocols are simpler since they do not need to implement memory operations for manipulating header fields; and *(iii)* by combining multiple information parameters into a single packet, the number of required packets decreases drastically (Section 4.1).

## 3.2 Shared queue

Traditional networks use a 'store-and-process' approach, where each network protocol stores the packets in an internal queue before processing. Each protocol layer requires these queues because packets may be passed up or down faster than they can be processed. Thus, the total amount of buffer memory increases linearly with the number of protocol layers.

In contrast, in the information driven architecture, incoming packets are stored in a system-wide queue (Fig. 1, 'Shared queue'). The system selects which packets are ready for processing by a network protocol so that one-after-another the protocols can process the packets.

The use of a shared, system-managed queue has several advantages: *(i)* protocols are simpler and smaller since they do not have to allocate queue memory; *(ii)* packets do not need to be copied between protocols, resulting in less processing overhead; *(iii)* since the queue occupation from all protocols is averaged, less total queue memory is required; and *(iv)* monitoring and managing the total number of packets in the system is simpler.

## 3.3 Packet façade

Traditional network protocols encapsulate packets in header fields. These header fields contain information which can only be read by the protocol that created the packet header. In our information driven architecture, a packet façade (Fig. 2) is responsible for packet creation.

The system uses the packet façade to create new, standardized packets. Information parameters are encapsulated in the payload of the created packet and stored in the shared queue.

Network protocols use the packet façade to interact with relayed packets. Protocols add or retrieve packet attributes, such as 'source', 'destination', 'QoS ID' or 'time-to-live'. These packet attributes fulfill the same role as traditional header fields, but are more dynamic: they can be omitted or added freely without redefining the packet structure. *Moreover, packet attributes have a system-wide significance: they can be inspected by the system or any other protocol.*

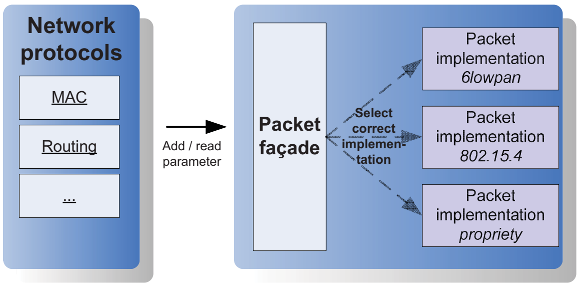
Network protocols
MAC
Routing
...
Add / read parameter
Packet façade
Select correct implementation
Packet implementation 6lowpan
Packet implementation 802.15.4
Packet implementation propriety

**Figure 2. Through a packet façade, protocols interact with packets. Protocols do not require any knowledge about the actual packet construction.**

The packet façade uses a separate 'packet implementation' module to convert the attributes of a packet into an actual radio packet. Thus, the system and the protocols do not need to worry about the actual storage of the control information associated with a packet. Developers can choose to use one of the existing 'packet implementation' modules, or provide their own (propriety) packet implementation. This way, the packet type can be changed without any changes to the system or the protocols: protocol logic and packet representation are decoupled.

Using a separate packet façade has the following advantages: *(i)* protocol development is simplified since there is no need to define headers; *(ii)* packet attributes have a system-wide significance and can be inspected by any protocol or architecture; *(iii)* since protocols are not tied to a specific packet implementation, the encompassing packet type can easily be changed or optimized (e.g.: 6lowpan, IEEE802.15.4 or a custom packet).

## 3.4 Pluggable protocols

The traditional OSI reference model [7] uses a non-flexible layered architecture: packets are sent to a predetermined protocol layer. In contrast, the information driven architecture does not statically wire packets to a specific protocol. The system decides at run-time which protocols should be selected to process incoming packets (Fig. 1, 'Protocol selector').

To be selected for packet processing, a protocol must register itself by adding *filters* to the system. These filters indicate for which packet types the protocol is optimized. Through the packet façade, the system checks if the characteristics of the arriving packets match any of the registered filters, and selects the appropriate network protocols to process the packet. When no filters match, a default routing or MAC protocol is chosen.

This approach has the advantages that multiple, specialized network protocols can be combined in the same system. For example, a routing protocol implements an efficient broadcast algorithm. It registers itself for all packets that are broadcasted over the network. Another routing protocol delivers high QoS guarantees. It can add a filter to indicate that it is optimized for routing voice packets.

'Plugging' in protocols, rather than statically wiring them, has several advantages. *(i)* Since many applications have diverse network requirements, the architecture is able to dynamically change between different routing or MAC protocols at run-time. The optimal protocol is selected by the system based on the network context or the packet type [8]. *(ii)* Run-time insertion of protocols is supported. *(iii)* Legacy systems can be supported (Section 4.4). More system-specific implementation details can be found in Section 6.

## 4 Exploiting the concept

Current system architectures are not designed to support energy-efficiency, QoS, heterogeneity or legacy networks. Traditionally, support for these advanced features is very complex and requires major changes to the underlying system architecture. The lack of architectural support for QoS and heterogeneity is a major obstacle that hampers the deployment of many next-generation applications for WSNs.

In the next sections, we demonstrate how these features can easily be implemented at an architectural level in our information driven system.

