Wireless Netw (2011) 17:1423–1440
DOI 10.1007/s11276-011-0356-5

IDRA: A flexible system architecture for next generation wireless sensor networks

Eli De Poorter · Evy Troubleyn · Ingrid Moerman · Piet Demeester

Published online: 24 May 2011

Abstract Wireless sensor networks consist of embedded devices (sensor nodes), equipped with a low-power radio. They are used for many applications: from wireless building automation to e-health applications. However, due to the limited capabilities of sensor nodes, designing network protocols for these constrained devices is currently very challenging. Therefore, this paper presents the IDRA platform: an information driven architecture designed to support next-generation applications on resource constrained networked objects. IDRA supports simple but useful optimizations at an architectural level. These include support for cross-protocol interactions, energy efficiency optimizations, QoS optimizations (packet priorities, dynamic protocol selection), mobility support and heterogeneous network support. The paper shows how the development of protocols is improved by using an architecture which delegates specific tasks to a central system, decreasing the memory requirements of associated network protocols. A thorough experimental performance analysis demonstrates that IDRA is much more scalable in terms of memory requirements, energy requirements and processing overhead than traditional system architectures. Finally, the paper discusses how the optimizations presented in this paper can be used for the clean-slate design of architectures for other wireless or wired network types.

Keywords Wireless sensor networks · Architecture · QoS · Energy efficiency · Heterogeneity · Protocol selection · Mobility · Experimental evaluation

1 Introduction

1.1 Wireless sensor networks

Wireless sensor networks (WSNs) are traditionally used to monitor large, often inaccessible, areas. They consist of many small devices (sensor nodes), which can sample information from their environment. Often, many sensor nodes are distributed over a large area, whereby information is gathered in a remote sink using multi-hop communication. As a result, monitoring applications can obtain very detailed measurements of the monitored environment [2, 3].

To keep the cost of sensor nodes low, sensor devices are very simple. They consist of a small microprocessor (typically with a clock speed of 8MHz or less), a low-power radio (typically 250 kbps or lower), a sensing device and, finally, a small battery to power these components. Due to the limited capabilities of sensor nodes, past WSN research has mainly focused on the design of simple networking solutions that minimize the energy requirements.

More recently, WSNs have been used for more advanced applications such as wireless building automation, industrial process automation, security monitoring, disaster intervention and medical interventions. These applications benefit greatly from the flexibility and low deployment cost of WSNs. However, these next generation

This article elaborates on the paper '*An Information Driven Sensornet Architecture*' [1]. It adds more in-depth descriptions of the proposed architectural optimizations and adds quantitative analyses for each discussed technique.

E. De Poorter (✉) · E. Troubleyn · I. Moerman · P. Demeester
Department of Information Technology (INTEC),
Ghent University—IBBT, Gaston Crommenlaan 8,
Bus 201, 9050 Ghent, Belgium
e-mail: eli.depoorter@intec.ugent.be

applications impose many network requirements which are not found in traditional WSNs.

- In addition to point-to-sink traffic, *more complex communication patterns* (such as multicast and point-to-point traffic) must also be supported.
- Many future applications use sensor nodes with very diverging capabilities [4]. As a result, future WSNs will become *heterogeneous*, containing both simple nodes (such as light switches) and more complex nodes (such as heating controllers).
- Many commercial applications require mass-produced sensor nodes which are cheaper and even smaller, sometimes up-to-the point where sensor nodes can be implanted. As a result, new ways have to be found to ensure that network protocols have an *even smaller memory footprint* and *consume even less energy*.
- 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.
- Sensor nodes can be used to monitor objects or persons. For these applications, *mobility* should be supported by the network protocols.
- Finally, *Quality-of-Service (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.

Due to these more and more challenging network requirements, developing network protocols for WSNs becomes an increasingly complex issue.

1.2 The need for new architectures

Rather than focusing on the design of optimal network protocols, we strongly believe that redesigning the system architecture is a much more promising approach. 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]. As such, there is a strong need for new architectures that inherently cope with the increasingly challenging network requirements of WSNs. The resulting architecture should ease the integration of network protocols, should support cross-protocol optimizations and have a very low implementation complexity to support even sensor nodes with very limited capabilities. At present, there is no architecture that supports all of these challenges.

Therefore, this paper presents several architectural techniques that can be used to (1) reduce the complexity of developing new network protocols for WSNs, (2) support advanced network requirements such as QoS and (3) support heterogeneous networks. For each of the proposed architectural optimizations, experimental measurements are given that describe how the network performance is improved or, alternatively, which performance penalty is associated with the increase in network flexibility. Finally, the paper evaluates the performance of a system in which all these individual optimizations are combined. Based upon our results, architecture designers should immediately be able to decide whether or not a certain optimization technique is suited for their network requirements. With this goal in mind, we hope that this detailed overview of the advantages and costs of architectural improvements will boost interest in architectural design rather than purely protocol development.

1.3 Remainder of the paper

Section 1 gave an overview of the characteristics of wireless sensor networks and discussed how these characteristics complicate the design of applications and network protocols. Based upon this discussion, IDRA was designed with three main goals in mind. (1) Section 2 illustrates that IDRA simplifies the design of network protocols. (2) Next, Sect. 3 discusses how IDRA is able to support the advanced network requirements that are needed to enable next generation applications sensor applications, such as support for energy efficiency, for diverging application requirements, QoS and mobility. (3) Finally, Sect. 4 demonstrates that the resulting architecture can perform efficiently even in strongly heterogeneous networks. A comprehensive evaluation of all the presented techniques and optimizations discussed in this paper can be found in Sect. 5. Finally, Sect. 6 compares IDRA with existing architectures for wireless sensor networks and Sect. 7 concludes the paper.

2 Simplifying network protocols

Currently, implementing a network protocol is time-consuming and complex. Besides formulating a fully functional algorithm, many unrelated issues must be solved. More specifically, each protocol layer must (1) define a message format (including header and trailer fields), (2) provide buffers to temporarily store packets and (3) gather information from other nodes. We argue that this approach is very inefficient. The main responsibility of a network protocol is to ensure that information is relayed to the correct destination. It makes no sense that every individual protocol layer has to bear the burden of gathering information, providing buffers and implementing header manipulations. Such functions, which are repeated in each protocol layer, should be implemented in a single shared library.

In the IDRA architecture, protocol designers have to consider only the 'information exchanges' when implementing a network protocol. Other tasks, such as packet

creation and buffer provisioning, are delegated to the architecture. As a result, network protocols are simpler and require less memory. In effect, the role of a network protocol is simplified to its 2 main tasks: *exchanging information* and *interacting with the relayed information*.

2.1 Information exchanges

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

- An application sends measured *data values*, such as the 'ROOM_TEMPERATURE', to a central monitoring node;
- a clustering protocol sends *status information* (e.g. 'ENERGY_REMAINING') to all neighboring nodes;
- or a routing protocol sends *control information* (such as a 'ROUTE_REQUEST') to a remote node.

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. The system will transparently create a new packet and encapsulate the parameter into this packet. Whenever a packet arrives at its final destination, the system will extract the information parameters from the packet and will distribute them to the interested protocols and applications.

The main advantages of transferring the creation of packets to the system are: (1) the system can ensure that similar control information is sent only once; (2) multiple interested network protocols can act upon the same exchanged information; (3) protocols are simpler since they do not need to create packets and do not need to interact with packet buffers; and (4) multiple information parameters can be combined into a single packet, so that the number of required packets decreases drastically (Sect. 3.1).

2.2 Interacting with packets

Even when the system encapsulates the exchanged information in a packet format, network protocols must still interact with the forwarded packets. Traditionally, protocol layers do this by associating information with passing packets in the form of a (fixed size) packet header that precedes the packet payload. A packet header typically contains multiple header fields that contain control information. As pointed out in [7], this solution is inflexible, since information that is contained in the headers is not available for higher layers, which limits cross-layer optimization possibilities.

