System Support for Cross-layering in Sensor Network Stack

Rajnish Kumar[†], Santashil PalChaudhuri[‡], Umakishore Ramachandran[†]

[‡] Department of Computer Science, Rice University, Houston, TX,
[†] College of Computing, Georgia Institute of Technology, Atlanta, GA.

Abstract. Wireless Sensor Networks are deployed in demanding environments, where application requirements as well as network conditions may change dynamically. Thus the protocol stack in each node of the sensor network has to be able to adapt to these changing conditions. Historically, protocol stacks have been designed with strict layering and strong interface between the layers leading to a robust design. However, cross-layer information sharing could help the protocol modules to make informed decisions and adapt to changing environmental conditions. There have been ad hoc approaches to facilitating cross-layer cooperation for adaptability. However, there has been no concerted effort at providing a uniform framework for cross-layer adaptability that preserves the modularity of a conventional protocol stack. This paper presents a novel service, information exchange service (IES), as a framework for cross-module information exchange. IES is a centrally controlled bulletin-board where different modules can post available data, or request for useful information, and get notified when the information becomes available. IES is integrated into the proposed *SensorStack* architecture that preserves the benefits of layering while facilitating adaptability. IES has been implemented in TinyOS and Linux, to show both the feasibility of the design as well as demonstrate the utility of cross-layering to increase application longevity.

1 Introduction

The explosive growth of the Internet has been spurred to a great extent by the modularity of the network protocol stack influenced by the OSI model. Adherence to the strict interfaces in the different layers, has enabled the independent development of robust protocols and their validation. While the focus on *modularity* (in the OSI model) has been a useful design guideline for Internet protocols, it is becoming clear that the decisions taken at runtime in the different layers could be better optimized with cross layer information. This is particularly true in dynamic settings when the network conditions can change quite dramatically. For example, researchers have shown the utility of explicit congestion notification from the routers to the transport layer [16], and link status information to the IP layer in a wireless setting [20].

While modularity is a key to protocol development and deployment, *adaptability* is emerging as a key determinant of performance, especially in a wireless setting. The design decisions in the protocol stack have to adapt to changing network conditions to maintain high performance. Such adaptability would be facilitated by the use of information available in different layers. Wireless Sensor Networks (WSN) amplify the

need for sharing cross-layer information even further. In addition to the vagaries of the wireless network itself, the inherent resource constrained nature of the nodes pose additional challenges for the protocol stack. Nodes may join or leave the network to save their individual battery power, or environment conditions may vary, thus resulting in dynamic changes to the network topology. To allow for adaptability in the face of such dynamism, many WSN protocols have proposed piecemeal use of cross-layer information. For example, information from link layer may be used by the routing layer, and routing table information may be used by the application layer. However, it is difficult to foresee all the adaptation needs. Hence it is a challenge to standardize protocol interfaces that expose all useful cross-layer information. Optimizing energy, the single most important resource for WSN nodes, requires a holistic view of the stack instead of a layer-specific view available with such piecemeal solutions.

It is interesting to note that in spite of the increasing importance of cross-layering, it is still viewed with skepticism by the system community [12]. There are good reasons for this skepticism. Without careful system support, cross-layering may result in minimal benefits, may be misused, and may lead to unintended problems in the long run. There are three main reasons that point to the need for a careful design of cross-layering. First, without standard interfaces for information sharing, cross-layering could lead to inefficiencies. Often different modules may collect the same information to adapt their behavior, leading to wastage of computation, memory, and energy resources. For example, neighborhood information is useful for both network level routing and application level role assignment; hence uncoordinated information gathering will result in significant resource wastage (see Table I). Second, piecemeal evolution of cross-layering would lead to a spaghetti design of the protocol stack that is hard to maintain and verify due to the complex interactions among the different modules. Third, without a holistic approach to information sharing and event notification different protocol modules may make sub-optimal decisions leading to poor adaptability. For example, unless the application layer is notified of a sudden change in a link quality by the network layer, its role assignment decisions will be sub-optimal thus affecting application longevity.

The question being addressed in this paper is the following: How can we facilitate holistic adaptability without losing modularity? The main issue boils down to overcoming the inherent tension between adaptability and modularity: adaptability needs cross-layer information that seems difficult to obtain without affecting modularity. In other words, how can we structure cross-layer information sharing that does not compromise the robustness and maintainability of the protocol stack? This problem can be solved by decoupling the adaptability needs (that are cross-layer data oriented) from the modularity needs (that are functionality oriented). We use this intuition of decoupling cross-layer data from functionality to achieve an adaptable and modular protocol stack called *SensorStack*. At the heart of this stack is a novel *Information Exchange Service (IES)* that is available to all the layers. Through a publish/subscribe interface, IES provides a predicate-based event notification service that can be used by the protocol modules for information sharing and for making adaptive decisions. By absorbing the onus of managing the cross-layer data for adaptability, IES allows the protocol modules to focus on the functionalities to preserve modularity.