### 4.1 Energy efficiency

Since energy is scarce in sensor nodes, wireless sensor networks aim to transmit as few packets as possible. To this end, 'measured data' from different nodes can be combined into a single packet by data-aggregation protocols [9]. This data-centric approach has several limitations: *(i)* data aggregation is application dependent; *(ii)* aggregation protocols are often coupled to a specific routing protocol; *(iii)* data aggregation is limited to the measured '*data*', it does not include other exchanged control messages; *(iv)* data that originates from different applications can not be aggregated.

We claim that this approach should be broadened so that *all* types of information exchanges are aggregated. Many information exchanges between nodes are not very time-sensitive, such as status information, remaining energy information, or low-priority routing information. As such, it is reasonable to assume that some of these packets can be delayed for a short amount of time before being sent. When a protocol requests the sending of a parameter, the protocol should also give an indication of the time-sensitivity of the parameter.

The system collects the information parameters in the waiting space of a central repository (Figure 3). Delay-tolerant parameters can remain in the waiting space for up to a per-parameter predefined period of time. Whenever a packet is relayed through the node, all information parameters to the same 'next hop' or 'destination' address are added to the packet. If no data has been relayed within the allowed waiting time, the system generates a new packet which combines all parameters that are destined for the same node. Finally, when a packet reaches its final destination, the system will distribute the encapsulated information parameters to the interested protocols.

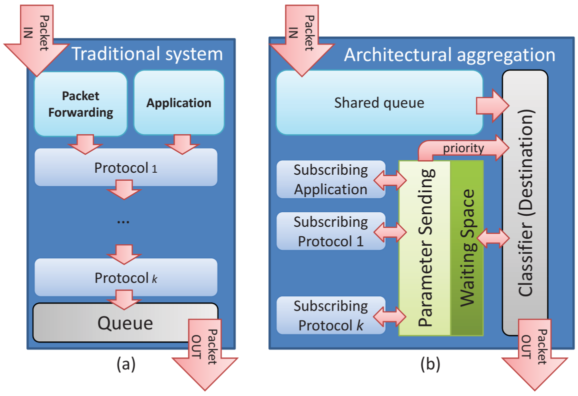

**Figure 3. Extending the data aggregation concept. (a) Traditional architecture. (b) Architecture with support for global aggregation.**

For a more in-depth analysis, we refer to [10], where it is shown that, ideally, the number of transmissions can be lowered by a factor, equal to the number of information parameters which can be combined in a single packet. In contrast with traditional aggregation schemes [9, 11], this approach is part of the architectural design. As a result, the network developer can combine this approach with any type of routing protocol, and information from all layers can be aggregated, rather than just application data.

### 4.2 Architectural QoS

QoS guarantees are required by many medical, security, critical monitoring and control applications. However, current QoS research focuses mainly on one of the network layers and solves only a few of the application requirements. As stated by Troubleyn et al. in [5], QoS should be supported in both the protocols and the architecture. Only then can system-wide QoS be guaranteed.

The information driven architecture is very suited to support architectural QoS. Through the packet façade, the system can read the QoS attributes of any relayed packet. Since all packets are stored in a shared packet queue, the system can monitor all available packets. This gives the system a clear view on the expected delay of each packet. QoS can be supported by giving precedence to packets with a higher QoS level, or by intelligently dropping non-priority packets. To fulfill QoS requirements, a QoS protocol can put the processing of low-priority packets on hold, even when those packets are currently being processed by a protocol. Finally, the architecture supports multiple protocols, so that high-priority packets which require strict QoS guarantees can be processed by specialized protocols.

### 4.3 Heterogeneity support

Applications such as process and asset monitoring, disaster intervention and wireless building automation require special devices ('actuators'), which can interact with the environment [2, 4]. Thus, future sensor networks will consists of nodes with strongly diverging capabilities.

In a layered architecture, every sensor node needs to support the same protocol stack. Since no protocols can be omitted or added, the layered approach has limited support for heterogeneity. When using an information driven architecture, packet attributes remain associated with a packet, whether or not the protocol that added them is executed. Thus, the system can choose to omit non-essential protocols from nodes with little capabilities (Figure 4). The system can also choose to execute different, more simple protocols on lightweight nodes. These protocols can add their own packet attributes or can reuse the packet attributes that were added by previous protocols. This flexibility ensures that our architecture is suitable for both high capacity and low capacity nodes.

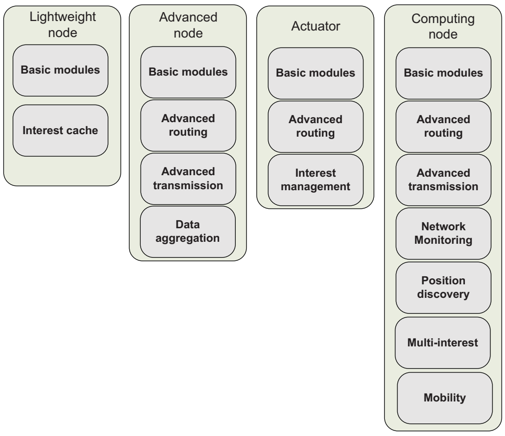

**Figure 4. Depending on their capabilities, the number of protocols can be varied**

### 4.4 Legacy support

One of the main problems new architectures face, is that they are not backwards compatible with existing infrastructure. Traditionally, the only solution is the installation of a translation node through which all communication passes, which results in a very inefficient network use.