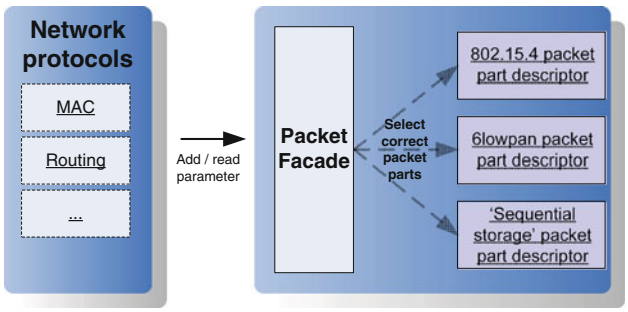

Fig. 1 Through a packet facade, protocols interact with packets. Protocols do not require any knowledge about the actual packet construction

In our information driven architecture, protocols are not tasked with header creation or manipulation. To ensure that network protocols can interpret all incoming packets, network protocols use a 'Packet Facade' (Fig. 1) to interact with packets. Using this packet facade, protocols can associate packet attributes with a packet, such as 'source', 'destination', 'QoS ID' or 'time-to-live'. The protocols do not require any knowledge about the actual packet structure. Instead, the packet facade is responsible for the storage and retrieval of the packet attributes. Added packet attributes can be interpreted by any network protocol, not only the protocol that added the attribute. As a result, *the protocol logic and packet representation are effectively decoupled, and no information is hidden from higher layers*.

To correctly store and retrieve packet attributes, the packet facade should know how each packet is constructed. This information is stored in packet part descriptors. *Packet part descriptors* describe how and where packet attributes are stored in a header (e.g: the header offset, the byte-ordering, the number of allocated bits, etc.). Examples of packet part descriptors are an IEEE 802.15.4 header, an IEEE 802.11 Wi-Fi header or an IP header.

New packet types can be created by combining multiple packet part descriptors. A *packet type* is defined as a unique sequence of one or more well-defined packet part descriptors. For example, an IEEE 802.15.4 packet part descriptor can be combined with a 6lowpan packet part descriptor to create IPv6 compatible packets. To create new packet types, developers can combine existing packet part descriptors, or develop new propriety packet part descriptors. Alternatively, a network designer can design a single highly optimized packet part descriptor that efficiently compresses all packet attributes that are used in his specific network scenario. Finally, it is important to note that network protocols are not limited to the use of (standardized) packet attributes. Packet attributes that are not recognized by any of the packet part descriptors are stored sequentially in the payload using a type-length-value (TLV) representation.

Using a packet facade to associate attributes with a packet has the following advantages: (1) protocol development is simplified, since there is no need to define headers or header operations; (2) packet attributes have a system-wide significance: they can be inspected by the system or by any other protocol; (3) multiple packet types can be supported: the transmitted packet type can transparently be changed without any changes to the protocols (e.g.: 6lowpan, IEEE 802.15.4 or a custom packet).

2.3 A system-wide shared queue

Finally, the system created packets must be stored. Layered networks use a 'store-and-process' approach, wherein each network layer stores its own packets (Fig. 2a). Each protocol requires a large enough internal queue to ensure that all received packets can be stored. Thus, the total amount of buffer memory increases linearly with the number of protocol layers.

As part of the simplification process, the IDRA system is also responsible for storing created and incoming packets. Arriving packets are stored once in a system-wide shared queue and remain there until processing is finished (Fig. 2b). This limits the total number of copy actions in the IDRA system. Network protocols can interact with any of the packets from the shared queue using the packet facade.

As stated by several authors [8, 9], the use of a shared, system-managed queue has several advantages: (1) protocols are simpler and smaller since they do not have to allocate queue memory; (2) packets do not need to be copied between protocols, resulting in less processing overhead; (3) since the queue occupation from all protocols is averaged, less total queue memory is required; and (4) monitoring and managing the total number of packets in the system is simpler.

3 Advanced architectural optimizations

Next generation WSN applications should not only be energy efficient, but they also require support for QoS and mobility. Moreover, as sensor networks become increasingly interactive, the application requirements can change frequently over time. This section discusses how the IDRA architecture efficiently copes with these next generation WSN requirements.

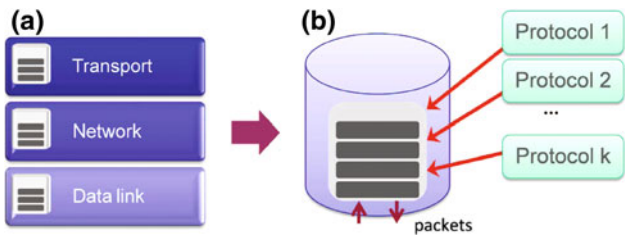

Fig. 2 (**a**) In traditional layered architectures, each network layer allocates a packet buffer. (**b**) In a shared queue approach, only one single, system-wide shared packet buffer is allocated

These next generation requirements can not be solved by adapting a single network protocol. On the contrary, efficient support for requirements, such as quality-of-service or mobility, requires the redesign of several network layers and requires advanced cross-layer cooperation. In an ideal situation, these features should be solved in protocols that are separate from the MAC and routing layer. This way, developed QoS and mobility solutions can be combined with any existing routing or MAC protocol. This separation is only possible when advanced WSN requirements are a part of the architectural design. It is well known that supporting additional features after the design phase of an architecture is increasingly difficult. In fact, *the lack of architectural support for energy efficiency, QoS, mobility and heterogeneity in existing WSN architectures is a major obstacle that hampers the deployment of many next generation applications for WSNs*.

This section demonstrates how these next generation applications requirements can be supported at an architectural level in our information driven system. Thus, our optimizations can be used to transparently enhance existing network protocols with a basic form of QoS and mobility. As a result, it is easier to design advanced QoS and mobility aware network protocols using the IDRA architecture.

3.1 Energy efficiency

In contrast to traditional networks, wireless sensor networks are typically battery powered. Even with a limited battery, a sensor network should have an operational lifetime of at least several years without the need for any manual intervention. In WSNs, most energy is spent when the radio is active. By periodically turning off the radio, the network lifetime increases significantly [10]. However, this approach is only feasible if the number of transmitted packets is very low. To reduce the number of packet transmissions, 'data aggregation' is often applied, which is a technique in which multiple measured data values are combined in a single packet.

Aggregation in WSNs is a well studied research topic [11, 12], on which many specialized aggregation protocols have been proposed. However, these are typically highly optimized for very specific types of traffic flows, and they often require complex fine-tuning to set-up optimal aggregation routes. To remedy this, IDRA contains an in-built aggregation function which can be activated when no other aggregation protocols are provided. This

aggregation is part of the architecture and is 'non-intrusive': no fine-tuning of aggregation settings is required and no additional control messages are sent.

Our main assumption is that not all packet types need to be forwarded immediately. Control packets generated by protocols (e.g. routing, power management, status information) often have a periodic character. Measurements, such as temperature or remaining battery power do not vary a lot between subsequent status updates. Therefore, it is reasonable to assume that these packets are not very time-sensitive, and can be delayed for a short amount of time before being sent.

Whenever a protocol requests the sending of a parameter, the protocol also provides information regarding the maximum delay after which the parameters should be sent. Before encapsulating the parameters in a packet, the system collects the parameters in a central repository, called the *waiting space* (Fig. 3). Whenever a packet is relayed through the node, all information parameters to the same 'next hop' or 'destination' address are added to the packet. Delay-tolerant parameters can remain in the waiting space for up to a per-parameter predefined period of time. 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. To prevent the end-to-end delay from becoming too high, parameters are only delayed in the waiting space of the initial node: packets are not further delayed in intermediate nodes.

In contrast with traditional aggregation protocols, an architectural approach has three main advantages: (1) both application level information and network level control information can be combined, (2) since aggregation is executed at an architectural level, the aggregation approach is compatible with any networking protocol, and (3) this approach does not require any communication overhead between different nodes.

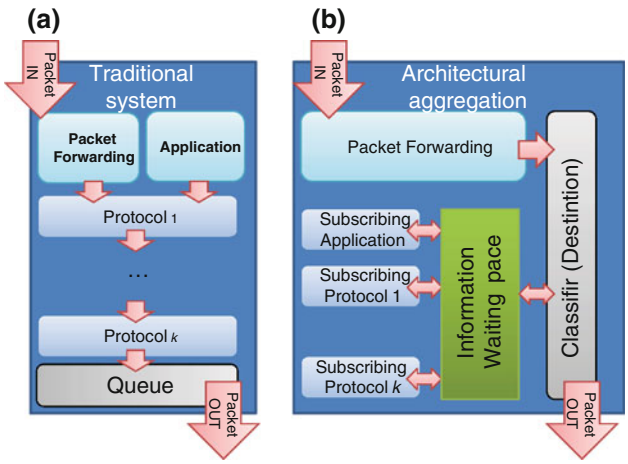

Fig. 3 Extending the data aggregation concept. (**a**) Traditional architecture. (**b**) Architectural support for aggregation

3.2 Supporting diverging application requirements

Sensor networks are used for increasingly complex applications, from controlling thermostatic elements to security and health monitoring applications. These applications have very diverging network requirements in terms of QoS (reliability, maximum delay, etc.) and network characteristics. In addition, a sensor network is typically not an independent entity, but should interact with the outside world. As a result, network protocols for WSNs are becoming increasingly complex: they should support QoS requirements, should interact with a plethora of diverging technologies, and at the same time remain simple enough to implement on a resource constrained sensor node.