We have implemented IES in TinyOS [8], and assembled a representative SensorStack using heterogeneous sensor network (HSN) routing layer from shareware [10]

2

and an application level data fusion layer called DFuse [13]. Through the implementation and evaluation we demonstrate the utility of SensorStack with IES both qualitatively and quantitatively. First, there is a qualitative benefit in that the component diagram of SensorStack with IES is simpler, with less interaction among the protocol modules for accessing cross-layer data. From a software engineering perspective, this design lends itself to maintainability and robustness of the protocol stack. Second, we show through micro-measurements that the code-path overhead of using IES to access cross-layer information is minimal. Third, we show that resource wastage (network, memory, and CPU) is minimized by aggregating the collection of neighborhood information that is shared by all the layers via IES.

This paper highlights several contributions:

1. By decoupling cross-layer information gathering and sharing from layer functionality, we facilitate adaptability without sacrificing modularity. The design and evaluation of IES is the primary contribution. There are two main nuggets in the design of IES:
   - *Data management module* provides a declarative publish/subscribe interface for protocols to share information facilitating a modular design. Further, it takes care of efficient use of the available limited node memory for information representation, eviction, and access.
   - *Event management module* provides a condition-based event notification mechanism to alert protocol modules of any changes in the environment thus facilitating adaptability.
2. Representative implementations of SensorStack with IES on TinyOS and Linux showing feasibility of the IES design to promote modularity and adaptability.
3. A simple taxonomy for cross-layer information sharing that provides transparency without affecting modularity.

The rest of the paper is organized as follows. Section 2 proposes a taxonomy for sharable information in the SensorStack. IES design is presented in Section 3. The implementation and evaluation of IES are presented in Sections 4 and 5, respectively. Related work is discussed in Section 6. Section 7 concludes the paper with summary and future work.

2   Organization and Information Taxonomy

It is clear from the dynamic nature of WSN environment that decisions in the different layers of the protocol architecture can benefit from cross-layer information sharing. To this end, we first identify the different cross-layer information. Table 1 presents a snapshot of such information commensurate with the functionality provided by a particular layer. For example, the link layer (such as SP [15]) uses the physical condition of the environment as input to produce "link status" information as output that may be useful to other modules. This table is not meant to be exhaustive, but simply serves as a boiler plate for the taxonomy to be presented in this section.

One way to facilitate efficient decision making in each layer is to query the other layers for relevant information. Direct querying of peer modules, however, will result in breaking the modularity of the protocol architecture and lead to an unstructured and

Table 1. Cross-layer Information Produced by Different Protocol Layers

Protocol Layer	Sample Implementations	Produced information	Consumed information
Application	DFuse [13], Surge, TAG [14]	Resource requirement, Sensed data, Transmission schedule	Resource availability, Neighborhood, Topology
Routing	Directed diffusion [9], GPSR [11], SPEED [7], TAG tree routing	Routing metric values, topology information	Neighborhood, Application requirement
Medium access control, Duty cycle control	SMAC [19], Z-MAC, T-MAC [17], ASCENT [1], SPAN [2]	Duty cycle, Neighborhood information	Application requirement, Link information
Link layer	SP (sensornetwork protocol) [15]	Link status	Physical condition

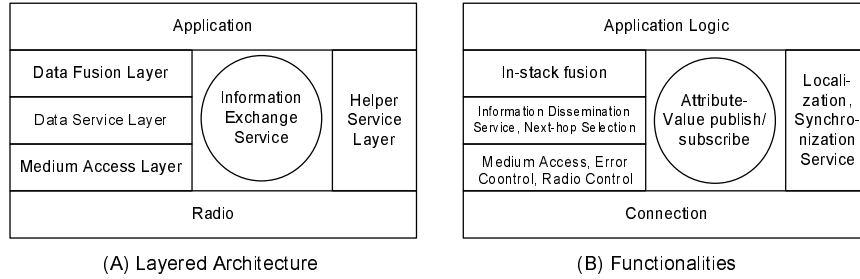(A) Layered Architecture   (B) Functionalities

Fig. 1. SensorStack: A proposed WSN stack.

hard to maintain code base. The fundamental challenge is in developing a layered software architecture that preserves the modularity while allowing cross-layer information sharing. This raises several important research issues:

1. *Organization:* How do we organize the layered software architecture? One promising approach is to decouple the data needed for such information exchange from the functionality of the layered architecture.
2. *Taxonomy:* How do we develop a useful taxonomy for the kinds of information that will be needed by the different layers?
3. *Information Sharing:* How do we facilitate information sharing across the layers that is efficient and non-intrusive on the functionality provided by each layer?