The information driven architecture allows legacy nodes and IDRA nodes to communicate directly by supporting *(i)* legacy packet types, and *(ii)* a legacy MAC protocol.

*Legacy packets* can be supported transparently for the new protocols by providing the corresponding 'packet implementation'. This implementation should store the relevant control parameters at the expected header locations of the legacy packets.

The *legacy MAC protocol* can be ported to the new architecture. By registering it as the optimal MAC protocol for neighboring legacy nodes, the 'protocol selector' will always select the correct MAC protocol to send packets to the legacy nodes.

## 5 Disadvantages

In the next section, we discuss some possible disadvantages of using an information driven architecture.

### 5.1 Protocol-defined packets

In an information driven architecture, the system defines how packets are constructed. In addition, messages defined by the radio, such as ACKs, are supported through the transmission settings library.

However, some MAC protocols use custom-defined packet types (such as 'strobes') for their operation. Currently, the architecture does not allow network protocols to send self-defined packets. We feel that allowing this would take away many advantages of the packet façade approach. Asynchronous MAC protocols can still be implemented by sending the same packet multiple times.

If further research indicates that support for protocol-defined packets is absolutely required, a library will be added that allows a protocol to send and receive a self-defined packet. Of course, these self-defined packets can not profit from many of the advantages of the information driven approach (such as reuse of packet parameters, efficient combination of exchanged information, and the decoupling of protocol logic and packet representation).

### 5.2 Standardization

An information driven architecture strongly benefits from an approach that standardizes both the information parameters and the attributes that are associated with a packet.

An *information parameter* represents information that is exchanged between protocols or applications on different nodes. Information parameters are often of interest to several protocols (for example: the remaining energy of neighboring nodes). When an already existing, standardized information parameters is added to the waiting space of the system, the corresponding value is updated. The information parameter is sent only once, rather than once for every interested protocol.

A *packet attribute* represents packet-associated information that is required to route a packet to its destination. Typical examples are a 'next hop' or a 'time-to-live' attribute. Standardizing control parameters ensures that they can be read by all network protocols, and that the system is aware of the properties of each packet.

# 6 Implementation

The presented information driven architecture ('IDRA') has been implemented using the TinyOS [12] operating system. Run-time addition of protocols is currently not supported, since TinyOS does not support dynamic code updates.

## 6.1 Internal workflow

Figure 5 shows the internal workings of the architecture. The shared queue is the central component of the architecture. Each packet in the shared queue has an associated *packet status*. Depending on the status of a packet, the following actions can be taken.

*Pre-processing* is executed once for each arriving packet. During pre-processing, duplicate packets and packets with a different next hop address are dropped. Also, when the packet reaches its final destination, the encapsulated information parameters are extracted and distributed to the interested protocols.

*Processing* is executed next. The 'Protocol selector' module analyzes the packet and selects the most optimal network protocol based on the packet characteristics (see Section 6.2).

*Post-processing* Packets that have been processed by the network protocols are prepared for sending. Relevant packet attributes (such as the sender address) are updated. Also, if parameters to the same next hop address or destination address are available, they are aggregated to the packet.

*Ready For Sending* After post-processing, the packets remain in the shared queue until the MAC protocol orders the system to send the packet.

## 6.2 Protocol sequence

When using a traditional layered architecture, arriving packets must first be stripped of their headers before they can be processed. Therefore, packets go through the protocol stack twice: once from the bottom to the top to remove the packet headers, and a second time from the top to bottom to actually process the packets. In the IDRA system, network protocols can access any packet information through the packet façade. Thus, there is no need to execute each protocol layer twice.

There is no fixed call sequence for the protocols since multiple protocols can co-exist on the same node. Depending on the protocol filters (see Section 3.4), the most optimal network protocol is selected. The algorithm for protocol selection is very simple so that it can be implemented even on lightweight nodes.

1. First, all interested monitoring protocols are executed. A monitoring protocol is considered interested if at least one of its filters match the current packet.
2. Afterwards, the most optimal routing protocol is executed. This is the network protocol with most matching filters (or, in the case of a tie, the first protocol that registered itself).
3. Finally, the most optimal MAC protocol is executed. This is the network protocol with most matching filters (or, in the case of a tie, the first protocol that registered itself).

The network protocols are always executed in the same order. Once a protocol finishes processing a packet, it signals one of the following return values to the system:

SUCCESS The network protocol finished successfully; the next protocol can be executed.

FAIL The network protocol can not process the packet; the packet should be dropped from the shared queue.

EBUSY The network protocol is not yet ready to process the packet; the protocol will be called again at a later time.

## 6.3 System libraries

A vertical component is available to the architecture (Fig. 5, 'module interactions'). Through this component, the following system provided libraries are available:

*Information Exchange*: through this interface, protocols can distribute information parameters to other nodes and receive information parameters from other nodes. The system converts the information parameters into a packet.