IDRA uses an alternative approach in which the network designer is given the option to deploy multiple smaller and more specialized network protocols on a single node. Whenever a packet requires processing, the system is responsible for choosing and activating the most optimal network protocol (Fig. 4). For example, a single node can contain: (1) an efficient broadcast protocol for disseminating information; (2) a data-centric routing protocol for collecting measured data; (3) a complex routing protocol that delivers high QoS guarantees for emergency voice communication; and finally (4) a protocol for routing packets to an external network. The architecture is able to dynamically change between these different routing and MAC protocols at run-time.

Currently, IDRA implements a simple filter-based solution to select the most optimal network protocol for each packet. Each IDRA protocol must register itself by adding one or more *filters* to the system. These filters describe the function of the network protocol and indicate for which packets the protocol is optimized. Consider the following examples:

– A voice routing protocol adds a filter 'QoS_label>5'. All voice packets that require high QoS guarantees will be routed using this specialized protocol.

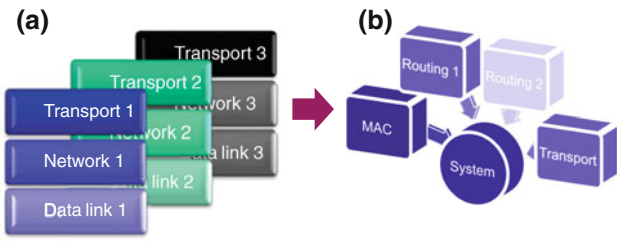

Fig. 4 (**a**) Using a traditional layered approach, the order and types of network protocols are fixed. (**b**) In IDRA, new network protocols can dynamically be added per application requirement. The most optimal network protocols are automatically selected by the system

– A routing protocol implements an efficient broadcast algorithm. It registers itself using the filter 'address = BROADCAST_ADDR'.

Through the packet facade, the system checks if the attributes of the arriving packets match any of the registered filters. IDRA selects the network protocol with most matching filters to process the packet. When no filters match, a default routing and MAC protocol is chosen.

In layered architectures, network protocols are executed in a fixed sequence so that each layer can remove the packet header from incoming packets. In contrast, IDRA users can specify in which order protocols should be executed by defining 'call sequences'. The default IDRA implementation contains a simple, deterministic call sequence that will suffice for most networks. The default call sequence (management protocol → routing protocol → MAC protocol) mimics very closely the behavior of traditional layered architectures. However, developers can define new call sequences to design more flexible systems. It is possible to execute network protocols in any order, and even change the execution sequence at run-time. Section 4 discusses how these advanced call sequences can be used to support heterogeneous networks.

Using this flexible protocol selection approach has several advantages: (1) smaller protocols that fulfill only a single function can be used; (2) these smaller protocols are more suited for resource constrained devices: they are often more stable and are easier to maintain than monolithic protocols; (3) the performance of the network can be optimized by switching between different network protocols, depending on the network circumstances [13].

3.3 Quality-of-service

Before sensor networks can be used for critical and time-sensitive applications, WSNs should be able to deliver Quality-Of-Service (QoS) guarantees. IDRA is optimized for the design of transparent QoS solutions:

– Since all packets are stored in a shared packet queue, the system can monitor all available packets. As a result, the QoS module has a clear view on the number of packets, their current processing state and their expected delay.
– The QoS module can influence the order in which packets are processed and which packets should be transmitted first.
– Through the packet facade, the QoS module can read and modify the attributes of relayed packets at any processing stage. This information can be taken into account for intelligent packet selection and dropping strategies. Similarly, network protocols can request all QoS related attributes and act upon them to the best of their abilities.
– Using dynamic protocol selection, packets with strict QoS requirements can be processed by specialized protocols.
– Analyzing which parameters can be aggregated with relayed packets results in additional processing delay. Therefore, the QoS module has the option to disable aggregation for high-priority packets. As a result, additional delay will only be introduced for low-priority traffic.

A single QoS module that is part of the IDRA system has control of all stored packets. By rewriting this module, new QoS solutions can be implemented. To implement new QoS logic, an interface is available through which the following commands can be given to the system:

– drop a specific (low-priority) packet (useful when the queue is full);
– select which packet should be processed first;
– put the processing of low-priority packets on hold (even when those packets are currently being processed by a protocol);
– activate the most suitable network protocol, depending on the characteristics of each packet;
– indicates which packet should be transmitted first;
– enable or disable aggregation on a per-packet basis.

Together, these commands can be used to design a wide range of possible QoS solutions. The developed QoS controller is *protocol independent*: it can transparently be combined with any IDRA network protocol. Of course, the developed QoS solution can also be combined with QoS aware network protocols for even more sophisticated results [14]. During the evaluation of IDRA, it will be demonstrated that these system commands suffice to transparently add simple QoS features to existing IDRA protocols.

3.4 Mobility support

Wireless sensor networks are also often used for localization or tracking purposes [15]. However, the presence of mobile nodes has a profound influence on the performance of the network. Whenever a node moves, all routes that involve this node have to be set-up again. In addition, the MAC protocol must ensure that communication is possible with all new nodes that arrive in the neighborhood.

IDRA provides the following features to facilitate the design of mobility-aware network protocols.

– A shared neighbor table is provided, which stores (network) statistics for each neighboring node.

– To discover new (or leaving) sensor nodes, the default call sequence can easily be extended with a neighbor discovery protocol (Sect. 3.2), that can be made responsible for updating the neighbor table and adding new neighbors
– Whenever a neighbor is removed, all network protocols are informed about these changes. This way, the routing protocol knows that it should update all routes that involve this node.

The added neighbor discovery protocol can be either active or passive. *Active neighbor discovery protocols* regularly broadcasts information to all neighboring nodes. Newly discovered nodes are added to the neighbor table, whilst all nodes that do not respond are removed. A *passive neighbor monitoring protocol* typically does not exchange messages. Whenever a packet is received from a node, this node is added to the neighbor table. When no packets have been received from a node during a predetermined period, the neighbor is removed. As a result, passive neighbor discovery requires less communication overhead, but reacts more slowly to topology changes.

4 Towards heterogeneous networks

The previous sections presented several optimizations for (1) simplifying the development of network protocols and (2) supporting advanced network requirements such as energy efficiency, dynamic protocol selection, QoS and mobility. These features are adequate for developing next generation sensor applications. However, in the long term, WSNs will become increasingly heterogeneous:

– New, next generation sensor nodes will be added to existing (legacy) WSNs.
– Advanced wireless sensor networks are often deployed on hardware with very diverging capabilities: from light switches to air-conditioning controllers [4, 16].
– Support for interaction with surrounding networks becomes increasingly important. An example is 'the internet of things' [17], which describes a vision in which any object is connected to any other object.

As a result, future sensor networks will know a wider diversity regarding the capabilities of the sensor nodes. These sensor nodes can differ in terms of:

– *node capabilities* (diverging memory, processing or energy provisions);
– *communication methods* (different network protocols, packet types or radio technologies).

This section describes how IDRA can be used to facilitate this transition towards strongly heterogeneous networks.

4.1 Diverging node capabilities

Typical sensor nodes are too simple for complex tasks such as intrusion detection, equipment tracking or for controlling advanced machinery. When additional interaction with the environment is required, more capable nodes ('actuator nodes') are added to the WSN. These actuator nodes are sometimes connected to the power grid, and are often equipped with secondary communication interfaces (e.g. wired or WiFi).

The availability of more capable nodes is often known at the design time of the network. As such, the network protocols can take into account the capabilities of the available nodes.

1. Non-essential protocols can be omitted from nodes with little capabilities (Fig. 5). Typical functions that can be delegated to more capable nodes include data aggregation, position discovery and mobility detection.
2. In addition, the system can execute simpler protocol implementations on lightweight nodes. A typical example is the use of a clustered MAC protocol, in which advanced nodes ('clusterheads') calculate the optimal slot assignments and distribute these to the less capable nodes.

