Information-Sharing Protocol Architectures for Sensor Networks: the State of the Art and a New Solution*

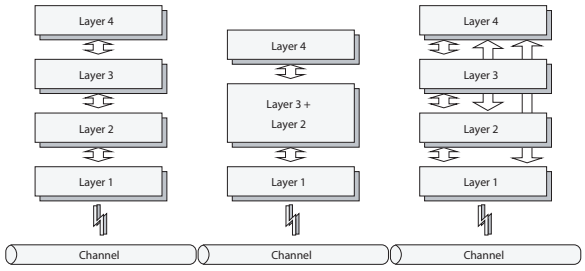
Christophe J. Merlin *merlin@ece.rochester.edu*
Chen-Hsiang Feng *feng@seas.rochester.edu*
Wendi B. Heinzelman *wheinzel@ece.rochester.edu*
Department of Electrical and Computer Engineering, University of Rochester, Rochester, NY, USA

Protocols for wireless sensor networks (WSNs) are very diverse. Reflecting this diversity, no single protocol architecture for WSNs dominates: programmers often modify the legacy-architecture to fit their set of protocols in the stack. However, there exists desirable goals for a sensor network architecture: modularity, flexibility and universality. At the same time, a WSN architecture should enable the protocols to achieve long network lifetimes for various applications. These are, in general, conflicting goals, with the former achieved using layered architectures and the latter obtained through cross-layer interactions. A good balance can be provided by architectures that enable layers to share common information, as these architectures allow for cross-layer protocol improvements, while preventing some of the short-comings of cross-layer designs. Confirming this observation, some architectures that enable information-sharing have been proposed in recent years. We survey these state-of-the-art information-sharing architectures for WSNs, and we introduce X-Lisa, a novel Cross-Layer Information-Sharing Architecture that provides many desirable properties such as flexibility and simplicity, and offers programmers a modular framework, simplifying cross-layer interactions.

I. Introduction

While protocols and algorithms for wireless sensor networks (WSNs) have been the subject of much research, little attention has been paid to sustainable architectures for these networks. The use of WSNs in every day life has been slow in coming, as sensor networks still endure challenges in energy conservation and bandwidth use, as well as the lack of a durable and flexible yet supportive architecture. Historically, little thought has been given to specific architectures for wireless sensor networks, with the most widely used architecture inherited from wired computer networks in the form of the OSI model.

To date, there have been numerous proposals of cross-layer protocols, which use the specificities of sensor networks to improve the network lifetime and the response to the end application. The word "cross-layer" may refer to various designs, many of which are identified by Srivastava et al. in [1]. One type of cross-layering allows information to be shared among several (non-adjacent) layers. Another type of cross-layering fuses two or more layers into a single, integrated layer. Figure 1 illustrates some of the differences between the various stack designs. Whatever their design, cross-layer algorithms leverage information from various sources to improve certain aspects of the network behavior. A supporting architecture should facilitate this process.



(a) Layered (b) Layer Fusion (c) Info. Exchange

Figure 1: Examples of cross-layer designs.

To have a chance at sustaining the development of wireless sensor networks, a protocol architecture should exhibit some desirable characteristics. Flexibility is among the most important ones: new protocols or improved versions of existing ones should find their way into current WSN deployments or developments in order to gain from the latest technology. However, this benefit may be outweighed by the amount of work necessary to include them into an existing framework. A strong WSN architecture should thus ease protocol swapping on any platform.

Information freshness comes next as protocols often rely on a set of parameters that define a local sub-network at a

*This work was supported in part by NSF #CNS-0448046.

certain time. An architecture should provide an up-to-date vision of a node's neighborhood. Additionally, simplicity: in order to guarantee the quick adoption of WSNs as a preferred solution for industry, an architecture should be simple to use. We believe a simple architecture is one that does not carry operations typically reserved to protocols (packet reordering, neighbor selection, etc.) and that provides a common access to its data structures.

Finally, the architecture should also incur low overhead for various applications. These are, in general, conflicting goals, with, for example, the goal of swapping protocols easily achieved using layered designs and adaptation to specific applications favored by cross-layer designs.

Following the seminal work of Kawadia et al. [2] cautioning researchers on cross-layer protocols, many use great care—if not a dose of skepticism—in designing cross-layer schemes. One solution to prevent the interweaving of protocols such that changes in one protocol can counterproductively affect others (the *spaghetti design* mentioned in [2]), is to retain a traditional layered structure in the protocol stack but share information among the layers, as promoted in [3]. While such architectures do not eliminate the need for careful protocol design, they can guarantee the availability and correctness of information shared among many levels of the stack.

The first steps taken towards such an information-sharing protocol architecture occurred in the field of mobile ad-hoc networks (MANETs) with MobileMAN [4] and CrossTalk [5]. Later, SNA [6] was introduced specifically for wireless sensor networks. Chameleon [7] shares many of the goals of SNA but provides a different solution. XLM [8], proposed by Akyildiz et al., takes the interesting opposite solution and fuses all communication layers to best support the sensor network application. These architectures present a variety of approaches to meeting the goals of flexibility, universality, simplicity and low overhead.

The contributions of this paper are twofold: it surveys the above-mentioned architectures, exposing their relative strengths and weaknesses against a set of desirable (and at times, contradictory) goals, and it introduces X-Lisa, an easy to use and maintain architecture for information-sharing. X-Lisa incorporates many of the benefits of existing architectures and improves on their design by providing service support, information propagation adapted to WSNs, and a greater level of flexibility under TinyOS.

In the following, section II introduces the architectures and designs surveyed in this paper. Section III provides a detailed description of the surveyed architectures, and section IV evaluates the merits of each of them. Because the existing work does not meet the set of desirable goals we set for a WSN-specific architecture, we propose X-Lisa in section V. Section VI provides evaluation of X-Lisa and section VII concludes this paper.

II. Related Work

This section provides an overview of related work on cross-layer protocol architectures. Relevant work surveyed in more detail in the following sections is omitted.

In [1], Srivastava et al. provide a definition of cross-layer designs and a survey of existing cross-layer models. The authors define cross-layer interactions as back-and-forth information flows, merging of adjacent layers, design coupling without a common interface, and vertical calibration across layers. They also list implementations for cross-layer interactions: explicit interfaces between different layers, shared databases, and heap organization, which provides new abstractions (no protocol layers).