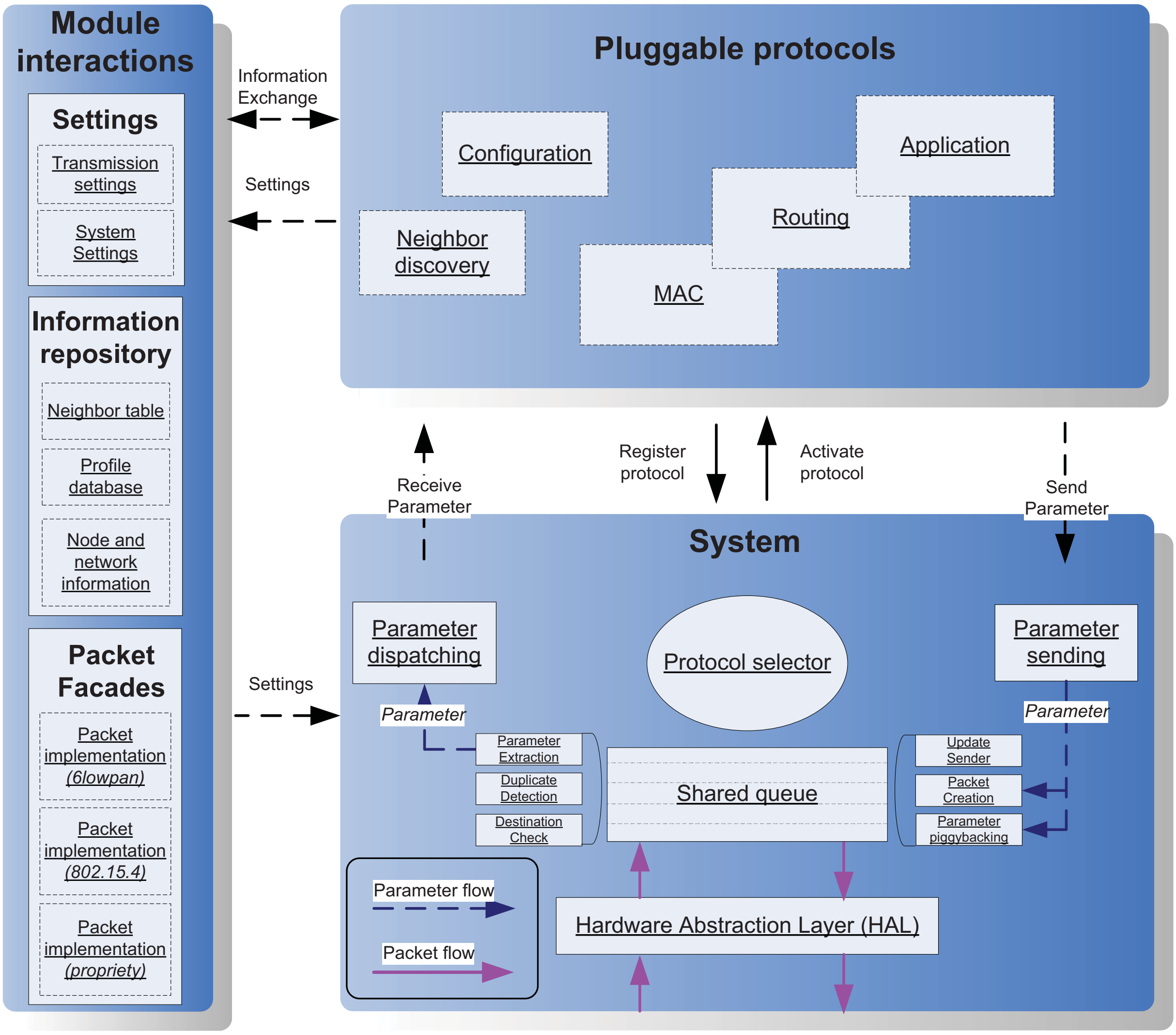

**Figure 5. The information driven architecture ('IDRA').**

*Packet Façade*: the packet façade is used to interact with system-created packets. Packet attributes can be added or read.

*Protocol Selection*: protocols can add filters to this component to indicate for which packet types the protocol is optimized.

*System Settings*: through this interface, modules change or read system settings, such as the node ID or the current battery voltage.

*Transmission Settings*: through this interface, protocols manage the sending of packets. It has provisions for *(i)* requesting how many packets from the shared queue are ready to be send, *(ii)* ordering the system to send a specific packet, and *(iii)* changing the radio settings.

## 6.4 Example code

In this section, we show how these libraries can be used to create a new network protocol. A simple routing protocol for broadcast packets is shown in Table 1. First, the network protocol registers itself as a routing protocol that is optimized for broadcast packets. Whenever the system selects this routing protocol for processing a packet, the network protocol uses the packet façade to update the packet attributes.