To enable such flexible systems, IDRA developers can design different execution sequences based on the capabilities of each node. By defining custom call sequences, very flexible systems can be implemented (see Sect. 3.2). Moreover, in layered architectures, omitting network protocols can result in conflicts (for example, by not or incorrectly removing protocol headers). In IDRA, packet

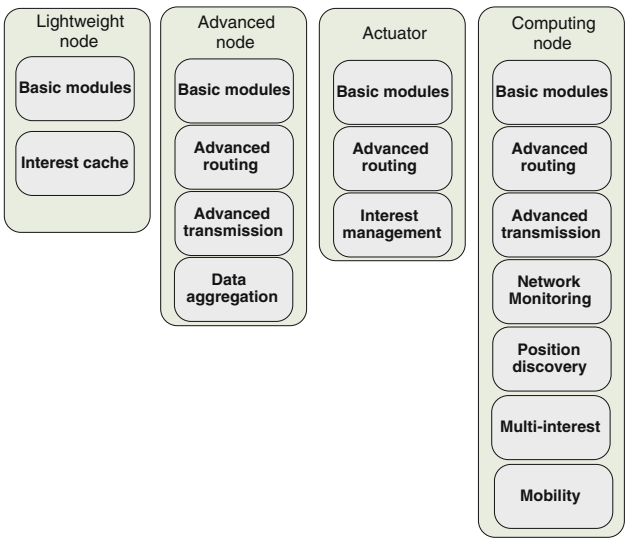

Fig. 5 Depending on their capabilities, the number of protocols can be varied

attributes remain associated with a packet even if network protocols are omitted at intermediate nodes. This flexibility ensures that the IDRA architecture is suitable for both high capacity and low capacity nodes.

4.2 Different communication technologies

Current wireless sensor network typically consist of only a single network technology. However, new technologies are constantly being developed and integrated in existing networks. These co-located technologies will need to cooperate to perform efficiently. By allowing communication between (co-located) networks the network performance increases. For example, sensor packets can be routed over a co-located mesh network to obtain a shorter end-to-end delay for the WSN (Fig. 6). Connecting different types of networks can also provide business advantages: when checking e-mails on a cell phone, rather than using an expensive 3G network, the cell phone can instead use bluetooth to connect with a body area network (BAN). The BAN, in turn, can make a connection with a nearby WiFi gateway to provide cheap internet access.

The IDRA architecture has several built-in solutions to enable connectivity between devices that use different communication technologies. To integrate different networks, the following challenges need to be solved:

1. Co-located networks use *different packet types*. In a heterogeneous environment, multiple packet types can transparently reside on the same node at the same time. To correctly interpret incoming packets (and thus use the correct packet part descriptors), the packet type needs to be uniquely identified. The identification task is part of the IDRA system. To this end, a shared neighbor table is provided that describes for each neighbor which packet type is used for communication. At the moment, this table is configured at compile-time. In the future, additional identification methods will be developed and added to the packet recognition module of IDRA.
2. Co-located networks use *different radio technologies*. To allow direct communication between two networks using different radio technologies, an intermediate node should be available that contains both types of radio interfaces. Using IDRA, each radio interface can have an associated MAC protocol. The shared neighbor table can be used to specify for each neighbor which MAC protocol and radio interface should be used. IDRA will automatically select the correct MAC protocol and send the packet over the correct radio interface.
3. Co-located networks use the *same radio technology but different MAC protocols*. In this situation, multiple MAC protocols should manage the same radio interface. To prevent conflicts, IDRA implements several simple algorithms for resolving MAC conflicts (for example: the radio will only be disabled when all registered MAC protocols have requested a low power radio state).
4. Co-located networks use *different routing protocols*. Dynamic protocol selection ensures that, depending on the capabilities of the destination node, the correct routing protocol is executed. By porting a legacy routing protocol, MAC protocol and packet type, IDRA is fully backwards compatible with existing legacy networks.

IDRA not only supports multiple incoming packet types, it is also possible to automatically convert outgoing packets to a different type. Packet conversion occurs only when a packet must be transmitted to a neighbor that is associated with a different packet type. The conversion mechanism is very simple. When packet conversion is required, the packet facade is first used to create a new packet of the correct type. Next, the packet facade is used to extract all

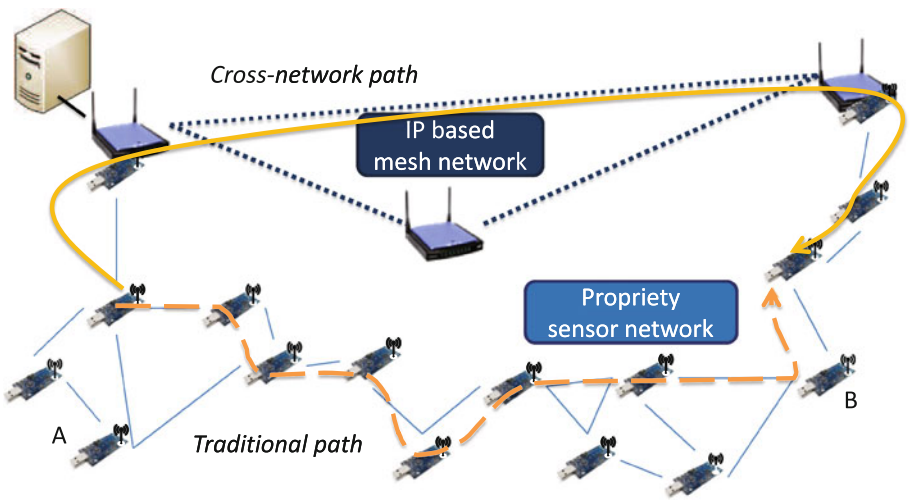

Fig. 6 Routing a packet over multiple co-located network technologies can result in more efficient paths with shorter delays

packet attributes from the original packet (thus dismantling the original packet). Finally, the packet facade is used to add all extracted packet attributes to the newly created packet. The conversion process is fully transparent for the network protocols: the network protocols can not distinguish the new packet from the original packet.

Together, these features ensure that single hop communication is possible with any co-located device that shares at least one common communication interface. The only requirement for communication between two devices is that the common packet type is known in advance. Alternatively, a negotiation protocol can be developed to exchange packet part descriptors between two devices.

4.3 Porting legacy protocols to IDRA

Finally, it is possible to port existing network protocols to IDRA. Three changes need to be made to the internal logic of existing network protocols before they can be used in the IDRA framework. (1) Instead of creating packets to exchange information, protocols and applications hand over a parameter to the global aggregation architecture. The architecture will either create a packet to send the parameter, or add the parameter to a passing packet. (2) Protocols and applications do not inspect the payload of received packets. Instead, the architecture extracts from received packets all the parameters that reached their destination and distributes them to all interested network protocols or applications. (3) Finally, the protocols should register themselves to the protocol selector of the IDRA framework.

5 System evaluation

For the performance evaluation of IDRA, the WiLab wireless sensor test bed [18, 19] was used, which is located in the IBBT - Ghent University office building in Belgium. The WiLab test bed consists of 200 TmoteSky sensor nodes, spread out over 3 floors. By setting the transmissions power of the sensor nodes to an output power of −15 dBm, packets require 4 to 5 hops to be transmitted from one side of the building to the opposite side.

The following sections evaluate how each of the techniques described in Sects. 2 and 3 influence the overall system performance of IDRA. For each of the described techniques, a thorough qualitative and quantitative analysis in terms of processing time, memory overhead, throughput and behavior is provided.

5.1 Evaluation of the aggregation approach

To evaluate the performance of the aggregation approach, all nodes of the test bed were configured to regularly broadcast status messages to their neighbors. In addition, 20% of the nodes were used as a source for point-to-point traffic: they contacted a random other node to which measured information was sent every 60 seconds. To set-up routes, the AODV [20] protocol was used. To compensate for the unreliability of the sensor nodes, the maximum lifetime of a AODV path is set to 10 minutes, after which a new path setup is executed. The maximum parameter delay of the information exchanges was set to 30 seconds.

Figure 7 shows that the resulting number of packet transmissions reduces by more than a factor two when architectural aggregation is enabled. The main reason for this profound reduction of packet transmissions is the fact that any type of information (both application and network information) can be combined with information generated by any other protocol layer.

For a more in-depth comparison of the IDRA aggregation approach with existing techniques, we refer to [21]. This paper shows that our approach can increase the network lifetime by 30 to 50 percent, depending on the MAC protocol.