Whitehouse et al. introduced Hood [9], a neighborhood abstraction for WSNs that allows nodes to identify neighbors with variables of interest. Nodes define *attributes* that may be shared with neighbors. Upon receiving attributes, each node evaluates whether these are valuable enough to be recorded in a neighbor list. Because it is not an architecture, it is not part of this study. However, the solution retained to manage its neighbor table is similar to ours.

In [10], Wang et al. survey existing cross-layer signaling methods, which most closely corresponds to the explicit interfaces architecture mentioned above. Additionally, the authors propose CLASS (Cross-LAyer Signalling Shortcuts), an architecture that allows propagation of cross-layer messages through out-of-band signaling. Because Wang et al.'s work already surveyed the state of this type of architecture extensively, the focus of this paper does not include these architectures.

Sadler et al.'s work [11] propose a shared *platform* among all layers of a protocol stack for cross-layer optimizations. The authors recommend using a table of interchangeable nodes (capable of handling the same application request) that lists equivalent nodes to be used by the routing protocol when a link breaks. The criteria used are connection oriented since the work focuses on providing reliable links in MANETs. While the *platform* certainly introduced architecture choices made by Sadler et al., the paper focuses on protocol issues rather than a universal architecture. The principles guiding the shared *platform* are, however, well represented in the surveyed architectures and are the object of further study in this work.

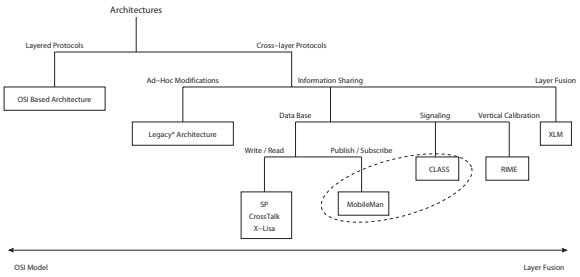


Figure 2: Classification of some architectures for MANETs and WSNs. MobileMan abstracts a data base through signaling, as highlighted by the dotted circle.

III. Architectural Approaches

The widespread success of the OSI model [12] in wired networks provided a starting point in architectural design for protocol stacks in the new fields of MANETs and WSNs, with the OSI model constituting the legacy architecture for these new, more complex networks. Although the OSI model had not been intended for the specificities of WSNs and MANETs, this architecture proved flexible enough (with the help of some ad-hoc violations to accommodate cross-layering designs) to stimulate the growth of these two fields. In this traditional architecture, direct communication is only permitted between adjacent layers. Protocols may obtain information from the packet headers and data units, and from the same layers in distant or direct neighbors. This incurred more horizontal communication (between the same layers of different neighbors), which the bandwidth of wired networks could easily support. However, cross-layer protocols have introduced architectural violations that allow a layer to communicate in some form or another with non-adjacent layers. Such designs aim to benefit from the specificities of WSNs to increase network lifetime and application quality of service (QoS) support, and to avoid wasting bandwidth, a scarce resource in WSNs.

Figure 2 presents a taxonomy of the architectures surveyed in this work. We focus on the designs that allow information-sharing through a shared database. We also include XLM because it supports cross-layering while taking the counter-part of information-sharing.

III.A. MobileMan: Subscription to an Abstracted Database

MobileMan [4] provides an abstracted database, called the network status (NeSt), which is made available to all layers of the stack through a publish / subscribe API. NeSt organizes the exchange of information in the stack: a protocol that needs information from other layers has to register with NeSt and subscribe to event notifications regarding this information. Protocols whose information is of interest to others must notify NeSt of the occurrence of an event. NeSt then delivers the incidence match to the various protocol subscribers.

To the best of our knowledge, NeSt does not organize information exchange about a node's neighbors (horizontal information-sharing), and it leaves the burden of event notification to data-supplying protocols. Adding such functionalities to all the protocols in the stack may involve levels of refinement and complication not suited for MANETs, and *a fortiori* WSNs. This is because the goal of MobileMan is to adapt Internet legacy protocols and applications to MANETs, where the benefits of cross-layering are greater. While MobileMan facilitates protocol swapping, the end goal is to support Internet technology over wireless ad-hoc technologies.

The solutions retained by the MobileMan workgroup are elegant and ingenious: NeSt does not store data but merely provides an abstraction for information exchange. It also allows complex events to be disambiguated and reported to the protocols that have an interest in them (using a publish / subscribe model). However, NeSt asks protocols to follow complex registration processes, possibly causing information exchange to be hindered by very procedural access.

III.B. CrossTalk: A Common Database for MANETs

Because publish / subscribe mechanisms to a database, such as those found in MobileMan, may be complex and lack generality, a non-abstracted database made available to all layers in the stack may be more suited to the needs of networks with limited resources. CrossTalk [5] exemplifies how such a database is used and populated.

The goals of CrossTalk are similar to those of MobileMan: adaptability and flexibility to allow ease of protocol maintenance and replacement. CrossTalk provides local information to all protocols in the stack, as well as a *global* view of the network: a set of parameters deemed of interest to the protocols is gathered about distant nodes. A global view of the network is thus propagated, forming a context for each node. With this information, individual nodes can evaluate their relative situation and modify their behavior accordingly.

The global view of the network is formed by propagating information over several hops: a source node appends the set of parameters that are then read by every relay node on the path. Winter et al. argue that a global view of the network at the individual nodes can greatly improve the per-

formance of the network. However, the implementation of CrossTalk was carried out in NS-2, a rich programming language. Under TinyOS, the incurred per-packet overhead can be (conservatively) broken down into the following: $(2 + 2)\ B$ for 2D location, $8\ B$ for time stamping, and $n\ B$ for the exchanged information.

CrossTalk and MobileMAN seem particularly well adapted to MANETs. However, the characteristics of WSNs limits the benefits of these two approaches.

III.C. SNA: Abstraction of Lower Layers and Basic Functions

Culler et al. proposed the Sensor Network Architecture (SNA) in [13] then [6] with the main objective to provide greater modularity to sensor hardware designs and communication protocols. The suggestion made in [13] was to decouple aspects of the software from the underlying hardware in order to abstract the platform on which the network stack is set. To do so, the Sensornet Protocol (SP) bridges the link and network layers by abstracting key parameters of the lower layers such as link quality and scheduling information. In the next step [6], Culler et al. identify common functionalities to encourage code-reuse and runtime function sharing. SNA breaks the network layer down into reusable communication modules.