A second example is shown in Table 2 and demonstrates the implementation of a simple MAC protocol. Through the 'Transmission Settings' library , radio settings such as the transmission power and channel can be changed. Even though packets are send by the system, the MAC protocol

```
Module BroadcastRouting{

    // 1. Declaration of variables
    uint8_t moduleID=unique("protocol");
    uint8_t ttl=0;

    // 2. The protocol registers itself as a routing protocol suitable for broadcast packets
    command error_t Module.init(){
        call ProtocolRegistration.addFilter(moduleID, ROUTING_PROTOCOL,
                        DESTINATION, 2, EQUALS, BROADCAST_ADDR);
        return SUCCESS;
    }

    // 3. Executed when the protocol is selected for routing a broadcast packet
    command void Module.processPacket(uint8_t moduleId, void* packet){
        if(moduleId == moduleID){

            // 3.1. Get the time-to-live of the packet
            result=call PacketFacade.getPacketAttribute(packet, TIME_TO_LIVE, sizeof(ttl), (void*) &ttl);

            // 3.2. If the time-to-live equals zero: drop the received packet.
            if(ttl==0) signal Module.processPacketDone(moduleId, packet, FAIL);

            // 3.3. Otherwise, update the time-to-live and broadcast the packet again[a].
            else{
                ttl=ttl-1;
                call PacketFacade.setPacketAttribute(packet, TIME_TO_LIVE, sizeof(ttl), (void*) &ttl);
                signal Module.processPacketDone(moduleId, packet, SUCCESS);
            }
        }
    }
}
```

[a] A more advanced implementation should also check for duplicate packets to prevent broadcast storms.

**Table 1. Example code: a simple broadcast routing protocol.**

remains in control of sleeping schemes and accurate timing of the sending. The MAC protocol can request at any time which packets are ready for sending and order the system to send a specific packet.

## 7 Evaluation

In this section, we evaluate three key criteria of our architecture: the memory footprint, the energy efficiency and the processing overhead.

### 7.1 Memory footprint

The memory footprint of the different components of the architecture is shown in Table 3. The entry 'Other system components' refers to the implementation of preprocessing and postprocessing functions, QoS provisions and a shared neighbor table. The full architecture requires about 20kb

| Component | ROM | RAM |
|---|---|---|
| Information exchanges (Section 3.1) | 800 | 585 |
| Shared queue (Section 3.2) | 1330 | 2560 |
| Packet façade (Section 3.3) | 1402 | 6 |
| Protocol selection (Section 3.4) | 206 | 234 |
| Radio support (HAL) | 8892 | 322 |
| Other system components | 7606 | 1637 |
| Total | 20236 | 5344 |

**Table 3. Memory footprint (in bytes) of the different architectural components of the system.**

```
Module SimpleMAC{

    // 1. Get a unique protocol ID
    uint8_t moduleID=unique("protocol");

    // 2. The protocol registers itself as a general purpose MAC protocol
    command error_t Module.init(){
        call ProtocolRegistration.addFilter(moduleID, MAC_PROTOCOL,
                        0, 0, 0, 0); // default protocol
        call alarmClock.startPeriodic(200); // Every 200 msec, the protocol checks if packets are ready to be sent
        return SUCCESS;

    // 3. Regularly check if a packet is ready for sending
    event void alarmClock.fired(){
        if(call TransmissionSettings.checkNumberOfPackets() > 0) {
            call TransmissionSettings.sendPacket();
        }
    }

    // 4. Inform the system that the packet can be removed from the shared queue
    event void TransmissionSettings.sendPacketDone(uint32_t timestamp, error_t result){
        if(result==SUCCESS) call TransmissionSettings.removeLastSentPacket();
    }
}
```

**Table 2. Example code: a simple MAC protocol.**

| Protocol name | ROM | RAM |
|---|---|---|
| TOS2.1 MAC [12] | 11528 | 320 |
| SCP-MAC [13] | 21372 | 1056 |
| X-MAC [14] | 19854 | 876 |
| IDRA LPL MAC [15] | 822 | 176 |
| IDRA S-MAC [16] | 1126 | 184 |

**Table 4. Comparison of the memory requirements (in bytes) of TinyOS MAC protocols with IDRA MAC protocols.**

| Protocol name | ROM | RAM |
|---|---|---|
| CTP [17] | 7234 | 1198 |
| DYMO [18] | 11404 | 482(+60 per route) |
| Lunar [19] | 5000 | 1518 |
| IDRA CTP [17] | 712 | 130 |
| IDRA DYMO [20] | 5008 | 312(+18 per route) |

**Table 5. Comparison of the memory requirements (in bytes) of TinyOS routing protocols with IDRA routing protocols.**

ROM and 5 kB RAM memory, well under the memory limit of most sensor nodes.

To demonstrate the feasibility of the IDRA architecture, we implemented 2 MAC protocols (S-MAC [16] and LPL [15]) and 2 routing protocols (Collection Tree Protocol [17] and DYMO [20]).

When using a layered architecture, the memory consumption increases linearly with the number of network protocols. In contrast, using the IDRA architecture requires a significant initial investment in terms of memory, even without any network protocols. However, adding protocols to the system requires significantly less memory than when using a more traditional approach (Table 4 and 5). Due to the system provided libraries for most typical operations, the memory requirements of network protocols are reduced by a factor 2 to 10. Thus, the greater initial memory cost of our system is quickly offset.