Organization and Information Sharing We propose a layered software architecture, called *SensorStack* (see Figure 1). At the heart of the SensorStack is *Information Exchange Service (IES)* that serves as an information broker among the different modules of the layered architecture and the application to facilitate cross-layer optimizations. Our approach is to decouple the data needed for such information exchange from the functionality of the stack. To this end, we first identify the different cross-layer data and

develop a taxonomy for grouping them. Table 1 presents a snapshot of such data commensurate with the functionality provided by a particular layer. For example, the link layer (such as SP) uses the physical condition of the environment as input to produce "link status" information as output that may be useful to other modules. This table is not meant to be exhaustive, but simply serves as a boiler plate for the taxonomy being presented in this section.

Taxonomy For the purpose of extensibility and documentation, we represent the attributes in the taxonomy in XML format. Clearly, it will be too inefficient to access information across layers by parsing the XML representation of each attribute. Rather, every attribute in the taxonomy is given a unique identifier known to all the layers, and the identifier is used to refer an attribute, thus avoiding the need of the XML parsing. We discuss the assignment of unique identifiers in Section 3.

The information produced and consumed by each layer to facilitate cross-layering can be grouped into four broad categories: *local resources*, *neighborhood*, *application requirements*, and *wildcard*.

1. *Local resources*: The application layer working in concert with the system monitoring module may produce information about the available node resources. Important resources to identify include details regarding available energy, CPU, memory, radio, and sensors.
2. *Application requirements*: An application may produce information that would be of use in the decision making at the routing and MAC layers.
3. *Neighborhood*: For scalability and load balancing reasons, WSN protocols take many decisions locally, and information about neighboring nodes play a very important role. Link layer protocols can produce link qualities of the neighboring nodes. Routing layer can collect routing metric based information, e.g., energy, location, and availability. Using this information, a link layer protocol can use the *timeOn* and the *timeOff* fields to minimize idle listening. The *listen* attribute can be used to inform the link layer to expect transmission from a neighbor, and it can be used for bi-directional low power communication [15].
4. *Wildcard*: There may be other information produced by a particular protocol layer that may not fall into the categories we have identified so far. Examples include abstract region specification for node cooperation [18], *area* abstraction in SPEED for multicast groups [7], *path* abstraction for energy-aware routing, and *role* abstraction for load balancing [13, 6]. We group them as *wildcard* in our taxonomy.

3 Information Exchange Service Design

IES is an information repository for data that helps in cross-layer optimization. Such data may come from one of the modules of the SensorStack or even from the application itself. The taxonomy presented in Section 2 allows grouping the data into different categories irrespective of where it came from and enables easy access by a requesting module. Also, by centralizing all the information in this repository, SensorStack exercises control over access/update rights in a centralized manner.

Design Goals

1. *Efficient use of limited memory*: Memory is a scarce resource in embedded devices, therefore it has to be used prudently. It is quite easy to populate the repository with any and all information that may be useful for cross-layer optimization. However, only a fraction of this information may actually get used by the layers for adaptability. Therefore, IES must filter out unnecessary information and allow the protocol modules to share the memory efficiently.

2. *Simple interface for information sharing*: To ensure that the SensorStack remains modular, cross-module information sharing for adaptability should not lead to coupling of functionalities across layers. Towards this requirement, modules should be able to share data without concerns of synchronization and consistency. Further, the information access should be transparent to producers and consumers (i.e., producers do not know who the consumers are and vice versa). Therefore, IES should provide a simple interface that allows the modules to be implemented independently and efficiently.

3. *Extensibility*: IES should facilitate new information that is outside its repertoire of taxonomy to be added without any change in either the interface or in the underlying architecture. For example, if a new routing protocol is added to the stack, it should be able to publish any new metric into IES that may not be currently in the taxonomy.

4. *Asynchronous access to information*: Producers and consumers of data should not be burdened with unnecessary work. This goal translates to IES providing an asynchronous interface for information from publishers to be *pulled* into the repository, or information to consumers to be *pushed* from the repository, obviating the need for polling on the part of the producers and consumers.

5. *Complex event notification*: To make SensorStack adaptable, protocol modules should be notified of changes reactively. This goal translates to protocol modules being able to register events of interest (which may be a composite of several attributes) with the IES, and receive asynchronous notification when the condition becomes true.

The main objective of IES is to ensure that the SensorStack remains modular (goals 1-3) while supporting adaptability (goals 4 and 5). Access control, security, and protection are also important for IES, but they are outside the scope of this paper.