SNA retains a layered structure, providing detailed information to the various network protocols present on a node. Two predefined structures within SP serve as information repositories: the "Neighbor Table," which maintains information about direct and relevant neighbors, and the "Message Pool," which allows protocols to request message transmissions.

The neighbor table allows several protocols to share information otherwise maintained in redundant structures. A neighbor table entry usually consists of the neighbor address or *ID*, link quality, and scheduling information. Because the number of neighbors may be greater than the number of existing spots in the table, SP implements an advanced table maintenance scheme: before adding a new neighbor to the list, SP polls every protocol. If at least one protocol agrees, the node is inserted. Protocols are also notified when a neighbor is evicted from the table. Since SP is a protocol, its actions go beyond that of an architecture: if the power management schedule of a listed neighbor is expiring, SP asks the network and link layers to provide a new schedule. It is not clear whether the neighbor table maintains information about the node itself.

Arguably, SNA is a considerable leap forward in WSN architectures, as it provides an unsurpassed level of flexibility and universality under TinyOS. The interface between the various network protocols and data links is rich and advanced, but also quite complex. This may hinder fast deployment of WSNs. Additionally, SP does not propagate neighbor information automatically and solely relies on protocols to populate the neighbor table (although SP may post requests regarding table information freshness). As a whole, SNA tries to achieve a different goal from ours since we are more concerned about data repository reuse rather than algorithm and code reuse.

III.D. Chameleon: Abstraction of Communication Protocols

The main goal of the Chameleon architecture [7] is to abstract communication layers so that WSN protocols may run over any network, from 802.15.4 to IP. Abstraction is achieved thanks to packet *attributes*, an abstract representation of information contained in packets. Rime, a layer contained within Chameleon, takes care of mapping attributes to any standard header.

Additionally, cross-layer interactions are supported through "vertical calibration": information is contained in the *attributes* of packets that are passed between layers. However, in order to propagate information to all layers, the packet headers are not removed after being processed.

Like SNA, the ultimate goal of Chameleon is to provide abstraction to lower layers by identifying basic protocol primitives, although Dunkels et al. selected different ones from SNA. It differs from our intention to allow cross-layer interactions between protocols, since information is not shared among all layers in the stack.

III.E. XLM, The Counterpart: Fused Layers

Akyildiz et al. proposed XLM [8] as a fused-layer module, regrouping all protocols from the data link to the node activation layers. Because this organization of the protocols has a redefining impact on the larger architecture, we consider XLM as an interesting counterpart to simple information-sharing.

The principle of node communication in XLM is *initiative determination* $\mathcal{I}$: after a node indicates it has a packet to send with an RTS, each neighbor decides whether to participate in the communication. The initiative determination sets conditions on link state, flows, buffer fullness, and energy. If all conditions are satisfied, the node participates in the communication; otherwise it goes to sleep until the next determination time. According to the authors, this set of conditions ensures reliability of the link, manages flows and buffer levels, and guarantees uniform energy consumption.

Table 1: Qualitative comparison of existing architectures (given without order of importance): support ($\surd$) or limited support ($\times$).

	Legacy*	SNA	Cham.	X-Talk	MobMan	XLM	X-Lisa
Modularity	$\times$	$\surd$	$\surd$	?	$\surd$	$\times$	$\surd$
Universality	$\times$	$\surd$	$\surd$	$\surd$	$\times$	$\times$	$\surd$
Event Notification	$\times$	$\times$	$\times$	$\times$	$\surd$	$\times$	$\surd$
Service Support	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\surd$
Table Maintenance	$\times$	$\surd$	N/A	?	N/A	N/A	$\surd$
Information Prop.	$\times$	$\times$	N/A	$\times$	$\times$	N/A	$\surd$
Overhead	Small	Small	Small	High	High	Small	Small
Simplicity	Yes	No	Yes	Yes	No	Yes	Yes

Akyildiz et al. argue that the tight meshing of all functionalities guarantees optimality of the network response—or at the very least the coordination of their actions. All protocols fused in XLM benefit from the same up-to-date information, much like the previous approaches. However, XLM supposes the use of CSMA/CA only. The fused nature of the protocol layers renders protocol swapping a complicated process. Unlike MobileMan, CrossTalk, and SNA, there is no standardized interface between the various protocols of the module.

IV. Architecture Comparisons

In this section, we present a qualitative comparison of the architectures described in the previous section. It is expected that the architectures will perform better in some regards, and more poorly in others, in accordance with their different design goals. The aim in this section is to provide a side-by-side comparison of each architecture's relative advantages and drawbacks.

Table 1 presents the strengths and weaknesses of each architecture for a specific set of goals. The term *Legacy** refers to the layered structure inherited from the OSI model, with potential ad-hoc modifications ("violations") to support cross-layering.

IV.A. Flexibility

This section evaluates the *flexibility* of an architecture, and thus its chance of becoming popular. In WSNs, the term flexibility indicates both modularity and universality, which are often contradictory in nature, as modular designs have, so far, denied support for cross-layer protocols.

IV.A.1. Modularity

Migration from strictly layered to cross-layer designs has incurred architecture violations that make swapping protocols an intricate task. Modular designs guarantee that replacing a protocol requires little more than inserting the new protocol and using the appropriate interface to connect the layers. For instance, it would be accomplished in TinyOS via a simple rewiring.

It is agreed that the OSI model adapted to WSNs (the *Legacy** architecture) is the most modular architecture. However, this architecture is unfit for cross-layer protocols, which introduce protocol-dependant violations. The legacy architecture rates poorly in modularity because of the unforeseeable nature of these violations.

Because the goals of SNA and Chameleon (Rime) are to abstract lower layers, modularity may not apply to higher levels in the stack. We believe, however, that SNA is the most modular architecture to date proposed for WSNs. The implementation details of CrossTalk are not provided in [5], and thus, we can only base this evaluation on what CrossTalk sets out to do. CrossTalk does not appear to keep packet information in a message table, which may hinder protocol replacement. Because sharing information between protocols requires writing additional access functions, MobileMan is not as modular as other designs, and to the best of our knowledge, does not support the sharing of neighbor information. Lastly, since XLM focuses only on network performance, replacing a core function in XLM requires knowledge of the full module, limiting the modularity of its design.