5.2 System processing overhead

Table 1 shows the contribution of each technique to the average processing delay of a packet. For ease of comparison, the results from this section are given in ms as measured on a 8MHz TMoteSky node. The estimated number of clock cycles can be derived by multiplying the resulting milliseconds by 8000. To get these results, the average execution time that was spent in each module was measured using the network scenario described in Sect. 5.1. The results in Table 1 give a broad indication of the contributions of each technique to the total processing

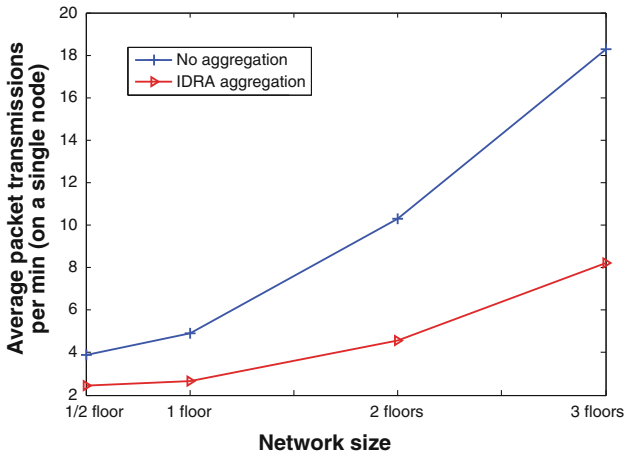

Fig. 7 Performance of in-built IDRA aggregation in a point-to-point scenario

Table 1 Processing overhead of the different architectural components of the system

Optimization	Avg execution time per packet (ms)	Percentage
IDRA system overhead	1.28	20.09
Information management (Sect. 2.1)	0.59	9.26
Packet facade and packet identification (Sect. 2.2)	0.97	15.23
Aggregation (Sect. 3.1)	0.58	9.11
Protocol selection (Sect. 3.2)	0.02	0.31
QoS (Sect. 3.3)	0.51	8.01
Network protocols (CTP & S-MAC)	2.42	37.99
Total overhead	6.37	100

Table 2 Memory footprint (in bytes) of the different architectural components of the system

Component	ROM	RAM
Information exchanges (Sect. 2.1)	1,862	353
Shared queue (Sect. 2.3)	1,934	1,858
Packet facade (interpretor) (Sect. 2.2)	806	8
Aggregation (Sect. 3.1)	1,012	58
Protocol selection (Sect. 3.2)	1,564	86
Quality-of-service (Sect. 3.3)	728	4
Radio controller	11,196	493
Other system components	7,016	909
Total	27,118	3,769

overhead. Of course, these results depend strongly on the network topology and the number of information exchanges. Even so, based on these results it is clear that most processing overhead results from the network protocols (almost 40%). The IDRA system overhead (almost 30%) is mainly for copying packets to the shared queue and for managing timers and tasks. Finally, the flexibility of the packet facade comes at an overhead of about 15% of the total processing overhead, which is a little less than 1 ms.

Table 1 can be used by system developers to decide for each feature whether or not the advantages compensate for the additional processing overhead. However, even when all IDRA features are enabled, the processing delay is about 6 to 7 milliseconds on a TMoteSky sensor node. As such, the total processing overhead is negligible for most WSNs, since the duty cycle (sleep period) of typical MAC protocols for WSNs is typically 200 ms or higher.

5.3 System memory overhead

The memory footprint of the different architectural components is shown in Table 2. The entry 'Other System Components' includes modules for duplicate detection, print statements and timer management. The full architecture requires about 27kb ROM and 4 kB RAM memory, well under the memory limit of most sensor nodes[1]. As mentioned in Sect. 2, this larger initial memory cost is compensated by the smaller size of the IDRA network protocols.

The memory requirements of the discussed optimizations are limited. This shows that these techniques can be used in most typical sensor networks. In addition, for nodes with lower memory limits, several components can be disabled.

5.4 Performance of the shared queue

Copying packets from one layer to another causes a significant processing overhead. According to [22], multi-hop throughput in WSNs is limited mainly by the number of times a packet needs to be copied. On the TMoteSky, the processing overhead for copying a single packet of 128 bytes corresponds to $\pm$ 1530 clock cycles, or 0.19 ms. Using a shared queue, only two copy actions are required: one to copy incoming packets from the radio to the queue and one to copy outgoing packets from the queue to the radio. In contrast, when using a layered 'store-and-process' architecture, packets traverse through each layer twice (once to remove all headers and once to process the packet). As a result, resulting processing overhead increases by twice this amount for *each* protocol layer.

The optimal size of the shared queue depends on many factors, such as the average processing delay of the network protocols (including route set-up), the number of required control messages, the network characteristics and the application requirements. As an example, Table 3 shows the average number of packet drops for different buffer sizes. These results were obtained using the data collection scenario (without aggregation) from Sect. 5.1 on a single floor with 80 sensor nodes. Not only is the processing overhead of a shared queue lower, using a shared queue is also (1) significantly better in terms of packet

[1] Due to added functionality and several optimizations, the total memory requirements differ from those reported in a previous publication [1].

Table 3 Comparing the packet drop ratios (lower is better) for different queue sizes when using a shared queue versus a layered approach

	Total queue entries		Packet drop ratio (%)
Shared queue	1		56.4
	2		5.6
	3		0.1
Layered	Routing	MAC	
	1	1	47.8
	2	1	46.6
	1	2	4.8
	2	2	3.7

drops (for a fixed number of queue entries) and (2) makes it easier to determine the required number of queue entries.

5.5 Support for traffic streams of different priorities

This section demonstrates that the available QoS commands are sufficient to transparently add simple QoS features to existing IDRA protocols. To this end. a simple QoS module was developed for IDRA that implements the following two rules: (1) when the packet queue is full, the packet with the lowest priority is dropped; (2) the packets with the highest priority are always processed and transmitted first.

To evaluate the effectiveness of the available QoS commands, the simple QoS module was added to a scenario whereby a simple collection tree protocol [23] was used to collect data. One node was configured as an always on sink node. The other nodes were generating data packets every 4 seconds but, due to their sleeping scheme, they could only transmit a packet every second. This way, an overloaded network was created where packets had to be dropped. Two types of traffic were generated, each having a different priority level. The high priority traffic flow was generated on only one node, while all the other nodes were generating a low priority traffic flow. Using the provided libraries, the implementation of these QoS policies required only 782 bytes ROM and a processing overhead of 4080 clock cycles (about 0.5 ms on a 8 MHz processor) per packet. Table 4 shows the end-to-end delay and packet drop results from the high priority traffic flow compared with the average of the low priority traffic flows on the same hop level. The delay and packet loss for the high priority stream is significantly lower, even though the network protocols do not support any QoS at all. Thus, when using IDRA, it is possible to increase the global network performance by combining protocol-independent QoS solutions with any network protocol.

Table 4 Influence of the QoS module on the average delay and reliability of packets

	No QoS	Simple QoS
	Avg delay	
Low-priority streams	14,69 s	17,59 s
High-priority stream	15,07 s	1,58 s
	Avg packet reliability	
Low-priority streams	8,90 %	3,76%
High-priority stream	15,67 %	98,22%

5.6 Performance of the packet facade

To decouple the packet structure from the protocol logic, the IDRA architecture relies on the use of a packet facade. This section investigates the cost that is incurred by using a packet facade to update packet attributes.

Table 5 shows the memory requirements that are used by different packet part descriptors to describe a packet structure. The table also compares the processing overhead for creating and manipulating packets with the overhead for traditional methods (e.g: using fixed C structures). Due to the low total processing delay, the overhead is expressed in clock cycles. To measure the packet update overhead, all packet packet attributes that are recognized by the packet part (for example: the sender, receiver and packet ID for a 802.15.4 packet part) were updated.

There is a clear correlation between the complexity of a packet part descriptor and its memory and processing overhead. For example, the 6lowpan packet part descriptor requires up to 588 bytes to describe the header structure and requires about 1300 clock cycles to store the packet attributes. In contrast, the most simple packet part descriptor uses a Type-Length-Value (TLV) representation to sequentially store attributes. This TLV packet part descriptor requires only 34 bytes ROM and is used to store packet attributes in the packet payload. This ensures that all packet attributes can be stored, even when no other packet part knows how to process a certain packet attribute.

In general, creating a new packet using the packet facade does not require significantly more processing than when using more traditional approaches. However, updating a header through the packet facade can require up to 5 times more processing cycles than directly assigning values to header fields. To put this into context, in Sect. 3.1, a packet facade was used in a real-life deployment. As shown earlier in Table 1, the total overhead of the packet facade is less than 1 ms per packet, which is much lower than the overhead of the network protocols themselves. At the cost of this additional processing overhead, the packet facade results in significant additional flexibility in terms of packet construction.