IES Architecture

As shown in Figure 2, IES comprises of two main components: *Data Management Module (DMM)* and *Event Management Module (EMM)*. DMM is responsible for helping achieve modularity, while EMM is responsible for helping achieve adaptability. DMM is designed as a shared memory abstraction augmented with a fully-associative cache for efficient access; it offers a publish-subscribe interface for sharing information across layers. EMM is designed as a rule-based event notification engine such that protocol modules can be notified as requested, allowing them to adapt to the changes in the environment.
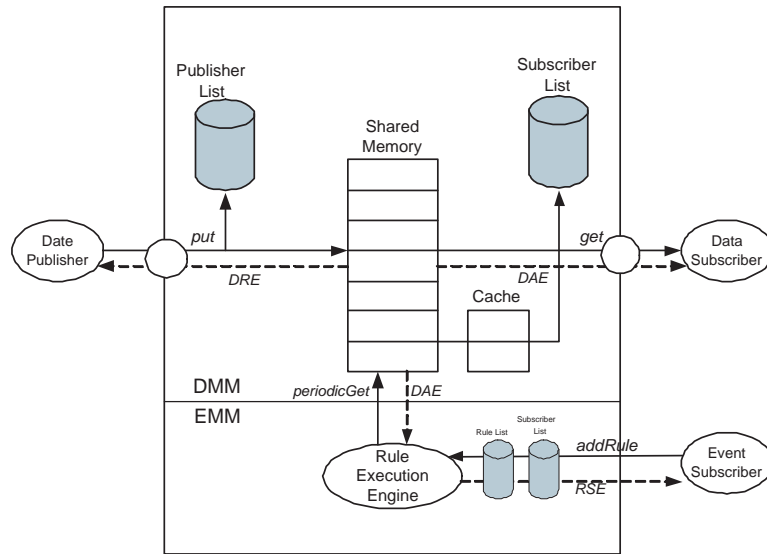
Fig. 2. IES architecture. Top half of the diagram shows the Data Management Module (DMM), while the bottom half shows the Event Management Module (EMM). Note that EMM acts as a subscriber to DMM component.

IES API consists of interfaces to publish, subscribe, and notify of any changes. DMM controls access to the data repository, and thus provides the *publisher* and *subscriber* interfaces. DMM maintains the publisher and subscriber list to support asynchronous exchange of information, especially to support the periodic get method. EMM is responsible for rule registration, execution, and notification to the subscribers. It provides a *watchdog* interface. Based upon the periodicity requirements of registered rules, EMM accesses the data published in DMM component using *periodicGet* call.

Below we elaborate on the design elements of IES that match the five goals identified in Section 3.

Efficient use of limited memory There are two aspects to efficiency in this context: firstly, prudent use of limited memory; secondly, fast access to the stored information. IES uses a block of pre-allocated memory as the information repository. The size of pre-allocation depends on the availability; however, in general it is the case that the amount of information that needs to be stored far exceeds the size. IES uses an LRU eviction policy when information has to be retired from it. There is a possibility that data may be retired from the memory before anyone requests it. For this reason, IES allows the producer to tag the data with a *sticky bit* to over-ride the LRU policy. Alternatively, IES also has the ability to asynchronously "pull" the data from a producer upon a request from a consumer.

For fast access to common data, IES uses a small fully-associative cache to keep the frequently requested data. Motivation behind using the cache is that if some information is requested by one module, it will likely be requested soon by other modules as well.

This is especially true in SensorStack because different modules cooperate to achieve some common goal, e.g., energy optimization and hence may be querying some common attribute from IES (such as application's data requirement or the remaining battery level).

Simple interface for information sharing IES provides a publish/subscribe interface to the shared memory for transparent sharing of information. Publishers can *put* information in standard data format, and subscribers can *get* the same without knowing the publishers. Since information is stored as attribute-value pairs, multiple publishers can publish the same information with different attributes.

Protocol adaptation depends on the information provided by IES. Therefore, it is essential to ensure the freshness of data provided by IES. Producers need to know how frequently they need to update information published by them; consumers need to know if the information they are getting from IES is fresh. Asynchronous access (to be described shortly) deals with the former, while the latter is dealt with by the producers tagging information with an "expiration date".

Extensibility Extensibility is achieved by using standard interfaces and data formats. IES is accessed using get/put over an *attribute_id*. *get* copies the value (if available) and returns the number of bytes corresponding to the data value; a return value of zero indicates that the data is currently unavailable. *put* writes the value into IES, and returns success/failure of the write operation as a boolean value.

```
int get( int attribute_id, byte[] value );
bool put( int attribute_id, byte[] value, int size);
```

Every *attribute_id* maps to a unique attribute description, an XML-based declarative description of the attribute. The attribute description corresponds to a unique entry in a standard ontology of information pertinent to the WSN.

