System Support for Management of Networked Low-Power Sensors

Jai-Jin Lim, Daniel L. Kiskis, and Kang G. Shin
The University of Michigan, Ann Arbor, MI 48109-2122, U.S.A.
{jjinlim, dlk, kgshin}@eecs.umich.edu

***Abstract*— This paper addresses the problem of managing a wireless sensor network with mobile managers. The mobile managers should be able to create their connectivity to the nodes they manage, and advertise their interests in the management data to be collected. Also, the network nodes should self-manage their connectivity to the managers in order to forward the management data. To meet these requirements, we propose (1) an algorithm for creating and maintaining *encounter-associated management connectivity* and (2) both management data exchange protocols and programming abstractions for network management applications. Both our simulation-based evaluation and experimentation of the proposed algorithm and architectural elements on real sensors demonstrated their effectiveness in meeting the requirements.**

I. Introduction

Wireless sensor networks (WSNs) are becoming an attractive, cost-effective means to monitor the physical world [1]–[3]. WSNs are typically characterized by their less-attended deployment, ad hoc replacement, and limited field view. This is because individual sensors (i) require little or minimal human intervention after their deployment; (ii) replace faulty nodes in an ad hoc manner; and (iii) have a short sensing range and limited processing power.

For example, the TinyOS [4]-based mica Mote sensor in Fig. 1 is equipped only with 128KB of programmable memory, 4KB of data memory, and a 916MHz RF wireless channel capable of transmitting packets at a rate up to 40Kbps. As shown in Fig. 2, these resource constraints will continue to be a key design consideration for sensor nodes. They also pose serious challenges in the design of energy-efficient self-manageable protocols and services from byte-level protocols to middleware components, then to applications themselves, finally to peer-to-peer in-network collaboration.

The introduction of mobile devices into stationary WSNs has been studied to perform tasks that would otherwise be difficult to achieve [5]. The same idea has been explored for WSN management. For example, the authors of [6] addressed mobile ad hoc network management and the authors of [7] coined the term of *nomadic managers* to describe autonomous, mobile managers in the architectural descriptions of their WSN management. Indeed, the idea of using mobile managers is not only the most general way of overcoming the limited model of centralized management through a fixed gateway or a base station, but also a practical approach to managing sensor networks deployed across a large unattended area.

The problem space under consideration in this paper is not characterized by the managers' mobility alone. It is also influenced by the data-centric characteristics of WSNs. In the data-centric paradigm pioneered by Directed Diffusion (DD) [8], a data sink issues a query with a list of attribute–value pairs, which is called the *interest* in [8], and data are sent by sensors with data matching the queried interest. During the propagation of the interest and data, several important decisions such as the data-reporting rate and direction are *locally* made.

An advantage of using this paradigm is significant energy savings [8], which is a measure of critical importance to WSNs. Another advantage is localized in-network processing; it not only saves energy but also enables *management in anonymity* where the identity of an individual sensor reporting management data does not matter as long as the reported data are a correct aggregate of the situation around a target region.[1]

A manager is a "sink" that collects management data, which is an energy-consuming task. Thus, a natural question on sensor network management is *how to exploit data-centric communication in the context of network management while getting the most of the data-centric paradigm to achieve energy savings*—this is where we as well as many others [9] start for WSN management. We design a network management protocol for *both* creating the DD-style management connectivity dynamically and delivering management data reliably in the presence of network dynamics such as link/node failures, manager mobility, etc. In fact, the need for enabling management applications to manipulate their management connectivity has been identified from both WSN-specific management requirements [7], [9] and real-world experimentation [10].

There are several technical challenges in meeting our goal. First, a mobile manager is not part of the network; as a mobile manager enters, moves about, and leaves the field arbitrarily, it has to create and maintain the management connectivity as needed. Second, nodes managed by a given mobile manager need to maintain association with the manager to reliably deliver management information even with manager mobility. Third, sensors and managers need to have a well-defined messaging mechanism to exchange management information between them, so that in-network processing, such as filtering

[1] Of course, the identity of a sensor is application-dependent or relates to the granularity of management; some management application may still be interested in data on each individual sensor.