Table 5 Memory footprint (in bytes) and processing overhead (in clock cycles) of the different packet part descriptors

Packet part descriptor	ROM footprint	Header size	Packet creation overhead		Packet update overhead	
			Traditional header	Packet part descriptor	Traditional header	Packet part descriptor
Sequential storage (TLV)	34 bytes	Variable	No equivalent	376 cycles	No equivalent	±620 cycles per update
IEEE 802.15.4	120 bytes	9 bytes	288 cycles	456 cycles	172 cycles	940 cycles
IPv6	396 bytes	Variable	604 cycles	640 cycles	232 cycles	884 cycles
6lowpan (HC4 spec.)	588 bytes	13 bytes	1,860 cycles	2,040 cycles	604 cycles	1,276 cycles

Table 6 Measured throughput of the IDRA architecture (all features are enabled)

Packet Part Type	Packet size		Processing	Measured throughput		
	Payload (bytes)	Total packet (bytes)	Overhead (ms)	IDRA throughput (kbps)	With blocking radio (kbps)	With default TinyOS radio (kbps)
Sequential storage (TLV)	100	128	6.06	167.64	94.488	37.592
IEEE 802.15.4	100	126	6.75	150.368	84.836	37.592
802.15.4 + 6lowpan (HC4)	100	122	6.80	143.472	77.104	37.592
Sequential storage (TLV)	10	38	5.86	60.09	40.128	18.74
IEEE 802.15.4	10	36	6.49	46.816	34.504	12.92
802.15.4 + 6lowpan (HC4)	10	32	6.80	37.632	29.184	9.504

5.7 System throughput

For most WSNs, little emphasis is put on the maximal throughput of the system. Theoretically, an 802.15.4 network has a maximal physical bitrate of 250 kbit/sec. However, MAC protocols for sensor networks make heavy use of duty cycling in the form of sleep schemes. These sleep schemes create artificial bottlenecks in terms of throughput, resulting in a maximal throughput which is typically a factor 10 lower. However, as sensor networks are increasingly used for more demanding applications, such as camera surveillance networks, the importance of the maximum throughput might increase in the future.

Table 6 shows the measured single hop IDRA throughput for different packet types and payload sizes. Creating a packet using more advanced packet part descriptors result in smaller packets, since a higher number of packet attributes will have a fixed header location.

The theoretical single hop throughput is shown in Formula 1. As can be seen, a higher system processing overhead T_{IDRA} results in a lower effective throughput. In general, creating and processing a packet with a simple packet description requires about 6 ms, which results in a measured throughput of about 167 kbps (without radio overhead). The use of a more complex 6lowpan packet type decreases the total packet size and increases the processing delay to 7 ms, which corresponds to a measured throughput of about 143 kbps.

$$Throughput = \frac{PacketSize}{T_{IDRA} + T_{radio} + T_{transmission}} = \frac{PacketSize}{T_{IDRA} + T_{radio} + \frac{PacketSize}{Phys.bitrate}} \quad (1)$$

These measurements correspond to an 'ideal' radio with no processing overhead. In practice, the radio controller also requires CPU cycles. When using a blocking radio, no calculations can be executed by the CPU while the radio is transmitting (T_{radio}). Since the performance of the default TinyOS radio is very low, IDRA includes an optimized CC2420 radio controller, which blocks the CPU for only 4 to 6 milliseconds per packet transmission. Table 6 shows the resulting measured throughput. The combined IDRA and radio overhead for transmitting a packet is up to 12 ms. As a result, the resulting maximal throughput is about 95 kbps.

In contrast, the default TinyOS CC2420 radio controller blocks the CPU for up to 20 ms per packet transmission. In addition, the maximum throughput is limited to about 50 kbps. When using this default radio, the time required for sending a packet is significantly larger than the packet processing time of IDRA, which results in very low throughputs.

Using Sect. 5.2, it is possible to *estimate the cost (in terms of throughput) of each proposed technique*. For example, as shown in Table 1, disabling QoS and aggregation lowers the processing overhead per packet by 1 ms.

A decrease of processing time results in an increase of the throughput. Since the processing time will decrease by 17%, the maximum throughput will increase by a similar percentage from 167 kbps to 197 kbps. Using similar calculations, the throughput cost for new architectural techniques can be estimated before the actual deployment of the network.

To summarize, these results show that the maximum single-hop throughput is currently mainly limited by the radio controller, rather than the system. The use of a non-blocking radio controller would almost double the throughput. Other techniques to increase the throughput include: the use of a faster radio, increasing the CPU clock frequency, using simpler packet types or disabling architectural features such as QoS or aggregation.

5.8 Performance of the network protocols

Section 2 demonstrated that it is easier to design network protocols for IDRA since packet creation, packet interaction and buffer provisioning are delegated to the architecture. As a result, network protocols also require significantly less memory. This is shown in Tables 7 and 8, where the memory requirements of typical layered protocols can be compared to those of different IDRA protocols.

When using a layered architecture, the total memory consumption increase linearly with the number of network protocols. Using IDRA requires a significant initial memory investment (about 12.6 kB ROM and 3.1 kB RAM), but this initial cost is compensated by the lower memory requirements of IDRA protocols. In some cases, the memory footprint of IDRA network protocols is reduced by up to a factor 10. This shows that, using IDRA, it is indeed feasible to combine multiple routing and MAC protocols on a single node.

Table 7 Memory requirements (in bytes) of typical layered WSN network protocols

	Traditional protocol	ROM	RAM
MAC	TOS2.1 MAC [24]	11,528	320
	X-MAC [25]	19,854	876
	SCP-MAC [26]	21,372	1,056
Routing	Lunar [27]	5,000	1,518
	CTP [23]	7,234	1,198
	TYMO [28]	11,404	482(+60 per route)

Table 8 Memory requirements (in bytes) of different IDRA network protocols

	IDRA protocol	ROM	RAM
MAC	LPL MAC [29]	822	176
	S-MAC [30]	1,126	184
	FlexMAC	10,210	858
Routing	CTP [23]	712	130
	AODV [20]	1,836	158(+7 per route)
	HYDRO [31]	1,924	692(+28 per route)
	DYMO [32]	5,008	312(+18 per route)

6 Related work

This section gives an overview of related existing WSN architectures.

6.1 A sensor network architecture (SNA)

The sensor network architecture (SNA) [33] 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') [8] and a modular routing layer (called 'NLA') [6, 34]. A protocol designer can use the available modules to 'build' a custom network protocol. The components are 'glued together' using a cross-layer database that shares information such as a message pool (similar to the 'shared queue'), a link estimation table and an extensible neighbor table. As such, SNA can be regarded a collection of 'puzzle pieces' that can easily be combined to create new network protocols.

The SNA has several similar goals as IDRA, but differs in the following ways. (1) 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. (2) Protocols need to define their own headers and must encapsulate packets from higher layers. (3) Dynamic selection between protocols is not supported, and protocols can not view or reuse each others packet attributes. (4) SNA has only 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. (5) Provisions for QoS or heterogeneity are not supported.

6.2 Mac layer architecture

A similar component-based architecture is the 'MAC Layer Architecture' (MLA) [35]. This architecture provides optimized, reusable components that implement common features that are shared by existing MAC protocols. Similar to the SNA architecture, the main focus of MLA is code reusability.

6.3 The chameleon architecture

The Chameleon architecture [9] is part of the Contiki operating system [36]. Similar to the packet facade presented in this paper, the chameleon architecture uses packet attributes which are transformed into packets by header transformations modules. The architecture includes example header transformation modules for IEEE 802.15.4, UDP/IP and TCP/IP packets. In addition, an interesting transformation module is included that automatically calculates the optimal packing of attributes into a packet.

The main differences with the IDRA architecture are the following. (2) IDRA packet attributes are stored directly at (and read directly from) the correct header offset, which results in minimal processing overhead. In contrast, the Chameleon architecture dismantles every incoming packet and stores each packet attribute at a separate location. As such, the Chameleon approach requires additional buffer spaces and copy operations to process each incoming packet. (2) The Chameleon header transformation modules are implemented in the higher network layers above the MAC protocol. As a result, the MAC header does not profit from the decoupling of protocol logic and packet structure. As discussed in Sect. 4.2, IDRA is able to support multiple packet recognition approaches. (3) The chameleon architecture provides only a single approach to identify incoming packets (a unique 'channel' identifier which is added to each transmitted packet). (4) In the Chameleon architecture, part of the MAC protocol logic needs to be implemented in the Chameleon header transformation module. As a result, the chameleon packet structure modules are significantly larger than the IDRA packet descriptors. (5) No solutions are provided to support energy efficiency, QoS, mobility or heterogeneity at an architectural level.