### 7.2 Aggregation efficiency

An important feature of IDRA is the automatic combination and aggregation of information exchanges. In [10] an ILP formulation is given for calculating required number of packet transmissions for different protocols when using our aggregation method.

The result is illustrated in Figure 6, where the number of packet transmissions is shown for a variable number of protocols. All protocols regularly send information to a neighboring node. Information exchanges for protocol $i$ are sent every ($\Delta T_i$) time units. The first protocol has a protocol cycle ($\Delta T_1$) of five time units, each additional protocol has its $\Delta T$ increased by one.

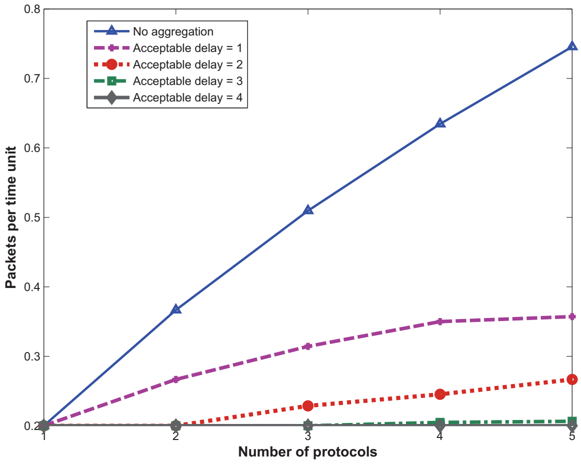

**Figure 6. Required number of packets per time unit when using global aggregation with multiple protocols ($\Delta T_i = i + 4$).**

Figure 6 shows that, when using global aggregation, adding new protocols does not significantly increase the required number of packets. When the acceptable delay increases, the number of packet required decreases. The minimal number of packets per time unit that needs to be sent is one packet every $\Delta T_1$ time units, with $\Delta T_1$ the lowest information interval. Thus, when the acceptable delays increases, the average number of packets per time unit decrease, provided that the packet size is big enough to contain the information from all the protocols. The maximum reduction of transmissions is equal to the number of control packet types. *To summarize, global-aggregation results in more profit when more parameters need to be exchanged and when these parameters have no strict deadlines.*

### 7.3 Processing overhead

A final performance criteria is the processing overhead of the system. System overhead is mainly caused by the following operations:

*Aggregation overhead* Searching for parameters to combine with routed packets results in additional processing delay. This causes problems when high-priority traffic is delayed. Therefore, the QoS module has the option to disable aggregation for high-priority packets. Thus, additional delay will only be introduced for low-priority traffic.

*Packet façade overhead* Using a packet façade to associate control parameters with a packet causes additional processing delay. If all types of packet attributes are not known in advance, they must be stored sequentially in a header byte array. This type of packet implementation is not very efficient, but can be used for any possible combination of packet attributes. When all packet attributes are known in advance, they can be associated with a specific header location. In this case, the packet façade can be implemented very efficiently using a simple switch statement.

*Overhead for storing packets* IDRA requires very few copy actions for processing packets: arriving packets are stored once in the shared queue, and remain there until processing is finished. This is much more efficient than copying a packet once for each protocol layer.

The total packet delay depends strongly on the complexity of the used packet façade and the complexity of the network protocols. However, even when using relatively network protocols, such as the combination of an AODV routing protocol, an LPL-like MAC protocol and 6lowpan packets, the processing delay is less than 20 milliseconds on a telosb [21] sensor node. This delay is well below the duty cycle of most typical WSN MAC protocols.

## 8 Related work

In this section, we will compare our architecture with other proposed architectures for WSNs.

### 8.1 A Sensor Network Architecture (SNA)

The sensor network architecture (SNA) [22] is based on ‘functionality’: the authors analyzed thoroughly which ‘functions’ or ‘components’ are often executed by protocols. They provided a modular MAC layer (called ‘SP’) [23] and a modular routing layer (called ‘NLA’) [6, 24]. A protocol designer can use these modules to ‘build’ a custom network protocol. Additionally, a cross-layer database is provided that shares components such as a message pool (similar to the ‘shared queue’), a link estimation and an extensible neighbor table.

The SNA has several similar goals as IDRA, but differs in the following ways. *(i)* Rather than delegating tasks to a central system, their goal is to enable the quick development of protocol layers, using the provided components for each layer. *(ii)* Protocols need to define their own headers, and must encapsulate packets from higher layers. *(iii)* Dynamic selection between protocols is not supported, and protocols can not view or reuse each others packet attributes. *(iv)* Limited support for energy-efficiency. Since their system can not extract meaningful parameters from packets, they combine full packets rather than only the relevant information. Additionally, they can not aggregate information to non-neighboring nodes. *(v)* Provisions for QoS or heterogeneity are not supported.

A similar component based MAC Layer Architecture (MLA) [25] also includes power management. The memory footprints of SCP-MAC and X-MAC from table 4 are calculated using the MLA architecture.