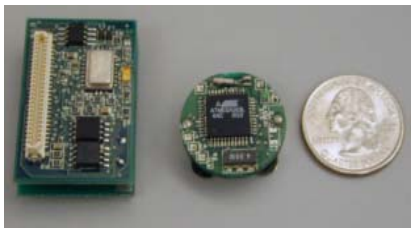


Fig. 1. TinyOS-based mica and dot Motes [4]

Mote	WeC	rene	dot	mica	mica2	mica2 dot	iMote
Released	1999	2000	2001	2002	2003	2003	2003
Processor	4 MHz				7 MHz	4 MHz	12 MHz
Flash (code, kB)	8	8	16	128	128	128	512
RAM (kB)	0.5	0.5	1	4	4	4	64
Radio (kBaud)	10	10	10	40	40	40	460
Radio Type	RFM				ChipCon	ChipCon	Zeevo BT
uController	Atmel						ARM
Expandable	no	yes	no	yes	yes	yes	yes

Fig. 2. TinyOS-based Mote hardware evolution [1]

and aggregation based on the type and importance of management data, can be easily supported.

Our main contribution in this paper is the design and implementation of a tiny network management protocol (TNMP) for low-power smart wireless sensors with mobile managers. TNMP enables sensors not only to build and self-manage management connectivity but also to exchange management information using the thus-constructed management connectivity in spite of managers' mobility. TNMP is characterized by the following salient features.

- **Self-managed management connectivity**: as a cross-layer interaction with the network layer, TNMP allows management applications to direct sensors to establish and maintain the management connectivity as needed.
- **Pattern-directed management operations**: as management programming abstractions, TNMP aims to support flexible composition of various network management functions and services in WSNs via pattern-directed classifications of management operations.
- **Component-grain management messaging**: for its physical realization on resource-limited sensors, TNMP employs a component-grain addressing, dispatching, and interaction model. Thus, it aims to fit best in a component-oriented sensor software architecture.

The rest of this paper is organized as follows. Section II discusses related work. Section III states our problem formally and details the design of algorithms and architectural elements. Section IV highlights the implementation of TNMP and the associated architecture. Section V evaluates the proposed protocol and architectural elements and their characteristics. Finally, we conclude the paper with Section VI.

II. Related Work

A. Network-management architectures and protocols

The Guerrilla Management Architecture (GMA) [7] is proposed to self-manage ad hoc networks with the collaboration of autonomous nodes and employs a two-tier infrastructure to facilitate adaptive, autonomous, and robust management of ad hoc networks. The management architecture for wireless sensor networks (MANNA) [9] discusses the management functional areas that are specific to WSNs. However, they do not address how to design and implement related architectural frameworks on real, low-powered wireless sensors.

Apart from two well-established standard network-management protocols such as SNMP or CMIP in a wired network, the Ad hoc Network Management Protocol (ANMP) [6] is a SNMP-compatible lightweight protocol for both managing and clustering mobile wireless ad hoc networks. Unlike TNMP that dynamically creates the management connectivity, ANMP relies on the existence of an underlying cluster-based routing algorithm for it. Furthermore, none of them are adequate for low-powered wireless sensors. For example, the mica sensor we are currently using has, by default, a MAC-layer payload of length 29 bytes; too limited for SNMP or CMIP.

B. Sensor programming and management abstractions

An abundance of new abstractions, protocols, and services for TinyOS-based wireless sensors has been reported in [1]. In spite of successful adoption of abstractions, such as TinyDB [11], Hood [12], Abstract Region [13], and SNACK [14], none of them properly addresses the general management concerns described in [6], [7], [9] for ad hoc wireless sensor networks.

Impala [15] describes a middleware architecture that enables application modularity, adaptivity, and repairability in WSNs. Moreover, it allows application software to be upgraded over the air in a modular fashion. Their solution, albeit useful, is limited to the network-wide software installation.

SNMS (Sensor Network Management System) [10] is also proposed as a management solution for TinyOS-based sensors. SNMS focused more on the design of a sensor health-data collecting service and an event logging service. However, its management abstractions are restricted in expressing various management concerns. For example, its messaging structure is too restricted to be useful for other services and applications.

sNMP [16], an on-going effort to build WSN management protocols and services, tries to find solutions without the limitations of existing network management approaches, as well as to offer WSN-specific management functions. However, no concrete protocol design or system building effort has been reported so far.