6.4 A declarative sensornet architecture

The declarative sensor network architecture (DSN) [37, 38] 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: (1) complex data structures are not supported, (2) total memory size increases (up to a factor 3) and (3} no fine grained radio control is supported (which makes the language unsuited for low-level MAC protocols).

6.5 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 [7] 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.

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.

IDRA is designed to support both layered, modular or hybrid approaches. To prevent a large number of dependencies between the different modules, IDRA protocols do not interact with each other directly. Instead, communications between modules go through a cross-layer database repository [39]. To support modular approaches, developers can define new call sequences (Sect. 3.2) that determine the order in which the modules should be executed. To support system-wide (cross-layer) cooperation between protocols a shared neighbor table and information repository is provided. Thus, IDRA is not only specifically designed to be suitable for both layered and layerless approaches, IDRA can also be configured for backwards compatibility with any existing layered or layerless architecture (see Sect. 4).

6.6 Performance comparison of IDRA with existing architectures

Giving a performance-based comparison with existing architectures is not an easy task:

- each related work architecture implements different functionalities and focuses its evaluation on different design goals;

– several of the conceptual architectures still lack a practical implementation;
– some performance metrics, such as ease-of-use, are subjective and can not easily be measured;
– finally, the performance metrics used to evaluate the architectures are often strongly different.

Keeping these limitations in mind, this section aims to compare the performance of IDRA with the performance of the architectures discussed in the related work section.

(1) The sensor network architecture (SNA) does not analyze the architectural behavior, but instead evaluates the performance of the implemented network protocols. The memory requirements of the implemented routing protocols varies from 6140 bytes ROM and 1862 bytes RAM for MintRoute to 9060 bytes ROM and 1889 bytes RAM for BVR routing [34]. As shown in Table 8, the memory requirements of IDRA routing protocols are significantly smaller. On the other hand, the processing overhead of SNA varies from 0.8 ms (for mintrouting) to 3.9 ms (for BVR routing), which is in the same order of magnitude as the IDRA protocol overhead. No advanced network requirements, such as QoS, are evaluated.
(2) The MAC Layer Architecture (MLA) also evaluates the behavior of several implemented MAC protocols. According to [35], the typical overhead of a MAC protocol that uses MLA is about 20kB ROM and 1 kB RAM. Again, as shown in Table 8, the memory requirements of IDRA routing protocols are significantly smaller (up to a factor 10). The processing overhead of the MAC protocols is calculated whilst using sleeping schemes. As such, the delay is in the order of several 100 milliseconds. No results are given about the architectural processing overhead.
(3) The header transformation modules from the Chameleon architecture [36] have a similar function as the IDRA packet facade. However, Chameleon header modules also contain protocol logic. For example, the UDP/IP module also includes the ARP protocol. The memory requirements of the Chameleon header modules vary from 475 bytes for a TLV-representation to 6042 bytes for the TCP/IP representation. Since the IDRA packet part descriptors do not contain any protocol logic, the IDRA header descriptors are smaller by more than a factor 10[2]. The processing overhead of the Chameleon header transformation modules is similar to the packet processing overhead from IDRA, which is shown in Table 5.
(4) Similar to SNA and MLA, the evaluation of the declarative network architecture [37, 38] focuses on protocol behavior, rather than architectural behavior. According to the paper, the high level of abstraction offered comes at a high memory cost. For example, the implemented CTP routing protocol requires 48.8 kB ROM and 3.2 kB RAM, which is more than the available memory on many sensor nodes.
(5) Finally, no performance comparison can be made with the heap based architecture from [7] since this architecture was not implemented in a proof-of-concept.

To conclude, the performance analysis of existing architectures mainly focuses on demonstrating that the architecture can be used to implement new network protocols. It is unfortunate that these papers do not evaluate the influence of their design choices: this information would have been very valuable for designers of future protocol architectures. However, when comparing the performance of the implemented network protocols, the IDRA architecture typically outperforms existing architectures, even those that offer less functionalities.

7 Conclusions

In this paper, the IDRA architecture was presented and evaluated. IDRA is designed for next generation sensor applications, such as e-health, wireless building automation, asset tracking or emergency rescue operations. These next generation applications impose very challenging network requirements which are difficult to support on resource constrained nodes. Since existing traditional layered architectures are not designed with these constraints in mind, new and innovative system architectures are required.

To solve this problem, the IDRA architecture was created. This architecture has three main goals: (1) the simplification of network protocols and (2) support for advanced requirements such as energy efficiency, flexibility, QoS and mobility at an architectural level and finally (3) support for an easy transition towards fully heterogeneous network environments.

The first goal, the simplification of network protocols, is realized by delegating common operations to the system. More specifically, the system is responsible for queue provisioning, packet generation and all packet interactions. By providing a separate packet facade for packet interactions, protocols are not tied to a specific packet structure.

[2] Even though the IDRA packet part descriptors do not contain protocol logic, it is possible in IDRA to associate packet-dependent protocol logic to existing packet types. For example, a ARP protocol can register itself to be executed before the transmission of a UDP/IP packet (see Sect. 3.2).

Thus, IDRA network protocols only worry about exchanging information. The system is responsible for managing all information exchanges.

By intelligently manipulating the exchanged information, the system can fulfill the second goal: supporting advanced network requirements for next generation sensor applications. (1) *Energy efficiency* is supported by intelligently combining multiple information exchanges in a single packet. (2) *Flexibility* is improved since multiple network protocols can be designed that are optimized for a specific function. The system will dynamically select the optimal network protocol on a per-packet base. (3) System wide *quality-of-service (QoS)* is enforced by intelligently managing the shared queue. Moreover, the developed QoS solutions can transparently be combined with any IDRA network protocol. And finally, (4) to support *mobility*, a shared neighbor table notifies all network protocols of arriving or leaving nodes.

With regard to the third goal: it was argued that the IDRA architecture can be used to facilitate a future transition towards strongly heterogeneous networks. To cope with different capabilities, network protocols can be added to a node according to its resources. The system supports transparent communication with co-located networks that use different communication technologies by intelligently managing multiple radio interfaces and automatically converting outgoing packets to the correct packet type.

Finally, the overhead of each of these techniques was thoroughly evaluated. As part of the evaluation, it was shown that the built-in aggregation mechanism can more than double the network lifetime, that the single hop throughput of our system is up to 4 times higher than the throughput of the default TinyOS radio implementation and that IDRA network protocols require significantly less memory, at the cost of a larger initial architecture memory cost. Moreover, it was demonstrated that the processing and memory overhead of each architectural technique is small enough to be implemented on resource constrained embedded devices. In addition, to help users decide which techniques are suitable for their applications needs, the paper includes detailed overviews that give a quantitative analysis of the overhead of each technique.

To conclude this paper, we are convinced that future applications for WSNs will be very demanding for the network in terms of flexibility, reliability and adaptivity. We claim that support for these network requirements should be part of the architectural design, rather than being added as an afterthought. As such, innovative architectural techniques that support these requirements will be indispensable for the successful development of next generation network architectures.

Acknowledgements This research is funded by the FP7 SPITFIRE project, by the FWO-Fl CLAWS project, by the IWT-SBO SymbioNets project and by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) through a PhD. grant for E. De Poorter and E. Troubleyn. The author also wishes to thank L. Tytgat for implementing the optimized CC2420 radio controller.