### 8.2 A declarative sensornet architecture

The declarative sensor network architecture [26, 27] (DSN) aims to facilitate the programming of sensor nodes, using a declarative language (called Snlog). This language provides a high level of abstraction: protocols describe what the code is doing but not how it is doing it. Algorithms are implemented using predicates, tuples, facts and rules.

The compiler represents all this information as tables. Rules are converted to dataflow plans using database operations (Join, Select, Aggregate and Project). Execution of the dataflow plans is triggered by the associated predicates. Finally, the intermediary operators are compiled into a nesC program.

DSN is especially suited for recursive protocols, such as tree construction (which requires only 7 lines of code). Additionally, protocol interoperability can be supported using database scheme matching techniques on the packets. However, the architecture currently has several disadvantages: *(i)* complex data structures are not supported, *(ii)* total memory size increases (up to a factor 3) and *(iii)* no fine grained radio control is supported (which makes the language unsuited for low-level MAC protocols).

### 8.3 Modular architectures

One of the limitations of a layered architecture is that it is difficult to incorporate new cross-layer services since interfaces are explicitly embedded in each layer. An alternative is to completely discard the layered structure.

Instead of using protocol layers, all responsibilities of a protocol layer are divided over separate modules [28] with a well-defined function. For example, a complex MAC layer can be divided into a neighbor management module, a sleep management module, a channel monitoring module and a retransmission module.

To prevent a large number of dependencies between the different modules, modules do not interact with each other directly. Instead, communications between modules go through a cross-layer database repository [29].

The use of a modular architecture has several advantages:

- Duplication of functionality is prevented;
- When developing a new network protocol, existing modules can easily be reused;
- cross-layer information can be exchanged, supporting the development of energy-efficient protocols;
- Depending on the node capabilities or network conditions, it is easy to add or adapt a single module;

In the design of our information driven architecture, no assumptions are made on the use of layering, modular or hybrid approaches. For modular approaches, our filter based protocol registration (Section 3.4) can be expanded to support dynamic call sequences [30]. Registering modules can indicate in which sequence the modules should be executed. Depending on the situation and packet type, the appropriate call sequence can be initiated. Thus, our proposed information driven architecture is easily adaptable to be compatible with both the layered and the modular approach.

## 9 Summary

Wireless sensor networks are used for increasingly complex applications, such as wireless building automation and process and asset monitoring. These applications demand more and more functionalities from the underlying sensor network. They are often deployed on strongly heterogeneous nodes and require adaptive and reliable end-to-end services. Such requirements cannot be supported at a protocol level but should be part of the overall sensornet architecture. However, no architecture currently exists that supports heterogeneity, easy protocol-integration and QoS as part of its architectural design.

Therefore, in this paper, we proposed an information driven sensornet architecture. This architecture is based on the notion that protocols should be simplified to their two main tasks: exchanging information and interacting with the relayed information. By intelligently manipulating this information, the system can support advanced network requirements for next-generation sensornet applications.

More specifically, the information driven approach has the following key advantages:

- by using a packet façade for packet interactions, protocol logic is decoupled from packet representation;
- rather than statically wiring protocols, protocols are dynamically selected (based on protocol-provided filters);

- by providing a shared, system-wide packet queue the overall memory footprint is reduced and system-wide QoS can be enforced;
- heterogeneity is promoted since protocols can be added to a node according to its capabilities;
- and finally, by efficiently combining the information exchanges, the number of transmitted packets can be strongly reduced.

To demonstrate the feasibility of these concepts, we implemented all techniques in a single architecture. We demonstrated that in our system, network protocols require significantly less memory (up to a factor 10), at the price of a larger initial memory cost. In addition, the number of packets per time unit can decrease up to a minimum of $\frac{1}{\Delta T_x}$, with $\Delta T_x$ the lowest information interval. Finally, we demonstrated that the additional processing time of our system is far less than the sleeping delay in typical wireless sensor networks.

To conclude this paper, we are convinced that future applications for WSNs will be very demanding on the network in terms of flexibility, reliability and adaptivity. In this paper, we claimed that network requirements, such as support for QoS, heterogeneity and energy-efficiency, should be part of the architectural design, rather than being added as an afterthought. As such, innovative architectural techniques that support these requirements, like the ones proposed in this paper, will be of great importance to the successful development of next-generation sensornet architectures.

## Acknowledgments

This research is funded by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) through a PhD. grant for E. De Poorter. The author wishes to acknowledge the valuable feedback of colleagues Pieter De Mil, Bart Jooris, Benoît Latré, Evy Troubleyn, Lieven Tytgat and partners from the IBBT-DEUS project.

## References

[1] Eli De Poorter, Ingrid Moerman, and Piet Demeester. An information driven sensornet architecture (best paper award). In *The Third International Conference on Sensor Technologies and Applications (sensorcomm 2009)*, Athens/Glyfada, Greece, June June 18-23, 2009.

[2] I. Akyildiz and I. Kasimoglu. Wireless sensor and actor networks: Research challenges. *Ad Hoc Networks Journal (Elsevier)*, 2(4):351–367, October 2004.

[3] Carlos F. García-Hernández, Pablo H. Ibargüengoytia-González, Joaquín García-Hernández, and Jesús A. Pérez-Díaz. Wireless sensor networks and applications: a survey. *IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.3,*, March 2007.