C. Sink-pull network formation protocols.

In a sink-pull system, a data sink periodically or occasionally propagates descriptions of data and events of interest to itself in a network. During the propagation of such interests, a routing path and associated cost from each data source to the sink are found, forming a cost or gradient field from each

source to the sink. Algorithms such as Directed Diffusion (DD) [8] and GRAB (GRAdient Broadcast) [17] fall into this category.

DD is closest to our work. The similarity and difference between the two are as follows. First, we adopt the diffusion of interests that a sink, in our case a manager, has and the establishment of the (management) connectivity among nodes that match the posted interests. Unlike DD, TNMP achieves the forwarding continuity of management data in the presence of sink mobility by adopting the destination sequenced distance in AODV [18]. Second, we choose to use structured messaging based on the refinement of data or operations. In DD, every data is exchanged in the form of a list of name-and-value pairs, which translates to unstructured messaging. The use of structured messaging helps easily develop and apply a message-level packet filter or aggregation based on the type and importance of management data it carries.

III. Problem Statement and Protocol Design

This section first states the problems to be addressed and then details the design of solution protocols.

A. Problem statement

Within the context of the aforementioned mobile managers, we would like to design and implement a protocol (i.e., TNMP) by solving the following three related subproblems.

Problem 1 (Self-managed management connectivity): how to create a network for on-demand management-data collection, and maintain it as current as possible even if the manager moves around?

Problem 2 (Pattern-directed management operations): what is an essential set of management operations, through which various sensor network applications can easily compose their management and coordination logic?

Problem 3 (Component-grain management messaging): how to design an addressing, dispatching, and interaction model for a wide range of component-based sensor applications?

B. Self-managed management connectivity

The key idea behind a solution to Problem 1 is to have (i) a manager advertise its interest as well as an encounter record, and (ii) any participating node in a management connectivity record the latest encounter with the manager while propagating the interest.

The *encounter* record is represented as a pair of {`mgrseq,mgrhop`}, where `mgrseq` is a monotonically-increasing sequence number assigned by the manager, and `mgrhop` is the distance (in number of hops) to that manager. As it follows from the name, `mgrseq` defines a temporal order of nodes with respect to the manager; the larger `mgrseq`, the more recent encounter a node has with the manager. Likewise, `mgrhop` defines a spatial order of nodes with respect to the manager; the smaller `mgrhop`, the closer a node is to the manager. Therefore, the encounter defines a spatio-temporal order of nodes with the manager.

Given this encounter-associated management connectivity, management data is forwarded to the manager via a hierarchial routing scheme; a node first forwards data to one of its neighbors that has a recent temporal encounter (larger `mgrseq`) and tries next on a neighbor with a close spatial encounter (smaller `mgrhop`) if both the node and the neighbor have the same `mgrseq`.[2] Such neighbor is here denoted as `mgrsucc`, a successor node to that manager. In general, the path from a node to the manager becomes a sequence of successors starting from itself. Since the management connectivity is created and maintained per manager independently of others, we assume that a single manager exists in the network.

Creating a management-data collection network. Similarly to DD, a manager starts by sending out an `mSetup` advertisement message. The `mSetup` message has a tuple {`mgraddr, mgrseq, mgrhop, lifetime, interest, ttl`}, where `mgraddr` is the manager's address. A management interest is specified in `interest` as a list of (name,value) pairs. The management network may be permanent or semi-permanent, depending on the value of `lifetime`: if this field contains a non-zero number, the network will disappear after the `lifetime` period; otherwise, the network will remain forever. Propagation of `mSetup` is bounded by the specified `ttl` (time-to-live in hop count).

The manager makes an initial *self-encounter* record {`mgrseq=<uniq>,mgrhop=1`}, where `<uniq>` is a monotonically-increasing sequence generator. When a node receives an `mSetup` message m from node `prev`, it makes the following *sequential* decisions;

- if it has no encounter with the manager, insert it and set `mgrsucc` to `prev`.
- if it has an encounter record e with the manager,
  - if $(e.mgrseq < m.mgrseq)$, replace e with m and `mgrsucc` by `prev`.
  - if $(e.mgrseq = m.mgrseq) \wedge (e.mgrhop > m.mgrhop)$, replace e with m and `mgrsucc` by `prev`.
- if there is any change in the encounter record, rebroadcast it with $m.mgrhop$ incremented by 1.