Given a declaration, the *attribute_id* can be obtained by contacting an attribute name server. The idea of attribute name server is similar to a DNS lookup for an IP address. However, discussion of the name server design is outside the scope of this paper.

Asynchronous access With asynchronous access to IES through the publish/subscribe interface, there are four possibilities for information sharing between publishers (P) and subscribers (S) under the arbitration of IES: *push-push*, *push-pull*, *pull-push*, and *pull-pull*.

Push-Push choice yields the best result from the point of freshness of information but it has two downsides: There is a potential for wasted effort if there are no subscribers to published data that is being frequently updated. There is a potential for duplication of effort if multiple modules are publishing the same information. This may be a preferred choice for sharing neighborhood information that is prone to change quite frequently. There are similar pros and cons for the other three choices: *Push-Pull*, *pull-push*, and *pull-pull*.

None of the above design choices serves best for exchange of cross-layer information; rather, different attributes may be best shared in different ways. For example, battery information may need to be shared in a reactive manner, while neighborhood

information may be shared in a proactive manner. This observation motivated us to explore how to support all of the above design choices with a simple interface. While *proactive* communication can be handled by simple *get* and *put* methods, we added *event based signaling* in IES to support *reactive* communication.

A subscriber can request for reactive access to data either by setting up a periodicity in the *get* call, i.e., the subscriber gets data periodically, or by using complex event notification service, where the subscriber gets notified whenever a specific condition is met. For supporting periodic update, a *get* call expects *periodicity*, and a *put* call expects *expiration* as extra parameters. IES uses two events for signaling an update: *Data Request Event (DRE)* to request a publisher to *put* data when data is either expired or unavailable in IES, and *Data Available Event (DAE)* to notify a subscriber of an available update.
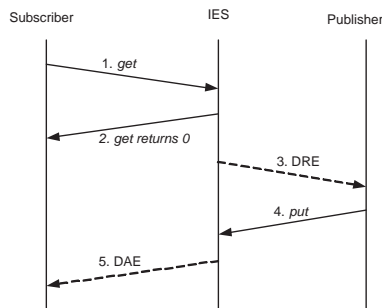
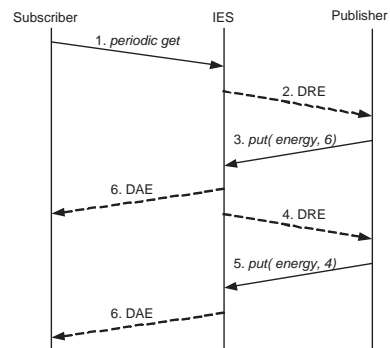**Fig. 3.** Use of asynchronous signaling in IES when requested attribute is not available in IES.

Fig. 4. Use of asynchronous signaling in IES for handling periodic updates.

Figure 3 shows the use of asynchronous signaling to handle a failed *get* request because the requested attribute is not available in IES memory. This may happen because either none of the publishers *put* the attribute or the attribute was evicted, possibly expired, from the IES memory. IES selects a publisher (if any) for the requested attribute, and it raises DRE for that publisher. Once the publisher puts the attribute, IES notifies the waiting subscriber using DAE with a data pointer. The subscriber gets the data from IES. However, it may happen that before the subscriber handles the DAE event, the attribute gets evicted from IES, making the DAE void. To avoid an attribute from getting evicted before DAE is handled, IES keeps a time window before which the attribute is not evicted. A subscriber is expected to handle DAE within the time window, or else the subscriber must issue a fresh *get* call.

Figure 4 shows the use of asynchronous signaling to handle periodic update request. IES periodically checks if the requested attribute has expired or is unavailable in the repository; it then signals the publishers with a DRE. IES maintains the periodicity by using multiple timers. Of course, because of the asynchronous nature, the periodicity

cannot be guaranteed accurately; it may depend on how fast the publishers are able to handle DREs.

Complex event notification Often a protocol module may need to adapt its behavior when certain conditions are satisfied: changes in the environment, resource availability, and/or application requirements. Such adaptability to dynamic changes is quite common in wireless protocol stacks, and this goal is aimed at helping protocol modules monitor these changes in a fast and efficient manner.

IES uses predicate based rule representation to capture complex conditions. A rule takes the form of 'if *condition* do *notify* module P'. Conditions are well formed formulae over the IES attributes. For example, a simple rule can be 'if *(energy < 5)* do *notify* routing module'. IES keeps checking if the specified condition is satisfied, and when satisfied, it notifies the respective subscribers with *rule satisfied event (RSE)*. The two important design questions in this context are: how frequently should IES check for rule satisfaction, and how should IES handle the case when the condition attributes are not currently available in IES memory?