IV.A.2. Universality

The term *universality* refers to the ability of the architecture to accommodate all platforms, protocols, and end applications, through code reuse and run-time sharing. The eventual goal of universality is to ease protocol development and upgrade. SNA and Chameleon propose an elegant solution to abstract the underlying hardware from the protocols. CrossTalk offers a good solution for MANETs, however the information propagation model is not suited for WSNs. Additionally, as for MobileMan, there are no provisions to abstract the data link layer, which heavily depends on the platform. Finally, changing a function within XLM requires the knowledge of the full module and its subsequent modification.

IV.A.3. Event Notification

One of the advantages of layer fusion is the ability to coordinate various functions easily: protocols now gathered under the same layer can share information and invoke the same set of functions. A layered design, however, is required to create more complex interfaces between different layers, which contradicts the goal of modularity. For instance, imagine a MAC protocol able to quickly detect broken links and a routing protocol whose route repair mechanisms are slow. A programmer may wish for the MAC protocol to notify the routing protocol of a failed link as early as possible. We argue that an architecture should allow event notification through its shared database, thus allowing to report changes in the network. Of the surveyed architectures, only MobileMan supports event notification.

IV.A.4. Service Support

Rarely mentioned, services provide key support to protocols. For example, services may gather information about the remaining energy or location of a node. Protocols should only focus on their main role, such as routing, without concerning themselves with gathering peripheral information. To the best of our knowledge, no existing architecture proposes organizing services in their stack.

On top of managing services, the architecture should provide a fixed and common interface to standardize access to the services and establish coding conventions. It should also allow individual protocols to selectively load the services before deployment, and to turn them off during runtime.

IV.B. Information Freshness

In this section, we examine how the network view is kept up-to-date, a key feature for many protocols. This feature is critical to having an architecture that is modular and easy to maintain and use.

Most architectures propose to store and manage information made available to part or all of the layers in the stack. SNA (SP) features advanced table maintenance, although more decisions are left for the individual protocols to make. Because we lack implementation details for CrossTalk, we cannot evaluate it in this regard. Chameleon does not provide a common information repository, and thus does not provide a local view of the network. Because MobileMan abstracts information structures, there is no actual common neighbor table, and information freshness and management is handled by the registered protocols. Finally, XLM does not need to make any information structure available to other layers in the stack because it already shares common data within all functions of the module.

In order to maintain relevant information for all the protocols, the (global) view of the network should be propagated regularly. Only CrossTalk proposes disseminating information. However, because the main focus for CrossTalk is MANETs, this propagation model is not fit for WSNs, which they tend to have smaller traffic rates and converging routes.

IV.C. Overhead

Low additional overhead is a critical notion that may qualify an architecture as the most energy efficient, and thus the most appealing. The overhead incurred by an architecture usually depends on the sophistication of the protocols in the stack. However, since MobileMan abstracts a database, redundant structures may still exist at different layers. It follows that MobileMan, like Chameleon, may require more RAM than necessary. The propagation model of CrossTalk can be viewed as expensive: at least 12B must be appended to each packet for X and Y locations and the time stamp. SNA and XLM provide RAM savings by removing redundant information: SNA uses a common database approach, while XLM lets all functions in the module view the same data.

IV.D. Simplicity

A simple architecture has a greater chance at allowing faster release and deployment of WSNs. A good view of the simplicity of an architecture is its modularity, thereby excluding the legacy architecture and XLM. However, beyond this consideration, architectures that do not carry protocol operations (routing, packet reordering, etc.) seem to be the simplest to maintain and understand because they do not have unpredicted behaviors. SNA, because it is supported by the Sensor Protocol, must also be understood as having protocol behavior, which may complicate its use. At the same time, information exchange must still happen because, although protocols have an interest in information, it is generally not their role to fetch and organize such information. In TinyOS, simplicity also means fewer wirings and hidden capabilities.

V. X-Lisa, a New Architecture for Cross-Layer Information Sharing

Armed with a new sense of the strengths and weaknesses of existing architectures for WSNs, we argue that architectures relying on a non-abstracted shared database are both simple and flexible. Among them, SNA seems the most appropriate to WSNs, but it does not populate the neighbor table, and

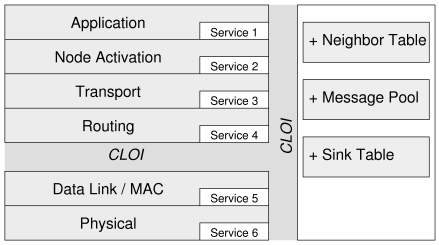


Figure 3: X-Lisa: An information-sharing sensor network architecture for cross-layer optimizations. This architecture retains a layered design while providing flexible information repositories as well as services to support the protocols.

more generally, its goal is to abstract the hardware platform from the protocols. Instead, what is needed is an architecture that provides support for cross-layer protocols using a modular structure.

Thus, we propose X-Lisa, a new Cross-Layer Information-Sharing Architecture that combines simplicity with support for cross-layer interactions, services, information propagation and event notification. In this section, we fully describe our information-sharing architecture, while in the following section we detail the improvements brought by X-Lisa. The TinyOS code for X-Lisa is available to download at *http://www.ece.rochester.edu/research/wcng*.

V.A. A New Unifying Architecture

Figure 3 shows the new protocol architecture defined by X-Lisa. The X-Lisa architecture retains a layered structure such that each layer is matched to a communication function in order to maintain a practical and simple design. While fused layer design is still possible with X-Lisa, it is not favored as it hinders modularity.

The Cross-Layer Optimization Interface (*CLOI*) provided by X-Lisa offers indirect access to a repository of information that may be needed by one or more protocols. *CLOI* maintains this information through three structures, a neighbor table, a sink table and a message pool, described in detail below, and it supports *services* that will fill these data structures either once or continuously, depending on the information. X-Lisa also supports event notifications to ease coordination between various layers.

While all layers and services in the stack have access to the interface, a *CLOI* layer was placed between the routing and MAC layers for two reasons. First, its location allows the interface to retrieve much of the information sent from the node onto the network as well as many incoming packets. This allows *CLOI* to directly obtain information needed to fill the neighbor and sink tables without going through the protocols. The MAC and physical layers do not have a global vision of the network and cannot provide enough information about its state for automated use with *CLOI*. The second reason is that it offers the potential for abstraction of the link layer as suggested in [6].