References

1. Poorter, E. D., Moerman, I., & Demeester, P. (2009). *An information driven sensornet architecture (best paper award)*. In The third international conference on sensor technologies and applications (sensorcomm 2009), Athens/Glyfada, Greece, June 18–23.
2. Akyildiz, I. F., Su, W., Skarasubramaniam, Y., & Cayirci, E. (2002). Wireless sensor networks: A survey. *Computer Networks, 38*, 393–422.
3. Bharathidasan, A., & Ponduru, V. A. S. (2002). *Sensor networks: An overview*. Technical report, Department. of Computer Science, University of California at Davis.
4. Yarvis, M., Kushalnagar, N., Singh, H., Rangarajan, A., Liu, Y., & Singh, S. (2005). Exploiting heterogeneity in sensor networks. In *Proceedings of the IEEE Infocom*.
5. Troubleyn, E., Poorter, E. D., Moerman, I., & Demeester, P. (2008). *AMoQoSA: Adaptive modular QoS architecture for wireless sensor networks*. SENSORCOMM 2008, Cap Esterel, France.
6. Polastre, J., Hui, J., Levis, P., Zhao, J., Culler, D., Shenker, S., & Stoica, I. (2005). *A unifying link abstraction for wireless sensor networks*. SenSys '05, San Diego, CA, USA, pp. 76–89.
7. Braden, R., Faber, T., & Handley, M. (2003). From protocol stack to protocol heap: role-based architecture. *SIGCOMM Computer Communications Review, 33*(1), 17–22.
8. Culler, D., Dutta, P., Eee, C. T., Fonseca, R., Hui, J., Levis, P., et al. (2005). Towards a sensor network architecture: Lowering the waistline. In *Proceedings of the tenth workshop on hot topics in operating systems (HotOS X)*.
9. Dunkels, A., Österlind, F., & He, Z. (2007). An adaptive communication architecture for wireless sensor networks. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, (pp. 335–349). New York, NY, USA: ACM.
10. Demirkol, I., Ersoy, C., & Alagoz, F. (2005). *MAC protocols for wireless sensor networks: A survey*. IEEE Communications Magazine.
11. Rajagopalan, R., & Varshney, P. K. (2006). Data-aggregation techniques in sensor networks: A survey. *Communications Surveys & Tutorials, IEEE, 8*(4), 48–63. Fourth Quarter.
12. Fasolo, E., Rossi, M., Widmer, J., & Zorzi, M. (2007). In-network aggregation techniques for wireless sensor networks: A survey. *Wireless Communications, IEEE, 14*(2), 70–87.
13. Hoebeke, J., Moerman, I., Dhoedt, B., & Demeester, P. (2005). Towards adaptive ad hoc network routing. *International Journal of Wireless and Mobile Computing, 1*.
14. Evy, T., Eli, D. P., Peter, R., Moerma, I., & Piet, D. (2010). Supporting protocol-independent adaptive qos in wireless sensor networks. In *Sensor networks, ubiquitous, and trustworthy computing (SUTC)*, 2010 IEEE international conference on (pp. 253 –260).
15. Ali, M., Voigt, T., & Uzmi, Z. A. (2006). Mobility management in sensor networks. *Workshop proceedings of 2nd IEEE DCOSS*.
16. Akyildiz, I., & Kasimoglu, I. (2004). Wireless sensor and actor networks: Research challenges. *Ad Hoc Networks Journal (Elsevier), 2*(4), 351–367.
17. The internet of things. ITU Internet Reports. (2005).

18. Ibbt ilab.t wireless sensor test bed. http://ilabt.ibbt.be.
19. Tytgat, L., Jooris, B., De Mil, P., Latre, B., Moerman, I., & Demeester, P. (2009). *Demo abstract: Wilab, a real-life wireless sensor testbed with environment emulation*. Published in European conference on Wireless Sensor Networks, EWSN adjunct poster proceedings (EWSN), Cork, Ireland.
20. Ad hoc on-demand distance vector (aodv) routing. networking group request for comments (rfc): 3561, http://tools.ietf.org/html/rfc3561, (July 2003).
21. Poorter, E. D., Bouckaert, S., Moerman, I., & Demeester, P. (2010). Non-intrusive aggregation in wireless sensor networks. *Ad Hoc Networks Journal (Elsevier)*.
22. Osterlind, F., & Dunkels, A. (2008). *Approaching the maximum 802.15.4 multi-hop throughput*. Technical report, Swedish Institute of Computer Science, SICS Technical Report T2008:05, ISSN 1100-3154.
23. Collection Tree Protocol (CTP) for tinyOS 2.x. (2008). http://www.tinyos.net/tinyos-2.x/doc/html/tep123.htm.
24. Tinyos operating system. http://www.tinyos.net.
25. Buettner, M., Yee, G., Anderson, E., & Han, R. (2006). *X-MAC: A short preamble mac protocol for duty-cycledwireless networks* (pp. 307–320). Boulder, CO.
26. Ye, W., Silva, F., & Heidemann, J. (2006). *Ultra-low duty cycle mac with scheduled channel polling* (pp. 321–334). Boulder, CO.
27. LUNAR-Lightweight Underlay Network Ad hoc Routing. (2008). http://cn.cs.unibas.ch/projects/lunar.
28. Tymo source code repository. tymo: Dymo implementation for tinyos. http://tymo.sourceforge.ne.
29. Hill, J., & Culler, D. (2002). Mica: A wireless platform for deeply embedded networks. *IEEE Micro, 22*(6), 12–24.
30. Ye, W., Heidemann, J., & Estrin , D. (2002). An energy-efficient MAC protocol for wireless sensor networks. In *21st conference of the IEEE computer and communications societies (INFOCOM)*, Vol. 3, (pp. 1567–1576).
31. Tavakoli, A., & Culler, D. (2009). HYDRO: A hybrid routing protocol for lossy and low power networks. IETF Internet Draft, http://tools.ietf.org/html/draft-tavakoli-hydro-0.
32. Chakeres, I. D., & Perkins, C. E. (2008). *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.htm.
33. Tavakoli, A., Dutta, P., Jeong, J., Kim, S., Ortiz, J., Culler, D., Levis, P., & Shenker, S. (2007). A modular sensornet architecture: past, present, and future directions. *SIGBED Rev., 4*(3), 49–54.
34. Ee, C. T., Fonseca, R., Kim, S., Moon, D., Tavakoli, A., Culler, D., Shenker, S., & Stoica, I. (November 2006). A modular network layer for sensornets. In *the Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006)*, Seattle, WA.
35. Klues, K., Hackmann, G., Chipara, O., & Lu, C. (2007). 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* (pp. 59–72). New York, NY, USA: ACM.
36. Contiki - the operating system for embedded smart objects—the internet of things . http://www.sics.se/contiki.
37. Tavakoli, A., Chu, D., Hellerstein, J. M., Levis, P., & Shenker, S. (2007). A declarative sensornet architecture. *SIGBED Rev., 4*(3), 55–60.
38. Chu, D., Hellerstein, J. M., te Lai, T. (2008). Optimizing declarative sensornets. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, (pp. 403–404). New York, NY, USA, ACM.
39. Melodia, T., Vuran, M. C., & Pompili, D. (2006). The state of the art in cross-layer design for wireless sensor networks. *Wireless Syst./Network Architect*., LNCS 3883, pp. 78–92.

Author Biographies

Eli De Poorter received his masters degree in Computer Engineering from Ghent University, Belgium, in 2006. He received his PhD degree from the Department of Information Technology at Ghent University in 2011. His main research interests include the development of system architectures for wireless sensor and ad hoc networks, the development of wireless network protocols and future internet architectures.

Evy Troubleyn received the M.Sc. degree in electrotechnical engineering from Ghent University, Belgium, in July 2007. Since August 2007, she is affiliated as a Ph.D. Student with the Department of Information Technology (INTEC) at Ghent University. Her main research interests include the development of QoS-aware netwerk protocols and architectures for wireless sensor and actuator networks.

Ingrid Moerman received her degree in Electrical Engineering (1987) and the Ph.D degree (1992) from the Ghent University, where she became a part-time professor in 2000. She is a staff member of the research group on broadband communication networks and distributed software (IBCN), where she is leading the research on mobile and wireless communication networks. Since 2006 she joined the Interdisciplinary institute for BroadBand Technology (IBBT), where she is coordinating several interdisciplinary research projects. Her main research interests include: wireless broadband networks for fast moving users, mobile ad hoc networks, personal networks, virtual private ad hoc networks, wireless body area networks, wireless sensor and actuator networks, wireless mesh networks, fixed mobile convergence, protocol boosting on wireless links, QoS support in mobile & wireless networks, intelligent transport systems, self-optimization in next-generation mobile networks, network architectures and protocols for heterogeneous mobile and wireless networks. She is author or co-author of more than 400 publications in international journals or conference proceedings.

Piet Demeester received the Masters degree in Electro-technical engineering and the Ph.D degree from the Ghent University, Gent, Belgium in 1984 and 1988, respectively. He is a full-time professor at Ghent University where he teaches courses in communication networks. He is the head of the Broadband Communication Networks group (http://www.ibcn.intec.ugent.be). His research interests include: multi-layer IP-optical networks, mobile networks, end-to-end quality of service, grid computing, network and service management, distributed software and multimedia applications. He has published over 500 papers in these areas in international journals and conference proceedings. In this research domain he was and is a member of several program committees of international conferences, such as: OFC, ECOC, ICC, Globecom, Infocom and DRCN. He is a fellow of the IEEE.