There is a trade-off between the promptness of event notification and incurred computation cost. Owing to the resource constrained nature of sensor devices, IES checks for condition satisfaction only periodically. IES uses the frequency of access/updates to the attributes to fine tune this periodicity. In case an attribute is unavailable at the time of checking, IES signals the publishers for the required attribute data.

4   Implementation

This section describes IES implementation in TinyOS. We have implemented all three interfaces, *Publisher, Subscriber,* and *Watchdog*. TinyOS provides support for asynchronous communication among the components, which is very useful in implementing the event notification service. However, the static nature of TinyOS makes memory management restrictive, and event notification inefficient.

TinyOS is a component based operating system designed for concurrent operations and resource constrained embedded devices. Components provide interfaces to be used by others. An application is written as a set of components wired together using the interfaces and events. Though TinyOS itself provides only basic send and receive interface support over CSMA based radio control, the other layers (such as routing and fusion) are implemented as independent modules. The modules are statically wired together through their component interfaces to realize the network protocol stack.

Data management module Since TinyOS is designed for resource constrained devices, e.g., Mica2 with 4 KBytes of RAM, it uses static memory optimization techniques to generate memory efficient codes. Because TinyOS does not support dynamic memory allocation, we allocate statically a chunk of memory to be used by IES, and use priority based eviction to control its usage.

Every IES entry is of fixed length, that helps an easy and efficient implementation of DMM even without any dynamic memory support. However, this restriction limits the flexibility of *get* and *put* methods: the attribute value must be of fixed size, which
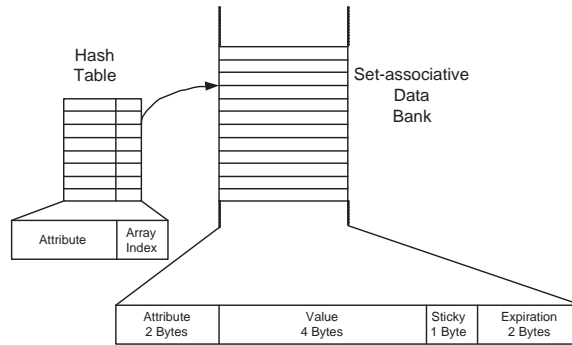
Fig. 5. DMM memory hierarchy. Direct-mapped cache maps an attribute to its location in the data bank.

is 4 Bytes in our case. Figure 5 depicts the DMM implementation. An IES entry is of 9 Bytes length, with 2 Bytes for attribute, 2 Bytes for expiration time, and one Byte for maintaining sticky bits. *Sticky* field value is used to influence memory eviction policy.

DMM is implemented as a two-level cache: first, a direct-mapped cache to keep frequently used attributes, and second, a set associative cache to keep more attributes which we call the data bank. The first-level direct-mapped cache maps an attributeID to a unique index in the second-level cache. The data bank stores a list of attribute-value pairs. So, if the attributeID is available in the direct-mapped cache, then the corresponding index value is used to get the attribute value from the data bank. If the attributeID is not present in the first-level cache, then the data bank needs to be searched. By keeping the data bank set associative, the search space is reduced to the associativity factor. As an example, for a memory mapped cache of 8 entries, and a 16-way set associative data bank of total 256 entries (16 sets), each direct-mapped cache entry is of 24 bits (16 bit attributeID and 8 bit data bank array index). For a hit in the direct-mapped cache (we call a cache hit), an attribute is obtained in 2 accesses (one to the direct-mapped cache, and another to the data bank). For a miss in the direct-mapped cache (we call cache miss), an attribute is obtained in at most 17 accesses (one to the direct-mapped cache, and at most 16 to the data bank as there are 16 entries per set). In case of a miss in the data bank, asynchronous signalling is used to notify a producer (see Section 3).

Event management module EMM implementation supports comparison based conditional rules. A module interested in being notified registers itself with the *Watchdog* interface. EMM, in turn, can register itself as a DMM subscriber for the attribute in the specified rule, and it can then periodically check the rule. Currently, periodic checking of rules is not implemented; rather, the checking is done whenever relevant attributes are updated through a *put* command.