Finally, *CLOI* has no authority to make any routing, node activation or medium access decisions, or packet reordering. *CLOI* simply acts as an interface to the protocols in the stack, allowing them to access common yet important information about the node and its neighbors that can be used to optimize each protocol's performance.

Table 2: A neighbor table is kept at every node i with non-predetermined fields. It keeps information about the node itself (for vertical cross-layering) and each of its neighbors j (for horizontal cross-layering).

ID	Time Stamp	n byte Array
2B	8B	nB
Id_i	t_i	$B_{i,0}B_{i,1}...B_{i,n-1}$

V.B. Information Sharing Structures

In order to support this information-sharing architecture, we need to determine the best data structures for storing the required information, and the *services* that will populate them. Additionally, today's platforms often utilize TinyOS, a simple programming language for embedded systems, that limits the scope of implementation solutions.

V.B.1. Neighbor Table

Since some protocols require knowledge of differing parameters, the neighbor table is implemented as a flexible information repository. Before runtime, each programmer may elect which parameters populate the table according to the needs of the protocols in the stack. Accordingly, the neighbor table is in fact defined as three fields, illustrated by Table 2: node *ID*, a time stamp for data freshness, and an array of integers. The last structure may be filled with a customizable set of parameters that may or may not have the same type or size. Some of the previous solutions proposed a neighbor table with fixed fields: since NesC does not support dynamic structures, extending the neighbor table included rewriting some of the definitions and functions, and replacing fields one by one.

In X-Lisa, the programmer needs only to declare an enumeration of the fields of interest and their size in bytes. This Key-Length-Value solution allows rapid modification of the neighbor table, without resorting to changing the fields "by

hand". Although similar in spirit to Hood, it differs in its ability to update itself (with permission from- but no direct supervision by- protocols) and in its implementation. Where Hood creates several vectors of a fixed type (`int` for light, `float` for another sensor reading) of size the maximum number of neighbors, *CLOI* keeps a unified neighbor table. It is as if one was the transposed matrix of the other. This allows *CLOI* to retrieve information from a neighbor with only one pass, whereas Hood can return neighbor IDs when searching for information about a variable (which was its main goal).

A neighbor table can have up to 30 entries, which marks a compromise between the number of valuable neighbors a node can have and RAM usage. When the table is almost full, retrieving information takes longer.

Because they are not predetermined, the fields of the neighbor table must be opaque and accessed through a fixed syntax, which provides read / write commands with a value and a field identifier. This process is illustrated by the following example:

```
cost = *(float*) call
            Cloi.extractValue(entry, COST);
```

where `entry` is a neighbor table entry, and `COST` designates a user-defined protocol metric.

V.B.2. Sink Table

Many protocols require critical information about the various sinks in the network to determine equivalencies between them or what data to send to a particular sink. Because we expect information about the sinks to change slowly, the sink table is not automatically updated. Maintenance must be done by the protocols (generally a middleware and the routing protocol). It is also the only structure carrying information more than one-hop away.

V.B.3. Message Pool

Others have proposed using a message pool that includes details about the received and sent messages [6]. We agree with the pertinence to use such a structure and propose incorporating the following fields: packet ID, description (a data packet or network administration packet), priority (urgency) and status (successfully sent or pending).

V.B.4. Accessing the Structures

Read and write access to all structures may only be granted through *CLOI*: this has the combined advantages of atomicity[1] and modularity. Because read and write operations are placed in atomic segments, they are executed in the order they are received, without prioritization. X-Lisa also provides additional functions for access management: because protocols do not know the identity of the neighbors present in the table *a priori*, they may invoke the function `nextEntry`, which returns the *ID* of the next entry in the neighbor table.

V.C. Event Signaling

X-Lisa provides two classes of event notifications: *protocol events* and *CLOI events*. The former designates events generated by protocols in the stack and relayed directly through *CLOI*. The type of the event is not known to *CLOI* but has to be meaningful to both the provider and user of this event. *CLOI events* refer to a set of events defined in X-Lisa and that include notification of a new packet in the pool, a new neighbor, a full neighbor table, etc. In general, *CLOI events* are much more common than *protocol events*[2], and thus a programmer can choose to not use *CLOI* events at compile time if they are not needed.

Protocols that require event signaling can subscribe to *CLOI* at compile time. Because NesC does not allow runtime dynamic wiring, a protocol may not unsubscribe from event notifications. The MeshC [14] language overcomes this limitation. Either way, this does not seem to be a strong constraint as we expect protocols to have an interest in event notification for the duration of the network lifetime. If no longer relevant, event notifications may simply be ignored by subscribing protocols.

V.D. Information Exchange

In order to maintain the information contained in the neighbor table, X-Lisa provides an automatic update service. The information exchange is carried by an *information vector* that updates the neighbors of a node.

V.D.1. Information Vector

The information vector includes some or all of the fields necessary to populate the neighbor table. These fields will automatically be filled by the *CLOI* of the sending node and read by the *CLOI* of the node's neighbors. The information vector may be piggy-backed onto broadcast packets or sent as a stand-alone packet when no broadcast packets are sent for a certain amount of time. This implies that when

[1] Whereby the same segment of code may be accessed by only one element at a time

[2] In our implementation of Section VI, one or two *CLOI* events were generated every time a packet was received or sent.

Table 3: A packet with a *CLOI* information vector piggyback (TOS_Msg fields not included).

ID	Content	Vector	data
2B	1B	nB	xB

the information vector is piggy-backed to packets traveling more than one hop away, the contents of the vector change at every hop.

CLOI retains the principle of abstracted encapsulation that guarantees that X-Lisa components need not be informed of other protocols' data structures and packet headers.

V.D.2. Size of the Information Vector

Not all fields may require a frequent update depending on the application QoS requirements, the needs of the protocols in the stack, or the nature of the field itself. *CLOI* piggybacks only the required parameters to its neighbors. These can be requested through a *CLOI* command for each parameter. The structure of the piggy-back and stand-alone update is illustrated by Table 3. To inform receiving nodes of the content of the information vector, we include a *content* byte in *CLOI* messages, with each bit representing a parameter in a pre-determined order.

CLOI exchanges information of the fields that are requested by at least one protocol through the `exchangeField` function by invoking:

```
Cloi.exchangeField(byte field, bool comm);
```

where `comm` is `FALSE` when the field is not required by the protocol, `TRUE` otherwise.