[4] M. Yarvis, N. Kushalnagar, H. Singh, A. Rangarajan, Y. Liu, and S. Singh. Exploiting heterogeneity in sensor networks. *in Proceedings of the IEEE Infocom*, 2005.

[5] Evy Troubleyn, Eli De Poorter, Ingrid Moerman, and Piet Demeester. AMoQoSA: Adaptive Modular QoS Architecture for Wireless Sensor Networks. *SENSORCOMM 2008, Cap Esterel, France*, August 25-31, 2008.

[6] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, , and I. Stoica. A unifying link abstraction for wireless sensor networks. *SenSys '05, San Diego, CA, USA,*, pages pp. 76–89, Nov. 2005.

[7] Hubert Zimmermann. OSI Reference Model — The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28, no. 4:425 – 432, April 1980.

[8] J. Hoebeke, I. Moerman, B. Dhoedt, and P. Demeester. Towards adaptive ad hoc network routing. *International Journal of Wireless and Mobile Computing*, vol. 1:pp. 1–8, 2005.

[9] R. Rajagopalan and P.K. Varshney. Data-aggregation techniques in sensor networks: a survey. *Communications Surveys & Tutorials, IEEE*, 8(4):48–63, Fourth Quarter 2006.

[10] Eli De Poorter, Stefan Bouckaert, Ingrid Moerman, and Piet Demeester. Broadening the concept of aggregation in wireless sensor networks. *SENSORCOMM 2008, Cap Esterel, France*, August 25-31, 2008.

[11] E. Fasolo, M. Rossi, and M. Widmer, J.and Zorzi. In-network aggregation techniques for wireless sensor networks: a survey. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 14(2):70–87, April 2007.

[12] Tinyos operating system. http://www.tinyos.net/.

[13] W. Ye, F. Silva, and J. Heidemann. Ultra-low duty cycle mac with scheduled channel polling. pages 321–334, Boulder, CO, November 2006.

[14] M. Buettner, G. Yee, E. Anderson, and R. Han. X-MAC: A short preamble mac protocol for duty-cycledwireless networks. pages 307–320, Boulder, CO, November 2006.

[15] J. Hill and D. Culler. Mica: a wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, November 2002.

[16] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *21st Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pages 1567–1576, June 2002.

[17] Collection Tree Protocol (CTP) for tinyOS 2.x., december 2008. http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html.

[18] Tymo source code repository. tymo: Dymo implementation for tinyos, december 2008. http://tymo.sourceforge.net.

[19] LUNAR - Lightweight Underlay Network Ad hoc Routing, december 2008. http://cn.cs.unibas.ch/projects/lunar/.

[20] Ian D. Chakeres and Charles E. Perkins. Dynamic manet on-demand routing protocol (dymo). *IETF Internet Draft, draft-ietf-manet-dymo-12.txt, http://ianchak.com/dymo/draft-ietf-manet-dymo-12.html*, February 2008 (Work in Progress).

[21] Telosb reference datasheet. *http://www.xbow.com/Products/-productdetails.aspx?sid=252*.

[22] Arsalan Tavakoli, Prabal Dutta, Jaein Jeong, Sukun Kim, Jorge Ortiz, David Culler, Phillip Levis, and Scott Shenker. A modular sensornet architecture: past, present, and future directions. *SIGBED Rev.*, 4(3):49–54, 2007.

[23] D. Culler, P. Dutta, C.T. Eee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, and J. Zhao. Towards a sensor network architecture: Lowering the waistline. In *In Proceedings of the Tenth Workshop on Hot Topics in Operating Systems (HotOS X)*, 2005.

[24] C.T. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, and I. Stoica. A modular network layer for sensornets. *In the Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006), Seattle, WA*, November 2006.

[25] Kevin Klues, Gregory Hackmann, Octav Chipara, and Chenyang Lu. A component-based architecture for power-efficient media access control in wireless sensor networks. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 59–72, New York, NY, USA, 2007. ACM.

[26] Arsalan Tavakoli, David Chu, Joseph M. Hellerstein, Phillip Levis, and Scott Shenker. A declarative sensornet architecture. *SIGBED Rev.*, 4(3):55–60, 2007.

[27] David Chu, Joseph M. Hellerstein, and Tsung te Lai. Optimizing declarative sensornets. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 403–404, New York, NY, USA, 2008. ACM.

[28] Robert Braden, Ted Faber, and Mark Handley. From protocol stack to protocol heap: role-based architecture. *SIGCOMM Comput. Commun. Rev.*, 33(1):17–22, 2003.

[29] Tommaso Melodia, Mehmet C. Vuran, and Dario Pompili. The state of the art in cross-layer design for wireless sensor networks. *Wireless Syst./Network Architect., LNCS 3883*, page 78–92, 2006.

[30] Eli De Poorter, Benoît Latré, Ingrid Moerman, and Piet Demeester. Universal modular framework for sensor networks. In *International Workshop on Theoretical and Algorithmic Aspects of Sensor and Ad-hoc Networks (WTASA'07)*, pages pp 88 – 96, Miami, USA, June June 28-29, 2007.