The first rule is called a "freshness rule" in that a fresher encounter with the manager is preferred, and the second rule is called a "short-hop rule" in that the encounter with a smaller hop count to the manager is preferred. We will henceforth call both rules simply the SUCCESSOR-UPDATE RULE since it changes `mgrsucc` for a given manager.

Fig. 3 illustrates this process. An `mSetup` message advertised by manager M will be propagated through the network. During this propagation, each node sets up a path to the manager, as well as a successor node leading to the manager. For example, node S learns that it is 3 hops away from the manager, i.e., `mgrhop=3`, for which it has `mgrseq` set to

[2] In case `mgrseq` wraps around to 0, nodes in the network can identify it as a more recent temporal encounter by comparing the difference between the previous value and the wrapped-around 0 against some threshold.

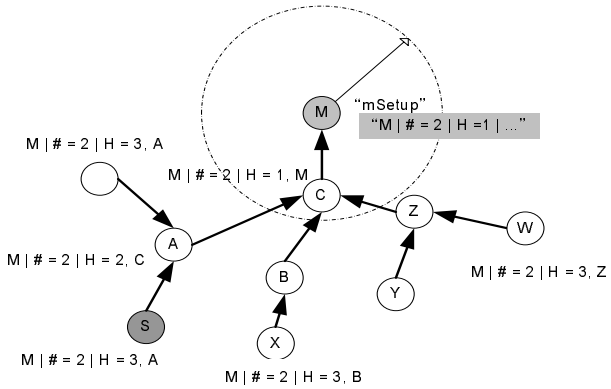


Fig. 3. An `mSetup` advertisement and cached encounter entries after the advertisement ends. Both $\sharp$ and H represent `mgrseq` and `mgrhop`, respectively.

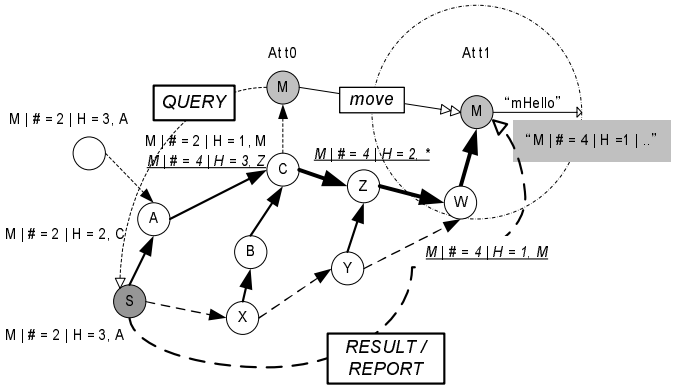


Fig. 4. Automatic path migration in the encounter-associated management connectivity. Both $\sharp$ and H represent `mgrseq` and `mgrhop`, respectively.

2, and that `mgrsucc` is A, through which it can reach the manager.

Maintaining management connectivity. Nodes advertise their encounter with the manager via 1-hop `mHello` messages. `mHello` has a tuple {`mgraddr, mgrseq, mgrhop, lifetime`}, where `lifetime` is the remaining lifetime of the link associated with `mgrsucc` towards `mgraddr`. Upon reception of `mHello`, nodes apply the same SUCCESSOR-UPDATE RULE as in the reception of `mSetup`.

There are two types of `mHello` advertisement: *mobility-induced* and *event-triggered*. The former type is used by a manager and works as follows. Assuming a manager node is aware of its moving speed (v) and current transmission range (r), it advertises `mHello` at every r/v seconds with a new self-encounter record. The manager may change its speed during its journey. The faster it moves, the more often `mHello` is advertised. Other refinements of this mobility-induced advertisement could be possible; for example, a manager knowing of its physical geographic location will advertise `mHello` only when it moves away from the previous location by more than the threshold distance, which may be again a function of the speed and the transmission range. We do not explore this any further since a simple periodic model suffices for our purpose.

The event-triggered advertisement is used by other nodes in the network, where nodes periodically check if their own encounter record has been changed since the time of previous encounter record check. It advertises `mHello` with an up-to-date encounter record only if their own encounter record has been changed. Note that the encounter record changes asynchronously upon reception of `mSetup` or `mHello` from its neighbors. A manager is assumed to be benign to, and cooperative with, other nodes in the network; it does not move back and forth very fast intentionally, flooding `mHello` and causing path oscillation. However, even in the case of such an undesirable movement any potential cascaded flooding of `mHello` can be tamed by the event-triggered advertisement that sends a new encounter record only once regardless of the number of new encounters received from neighbors during its encounter check interval.