This Key-Length-Value solution is the first one of its kind for packet exchanges and, together with the structure of the neighbor table, allows changing fields of interest quickly.

V.E. Maintenance of the Neighbor Table

In order to maintain the freshness of the neighbor table, *CLOI* must detect obsolete information and remove the corresponding entry after a node has died or moved away, thanks to a time stamp applied every time data is added. A protocol may block this process to avoid discrepancies in neighboring sensors' tables and directly call for an inspection of outdated entries when it finds convenient.

If the neighbor table reaches near capacity, an entry may be removed even if its information is not stale. However, since some neighbors may be more important than others, protocols may signal so by setting the field *hold*. Neighbors not *held* will be first choices to be removed.

V.F. Important Services

As mentioned previously, one of the advantages of X-Lisa is that it allows protocols to re-focus on their primary functions. To enable this, X-Lisa adds peripheral services that supply some of the information needed to fill *CLOI*'s information repositories. These include the following: ID assignment, location, time synchronization, channel estimation, remaining energy measure. A more detailed description can be found in [15].

These services are available as libraries and thus individual services can be selected during compilation (*i.e.*, only those services that are needed for the particular set of protocols in the stack would be selected). Furthermore, selected services can be turned off during run-time to guarantee the maximum flexibility.

VI. Results and Discussion

In this section, we intend to show the benefits of using X-Lisa through simulation of an existing protocol whose behavior with and without X-Lisa was studied in TOSSIM, the TinyOS simulator. A full account of the evaluation of X-Lisa can be found in [15]. Test in an actual deployment is part of our future work.

VI.A. Modus Operandi

VI.A.1. Qualitative Study

We begin by providing a data-point that illustrates the simplicity and generality of X-Lisa through protocol swapping.

Our starting point was XLM [8], which exhibits total layer fusion and is thus the counter-point of X-Lisa. We wanted to see if X-Lisa was rich enough to replicate the behavior of XLM while maintaining the convenience of separated protocol layers. We illustrate the advantage of keeping a layered scheme by swapping the MAC protocol from the original XLM MAC functions to a Low-Power-Listening (*LPL*) scheme [16].

VI.A.2. Quantitative Study

We then quantify some of the limitations and gains of using X-Lisa such as the extra overhead and increase in quality of service (QoS) induced by X-Lisa. There exists a plethora of protocols for WSNs, many of which could benefit from X-Lisa. We selected DAPR [17], a distributed fused-layer routing and node activation protocol, for our familiarity

with the protocol and because it strikes a good balance between cross-layer (combined routing and node activation) and layered (interface of DAPR with other protocols in the stack) schemes.

We implemented the original version of DAPR as well as a modified version of DAPR that takes advantage of the X-Lisa architecture. In our simulations, we measure the total number of unicast and broadcast packets, which provides an indication of the extra energy required by X-Lisa. We also determine the packet delivery ratio, as well as the number of reports delivered to the data sink: together, these can measure the QoS provided to the end application.

VI.B. Qualitative Study: The Expressiveness of X-Lisa

In this section, we show that X-Lisa is an architecture that is rich enough to mimic the behavior of XLM, a fully-fused scheme. XLM is particularly of interest because it represents an extremum in cross-layer designs, and hence is a candidate of choice to test the limits of X-Lisa.

VI.B.1. Implementation Details

XLM establishes unicast communications through an RTS / CTS handshake before DATA is exchanged and acknowledged. Power is saved through duty-cycling, which turns the radio chip on and off periodically: nodes remain asleep for the remainder of the cycle if they cannot participate in the communication ($\mathcal{I} = 0$) or if they lose contention to another node. Contention is handled through a CTS response backoff proportional to the node's distance to the destination (assumed to be known). The stateless greedy routing is receiver-based, and the node sending the first CTS signals its intention to forward a packet. Congestion control reduces the application packet generation rate in case of communication failure, and increases it otherwise.

In the initial protocol suite, we decomposed XLM into the five layers (including application) of Figure 3 and included them in the X-Lisa architecture. The new entity, called XLM / X-Lisa, is a layered version of XLM. The transport layer now extracts information from user data (and updates some fields in X-Lisa's neighbor table) and segments long data packets. In this suite, the network layer has a limited role. The link layer performs the RTS / CTS / DATA / ACK hand-shake and controls the radio chip for duty-cycle. Among other things, X-Lisa shares packet delivery failure information with the application layer for congestion control purposes and uses the neighbor table as a central storage of node positions for location look up. Figure 4 shows the new organization.

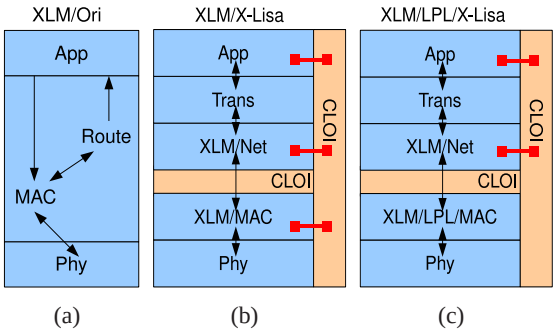


Figure 4: The original XLM (a), was broken into a layered scheme (XLM / X-Lisa) (b), and its MAC layer was replaced (c). Arrows show packet exchanges between layers, and squares information exchange.

We tested XLM / X-Lisa and found that it replicates the behavior of XLM (routing packets to the destination using the same number of packets) while retaining a layered structure.

VI.B.2. Protocol Swapping

During the introduction of this paper, we conjectured that protocol maintenance would be eased by modularity. To test this supposition, we attempted to swap MAC protocols to, for instance, a LPL MAC protocol. Although we cannot quantify the ease with which we did so, a successful MAC replacement is meaningful in itself.

The second protocol suite is a variant of XLM / X-Lisa: the original MAC layer was replaced by the LPL MAC protocol SpeckMAC-D [16] as illustrated in Figure 4. We named the new entity XLM / LPL / X-Lisa. In SpeckMAC-D, every node sleeps for t_i s (the inter-listening time) between wake-ups. In order to guarantee that the receiver will wake up at some point during a transmission, a sender must repeat the same packet for t_i s. If a node awakens and receives a packet, its MAC protocol forwards it to the network layer before sleeping for the rest of the cycle.

Similarly to XLM, we modified XLM / LPL / X-Lisa to route packets only if the node is the closest to the destination, a strategy akin to restricted flooding.