Another source of inefficiency comes from TinyOS limitations. A TinyOS application can be thought of as a set of modules, whose dependency graph needs to be specified statically at compile time. Because of this static nature, event subscription also becomes static. Thus all event subscriptions need to be encoded at compile time itself. In our implementation, we facilitate dynamic rule addition by a simple trade-off:
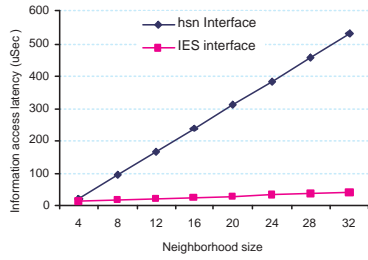
Fig. 6. Comparing information access latency using HSN's neighbors interface and using IES interface. In IES case, the neighborhood data is being accessed directly from the DMM cache.
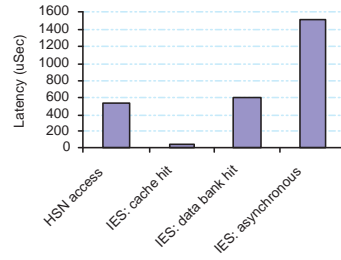

Fig. 7. Memory access overhead comparison for DMM in TinyOS. For IES case, 32-way set associative data bank is used.

we allow an event notification to be triggered when any one of a *set* of rules are satisfied. Thus a rule can be dynamically added to a rule set, but the rule satisfaction is notified to all the modules registered for *any* rule in that set. A rule satisfaction event has a rule identifier field, which can be used by the subscribers to filter the notifications of interest to them.

5 Evaluation

This section evaluates the effectiveness of IES in supporting cross-layering in SensorStack. First, through an extensive set micromeasurements, we investigate the overhead of data access through the IES interface for different scenarios, and compare them with the case where data is accessed directly through protocol modules' interfaces. We also measure the overhead incurred in checking rules in EMM. Finally, we evaluate a complete protocol stack to quantify benefits of using IES, specifically in terms of application longevity and communication savings in data collection from neighboring nodes. Below, because of the lack of space, we present only the micromeasurements.

We use the *SysTime* interface of TinyOS for timing measurements on Mica2 platform. *SysTime* provides timer values at 1.2 micro seconds granularity. The direct-mapped cache size is fixed to 32 entries, and the data bank size is fixed to 256 entries. Set associativity of the data bank is varied from 8 to 64. Each data point is an average over 100 readings.

Figure 6 shows the memory access latency values when all the attributes are present in the direct-mapped cache. It also presents the latency values when the same information is accessed directly by invoking *neighbors* interface using the HSN routing component. As expected for this ideal case, IES memory access is much faster than access using the HSN routing module. IES allows only 4 Byte values for an attribute. As expected, the latency increases linearly with the increase in the number of attributes.

Figure 8(A) shows the case when there is a miss from direct-mapped cache, and Figure 8(B) shows the case when there is miss even from the data bank. As the asso-
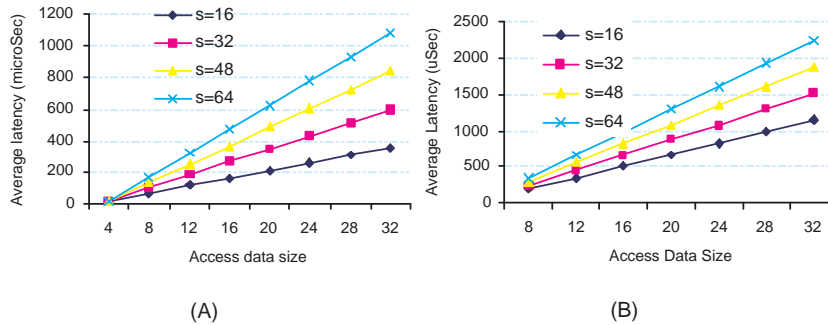
Fig. 8. Memory access overhead for DMM in TinyOS. Figure (A) shows the results when attribute is present in the data bank, and Figure (B) shows the results when the attribute is not in IES.

ciativity is increased, the access latency increases linearly because of increase in the number of comparisons DMM has to do to get the attribute. In case of miss from the data bank, the latency results include the cost of signalling DRE, the publisher doing *put*, and finally signalling the DAE. Figure 7 compares the memory access latency of accessing 32 Bytes data for various cases. It confirms the benefit of using the direct-mapped cache. When the data is available in the data bank, the memory access latency for a 32-way set associative data bank is comparable to that of directly accessing data from the HSN interface. For frequently accessed attributes, data access latency using direct-mapped cache is negligible compared to the latency using the HSN interface. If the data is neither in the direct-mapped cache nor in the data bank, the latency incurred is about three times more than the latency using the HSN interface.

6   Related Work

Related work to cross-layering can be broadly divided into two groups: the first considers all the layers together in a holistic way, and the second considers pairs of protocol layers. SensorStack falls in the first group; however, it uses the findings from the specific cross-layering instances between layers.

MobileMan project [4] also has similar goal to SensorStack to support cross-layering in a centralized way by facilitating information sharing. But there are two main differences between IES architecture and MobileMan's architecture [3]. First, instead of providing centralized shared memory, MobileMan provides call-back based approach such that consumers can directly access producer's private data. This approach implies that consumer has to know the publisher, the consumer has to do early binding to the producer, and asynchronous access to data becomes difficult. Second, conditions for asynchronous access are set as black-box functions instead of predicates over shared variables. Using their approach, even when there may not be any change in the shared data, every condition has to be checked periodically, thus leading to inefficiency.