Both of the advertisement schemes guarantee *automatic path migration* in the management connectivity. Fig. 4 illustrates the concept of automatic path migration. Suppose nodes C and Z in the figure are those receiving the latest encounter with M which just moved to another location at time `t1`. Eventually, node S may know the movement of its manager by receiving `mHello` with `mgrseq` set to 4. From that point and onward, packets from S may take a shorter trail toward M along the path S-X-Y-W-M.

Automatic path migration, however, does not completely remove the possibility of packet loss resulting from mobility-caused link breakage. For example, node C may still have an old encounter record if a new `mHello` from neighbors with more recent encounter records is lost, thus making it still believe that the manager is a 1-hop neighbor of itself. Although dense sensor deployment in its neighborhood makes such an event unlikely to occur, node C still needs to discover a new path to the manager.

To handle this problem, a node issues a local route request `mRRequest` via 1-hop broadcast upon detection of a link or node failure through a link-layer ACK mechanism. `mRRequest` has a tuple {`mgraddr, mgrseq, mgrhop, source, srchop`}, where `source` is the node that issues `mRRequest`, and `srchop` is the number of hops taken from `source`. When `mRRequest` is broadcast by `source`, `srchop` is set to 1.

Upon reception of an `mRRequest` message m, immediate neighbors or multi-hops-away nodes make the following decisions:

- If it has no encounter with the manager, drop it.
- If it has an encounter record e with the manager,
  - if $(self = m.mgraddr)$, send `mRReply` with a new self-encounter record;
  - if $(e.mgrseq > m.mgrseq)$, send `mRReply` with e;
  - if $(e.mgrseq = m.mgrseq) \wedge ((e.mgrhop + m.srchop) \leq m.mgrhop)$, send `mRReply` with e;
  - if (my successor for $m.mgraddr \neq m.source$), *unicast* `mRRequest` to successor with $m.srchop$ incremented by 1;
  - otherwise, drop it.

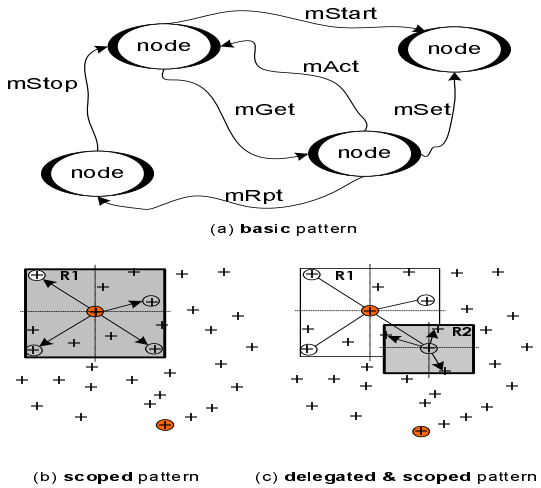


Fig. 5. Patterns in sensor network management operations.

Three rules used to send `mRReply` are called the ROUTE-REPLY RULE. Note that `mRRequest` is unicast toward the manager if there is no feasible successor in the immediate neighborhood and that `mRReply` is unicast to the originator even though `mRRequest` is received via 1-hop broadcast.

`mRReply` has a tuple {`mgraddr, mgrseq, mgrhop, lifetime`}, similar to that of `mHello`. While `mRReply` is relayed to `mRRequest.source`, a reverse path cached during the propagation of `mRRequest` is used. Each time `mRReply` is relayed, `mRReply.mgrhop` is incremented by 1. In case multiple `mRReply`s are received, the SUCCESSOR-UPDATE RULE is applied on every received message. Since `mRReply` is likely to carry information of a new path to the manager, an intermediate node too applies the same SUCCESSOR-UPDATE RULE.

However, the originator of `mRRequest` may still receive no `mRReply` at all before its route discovery timer expires. In such a case, the originator invalidates all encounter records for the manager. Then, it enters the *probe* mode in which it sends `mHello` with `mgrhop` set to ∞ at every probe interval, which should be long enough. The probe mode is completed when it receives a new encounter advertisement from neighbors or a link `lifetime`, which was known at the first encounter with the manager, expires. Any upstream node using that node as a successor to the manager will, in turn, detect a link/node failure associated with that node. Then, a new recovery procedure will be started automatically.