The two suites of protocols were implemented in TinyOS and simulated with TOSSIM. We conducted simulations on 10 nodes spread on a 70 m by 70 m area with nominal radio range 30 m. The Source node sends a packet to the sink every 5 s, for a total simulation time of 100 s. The results are shown in Table 4.

Here, the number of sent packets should not be seen as an indicator of the energy consumed by each suite, we thus do not present it in our results. A CSMA-based MAC proto-

	XLM/LPL/X-LISA	XLM/X-LISA
Received Packets	165	2036
Goodput (%)	75	100
Latency (s)	1.503	0.683

Table 4: Selected metrics comparing the behaviors of XLM / X-LISA and XLM / LPL / X-LISA.

col forces the radio to stay in idle mode (a state that incurs the same energy consumption as active receiving mode) significantly more than a LPL scheme. Consequently, sending more packets with a LPL MAC protocol does not always result in increased energy consumption.

These results show that XLM / X-LISA receives more packets than XLM / LPL / X-Lisa because every communication requires a hand-shake, and because many nodes receive RTS / CTS / DATA / ACK packets even though they lost the contention to other nodes and are not part of the communication. Finally, both suites exhibit high goodput (greater than 75%), with XLM / X-Lisa showing the better performance. Likewise, XLM / X-Lisa yields lower latency. Both metric differences can be explained by the fact that nodes that are candidates to participate in a communication ($\mathcal{I} = 1$) are always on during a cycle, allowing few packets to be dropped or delayed.

These results show that the replacement of the original XLM MAC protocol by SpeckMAC-D led to observably similar behaviors: according to a set of quantifiable metrics, the two suites are within the same order of magnitude. Thus, swapping protocols was possible and X-Lisa did not degrade the performance of the protocol. What these results do *not* demonstrate is that one particular MAC protocol yields a longer lifetime or any other desirable QoS improvements over the other, because a protocol must take advantage of the information brought by X-Lisa.

VI.C. Quantitative Study: Measurable Cross-Layer Improvements

We now show that X-Lisa helps improve the performance of the network by allowing protocols to take advantage of cross-layer interactions to their fullest.

VI.C.1. The DAPR Protocol

DAPR assigns "application costs" to all nodes by periodically flooding a *query* request to the network: the higher the cost, the more important the node is to the application (because it may be fitted with unique sensors or because it is located in a sparsely covered area of the network, etc.) Nodes with high costs are eager to deactivate and make poor choices as relays for other nodes' packets. In this simulation, we divided the whole network into *zones*: nodes in the same zone may communicate with one another, and with nodes from adjacent zones. A routing tree can thus be formed, and nodes with low costs route data packets to a single data sink. Nodes located in a target zone repeatedly send data reports at a rate of $0.2\ pkt.s^{-1}$.

Because DAPR sends queries at the beginning of every $60\ s$ round, changes happening to the tree are unknown to the protocol stack until a new query is flooded, with sometimes serious consequences. For instance, if a relay node moves from one zone to a neighboring zone, data packets will stop being delivered.

X-Lisa brings new information to the protocols it serves: for the case at hand, DAPR can now be notified when a change of zone occurs or if a new neighbor has been added to its table. Such changes usually mean that a node's next-hop neighbor might have changed, and that it may need to start or stop sending data reports. One important metric is the average update delay, defined as the time between a change and its notification to the nodes' neighbors. The longer the delay, the longer nodes affected by a change fail to take appropriate corrective measures.

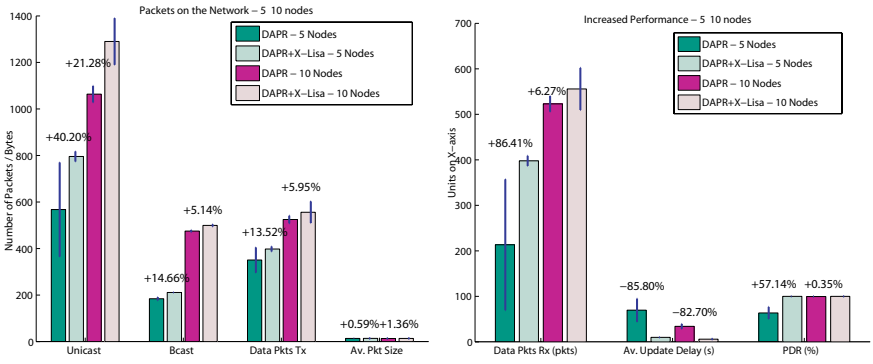
Without the added knowledge provided by X-Lisa, DAPR must include extra information (such as a node's current zone) along with queries, regardless of whether a change has indeed happened.

VI.C.2. Simulation Results

Since the goal of this section is not to evaluate DAPR, we limit the simulation to a simple scenario: a likely candidate to route other nodes' packets is mobile and may move around (on average, every $150\ s$). We use a small number of nodes (5 and 10) so that we may easily interpret the behavior of the network. Had we chosen several mobile nodes, we would have tested the resilience of DAPR, rather than the benefits brought by X-Lisa.

We implemented DAPR with and without X-Lisa in TinyOS. While we ran simulations in TOSSIM, we also compiled the code for the Tmote Sky platform. DAPR alone takes up approximately $21.5\ KB$ of ROM, and $1.3\ KB$ of RAM. With X-Lisa (and all its features), these numbers become $34\ KB$ and $1.9\ KB$ respectively, which can easily be accommodated by the Tmote Sky.

Figure 5(a) presents the number of unicast and broadcast packets for DAPR alone and DAPR with X-Lisa. Also shown are the number of report packets that were sent and the average packet size. For both 5 and 10 nodes, the number of unicast packets increases when X-Lisa is used. Since updates are broadcast packets only, we know that it does not



(a) (b)

Figure 5: Comparison of (a) the impact and (b) the QoS gains of DAPR and DAPR + X-Lisa on a network of 5 and 10 nodes. The numbers on the graph are the relative change from DAPR to DAPR + X-Lisa.

constitute overhead for X-Lisa, but merely increased data traffic brought about by improved performance of DAPR with X-Lisa. The increase in broadcast packets remains modest (a maximum of 15%) even for 10 nodes and measures the added cost of X-Lisa in the face of frequent topology changes. Quantifying the energy impact of this increase depends greatly on the MAC protocol in use, with heavier consequences for LPL MAC protocols. The number of data reports sent to the data sink differs only slightly: when a relay node moves to and from the target zone, X-Lisa can notify its application layer to start or stop sending reports. In the absence of X-Lisa, the application layer does not learn of a change until the next DAPR round, which causes the node to send too many or too few data reports. Over the whole node lifetime, however, these tend to average out. Finally, the average packet size stays approximately the same: the information vector sent by X-Lisa does not need to be exchanged if no movement between zones has been recorded. This compensates for larger packets when the full information vector is present in all packets.