Researches from Berkeley have proposed a sensor network architecture that takes a micro kernel approach. They advocate bringing down the standard interfaces to the

applications from the transport layer of the Internet stack to the link layer [5]. The proposed link layer abstraction, SP [15] also aims to share neighborhood information and message pool with all other protocols. While sharing of information is motivated similarly to IES, SP is confined only to link layer information, and they do not provide generic publish/subscribe interface like IES. Also, SP does not allow rule-based event notification as done in IES.

7 Conclusion and Outlook

This paper is a proof of concept that stackability and adaptability can be achieved simultaneously in a network protocol stack. We observe that cross-layering is important to achieve adaptability, but doing so arbitrarily limits the stackability. To solve this problem, we decouple cross-layer data from the functionalities provided by the layers. Based on this idea, we present the design of a novel *Information exchange service (IES)* to facilitate the cross-layering. The publish/subscription based data management module helps achieve stackability by standardizing the cross-layer interaction, and rule-based event management module helps achieve adaptability by supporting reactive notification of changes. We present a simple taxonomy for cross-layer information sharing that provides transparency without affecting stackability. We share our experience in implementing IES on TinyOS and Linux. TinyOS provides support for asynchronous communication among the components, which comes in handy for supporting the event notification service. However, the static nature of TinyOS makes the memory management restrictive, and the event notification inefficient. Linux, on the other hand, does not provide direct support for asynchronous communication among kernel modules, thus needs indirect mechanisms. We have presented results that show that the cross layer information gathering adds little overhead to the basic functionality of the stack. Currently, the IES is limited to information sharing for the modules within a single node. Our future work includes extending this information exchange service across different nodes of the sensor network.

References

1. A. Cerpa and D. Estrin. Ascent: Adaptive self-configuring sensor networks topologies. In *Proceedings of Infocom*, 2002.
2. B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *Mobile Computing and Networking*, pages 85–96, 2001.
3. M. Conti, G. Maselli, and G. Turi. Design of a flexible cross-layer interface for ad hoc networks. In *Fourth Annual Mediterranean Ad Hoc Networking Workshop*, June 2005.
4. M. Conti, G. Maselli, G. Turi, and S. Giordano. Cross-layering in mobile ad hoc network design. In *IEEE Computer*, number 2, pages 48–51, Feb 2004.
5. D. Culler, P. Dutta, C. T. Ee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, and J. Zhao. Towards a sensor network architecture: Lowering the waistline. In *The Tenth Workshop on Hot Topics in Operating Systems (HotOS X)*, June 2005.
6. C. Frank and K. Romer. Algorithms for generic role assignment in wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 230–242, New York, NY, USA, 2005. ACM Press.

7. T. He, J. A. Stankovic, C. Lu, and T. Abdelzaher. SPEED: A Stateless Protocol for Real-Time Communication. In *Proceedings of ICDCS 2003*.

8. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

9. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobile Computing and Networking*, pages 56–67, 2000.

10. Intel Research, Berkeley. Heterogeneous Sensor Network: http://www.intel.com/research/exploratory/heterogeneous.htm; Software available in TinyOS 1.1.10 snapshot from Sourceforge.net.

11. B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Mobile Computing and Networking*, pages 243–254, 2000.

12. V. Kawadia and P. R. Kumar. A cautionary perspective on cross layer design. In *IEEE Wireless Communication Magazine*, volume 2, pages 3–11, Feb 2005.

13. R. Kumar, M. Wolenetz, B. Agarwalla, J. Shin, P. Hutto, A. Paul, and U. Ramachandran. Dfuse: a framework for distributed data fusion. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 114–125, New York, NY, USA, 2003. ACM Press.

14. S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Operating System Design and Implementation(OSDI)*, Boston,MA, Dec 2002.

15. J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. A unifying link abstraction for wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 76–89, New York, NY, USA, 2005. ACM Press.

16. K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN) to IP. RFC 3168, IETF, Sep. 2001.

17. T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 171–180, New York, NY, USA, 2003. ACM Press.

18. M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)*, March 2004.

19. W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC protocol for Wireless Sensor Networks. In *Proceedings of INFOCOM 2002*, New York, June 2002.

20. H. Yokota, A. Idoue, T. Hasegawa, and T. Kato. Link layer assisted mobile ip fast handoff method over wireless lan networks. In *Proceedings of the 8th annual international conference on Mobile computing and networking (MobiCom '02)*, pages 131–139, New York, NY, USA, 2002. ACM Press.