Figure 5(b) measures the advantages provided by X-Lisa: a net decrease in the average update delay, which translates into significant gains of QoS. Neighbors of a moving node are notified of a change up to 85% faster when X-Lisa is used. Protocols may take advantage of this in several ways. With our implementation of DAPR, this translates into a steep increase in the number of packets delivered to the data sink, as well as an increase in packet delivery ratio. The magnitude of this increase depends on conditions in the network (when a change happens, how many alternate routes there are, etc.) and on the protocols in the stack; however, the improvement can be significant.

A discussion on the overhead of X-Lisa (up to 15% more total broadcast packets) must include the fact that X-Lisa regroups and limits the size of neighbor update messages and that it is conducive to a significant increase in QoS. Comparing to the original DAPR design, much of the overhead produced becomes stale after changes in the network topology. These results show that while X-Lisa provides more flexibility and generality, it does not degrade protocol performance. Better yet, correct use of extra information helps increase the QoS to the application, with X-Lisa incurring only small overhead.

VII. Conclusions and Future Work

In this paper, we surveyed the state of the art of information-sharing architectures whose merits include support for cross-layer interactions while exhibiting high modularity. We compared these various architectures and found that none of them provide all the requirements needed for sensor networks: support for 1. cross-layer protocols using a modular architecture, 2. services, 3. information exchange suited to sensor network traffic models, and 4. event notification.

Thus, we proposed X-Lisa, an information-sharing architecture that facilitates vertical and horizontal cross-layer optimizations in WSNs through a cross-layer optimization interface (*CLOI*). *CLOI* maintains updated information on the network state, the nodes' states, the data sinks and the messages to be sent. All layers have access to the information maintained by *CLOI*, which ensures that all protocols in the stack can benefit from cross-layer optimizations facilitated through information-sharing.

In our Tmote Sky platform implementation, we have verified that existing protocols (such as XLM [8] and DAPR [18]) can fit into this information-sharing architecture. However, the real advantage of this architecture is that it will facilitate the design of future protocols by removing the burden of finding, maintaining and sharing important

node and network information from the protocols and placing this task within *CLOI*. In spite of some limitations, the ease of use and the implementation freedom of X-Lisa make it a viable option for future sensor network deployments.

Our future work will focus on reducing the code size of X-Lisa and on continuing to quantitatively evaluate the benefits and drawbacks of X-Lisa, including on actual deployments. We also plan to add more services to the libraries, and application requirement information to X-Lisa in an integrated fashion with *CLOI*. Providing the protocol stack with application-level QoS requirements will make X-Lisa highly adaptable to different application requirements.

References

[1] V. Srivastava and M. Motani, "Cross-layer design: A survey and the road ahead," in *IEEE Communications Magazine*, vol. 43, no. 12, Dec. 2005, pp. 112–119.

[2] V. Kawadia and P. Kumar, "A cautionary perspective on cross-layer design," in *Proc. Wireless Communications*, Feb. 2005.

[3] C. J. Merlin and W. B. Heinzelman, "Cross-layer gains for sensor networks," in *Proc. DCOSS'06, Poster Session*, Jun. 2006.

[4] M. Conti, G. Maselli, G. Turi, and S. Giordano, "Cross-layering in mobile ad hoc network design," in *IEEE Computer*, Feb. 2004, pp. 48–51.

[5] R. Winter, J. H. Schiller, N. Nikaein, and C. Bonnet, "Crosstalk: Cross-layer decision support based on global knowledge," in *IEEE Communications Magazine*, Jan. 2006.

[6] 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 *Proc. OSDI'06*, Nov. 2006.

[7] A. Dunkels, F. Osterlind, and Z. He, "An adaptive communication architecture for wireless sensor networks," in *Proc. SenSys'07*, Nov. 2007.

[8] I. F. Akyildiz, M. C. Vuran, and Ö. B. Akan, "A cross-layer protocol for wireless sensor networks," in *Proc. CISS'06*, Mar. 2006, pp. 22–24.

[9] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler, "Hood: A neighborhood abstraction for sensor networks," in *Proc. MobiSYS'04*, Jun. 2004.

[10] Q. Wang and M. A. Abu-Rgheff, "Cross-layer signalling for next-generation wireless systems," in *Proc. IEEE WCNC'03*, vol. 2, Mar. 2003.

[11] C. M. Sadler, L. Kant, and W. Chen, "Cross-layer self-healing mechanisms in wireless networks," in *Proceedings* 6^{th} *World Wireless Congress (WWC'05)*, May 2005.

[12] H. Zimmermann, "Osi reference model-the iso model of architecture for open systems interconnection," in *IEEE Transactions on Communications*, vol. 28, no. 4, Apr. 1980, pp. 425–432.

[13] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica, "A unifying link abstraction for wireless sensor networks," in *Proc. SenSys'05*, Nov. 2005.

[14] A. Belenki, Product Director, Luxoft Labs, "Overcoming challenges of TinyOS use in commercial zigbee applications," in *TinyOS Technology Exchange III*, Feb. 2006.

[15] C. J. Merlin and W. B. Heinzelman, "X-Lisa: A cross-layer information-sharing architecture for wireless sensor networks," http://www.ece.rochester.edu/∼merlin/Xlisa/X-lisa_URTR.pdf, Dec. 2007.

[16] K.-J. Wong and D. Arvind, "Speckmac: Low-power decentralised mac protocol low data rate transmissions in specknets," in *Proc.* 2^{nd} *IEEE Int. Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality (REALMAN'06)*, May 2006.

[17] C. J. Merlin and W. B. Heinzelman, "Sensor network middleware for managing a cross-layer architecture," in *Proc. DCOSS'06 - EAWMS Workshop*, Jun. 2006.

[18] M. Perillo and W. Heinzelman, "DAPR: A protocol for wireless sensor networks utilizing an application-based routing cost," in *Proc. WCNC'04)*, 2004.