C. Pattern-directed management operations

Nodes need to exchange their management information over the management connectivity. A solution to Problem 2 is the definition of sensor-network-specific management operations similar to SNMP or CMIP management operations in a wired network.

Three patterns are identified as shown in Fig. 5: *basic*, *scoped*, and *delegated* patterns.

The basic pattern. This reflects a basic interaction model among autonomous sensors participating in management and coordination applications. Fig. 5(a) gives a pictorial representation of arbitrary basic interactions among sensors; the dark area around nodes represents the exposed attributes and methods, and a directed arrow represents the invocation of an operation. The operations in the basic pattern are modelled after the two standard network management protocols, SNMP and CMIP, and existing sensor network abstractions. Table I summarizes their meanings and operational modes. Of the defined basic operations, `mStart` and `mStop` are management operations specific to sensors, whose software is most likely to be built in a modular fashion with mountable physical sensing modules and soft-wired functional modules.

TABLE I

THE PROPOSED BASIC MANAGEMENT OPERATIONS.

name	description	mode
mStart	Activates a component. Holistic operation.	U
mStop	Deactivates a component. Holistic operation.	U
mGet	Requests to retrieve the values of attributes.	C
mSet	Requests to modify the values of attributes.	U/C
mAct	Requests an action to be executed with operands.	U/C
mRpt	Reports the occurrence of an event.	U
mErr	Reports the occurrence of a processing error.	U

The "mode" classifies operations into two categories: a request-response style, i.e., confirmed (C), and a one-way request style, i.e., unconfirmed (U). It defines a choice of end-to-end reliability from an application's perspective.

Almost all management operations support the unconfirmed mode, which is a best-effort invocation and is less expensive messaging in wireless networks. An unconfirmed operation, albeit less reliable, is suitable for repetitive invocations of continuous and periodic monitoring operations that are robust to packet losses to some extent.

A confirmed mode operation is replied with either a valid/error response from the performer or a timeout locally. The choice of a confirmed or unconfirmed operation is up to applications that know best the need of end-to-end reliable exchange of management operations. For example, unlike a continuous monitoring operation, migrating or spawning a management function to another capable node may require end-to-end reliability between the invoker and the performer of the management function.

To achieve end-to-end reliability, `mErr` is used to indicate the occurrence of an error in processing the requested *confirmed* operation at a remote sensor. However, `mErr` is not sent for an unconfirmed operation even if a remote sensor encounters difficulties in processing the received unconfirmed request. Thus, any application invoking a confirmed operation on a remote sensor is expected to receive a normal or error response from a remote sensor, or a local timeout. Note that `mErr` is also subject to loss since it is an unconfirmed operation.

The scoped pattern. This reflects localized computations at sensors where a basic operation can be applied to multiple sensors simultaneously to process a one-to-many management operation such as configuration changes or many-to-one data aggregation. Fig. 5(b) illustrates the scoped operation with a simple rectangular geographic region around the querying

node as its scope constraint. In effect, a scoped operation is the basic management operation with a scope constraint specifying which sensors the operation should be applied to. Such a scope constraint can be expressed several ways: *hop-based, region-based*, and *attribute-based*.

The hop-based or region-based scope constraints are of the simplest form, such as k-hops away nodes or nodes within a given geographical region, whereas an attribute-based one takes a more or less complex form involving a general boolean filter on a set of attributes.[3] Regardless of how the scope constraint is specified, only those nodes satisfying the specified scope constraint are eligible to execute the requested operation.

The delegated pattern. The last type is the delegated pattern through which a basic operation is delegated to another sensor for its remote execution and the corresponding result is returned only to the original querying node. Such delegation will become indispensable to *in-network data aggregation and management*. For example, one sensor may issue another sensor a query for a summary of events or aggregation of sensor readings within the region surrounding that sensor. Fig. 5(c) illustrates this concept, where a management operation is delegated by some node in region R1 to another node in region R2. Although the execution of a delegated operation on a remote sensor is an algorithmic design problem of the invoked operation on that sensor, it requires a separate mechanism to return the execution results to the original querying node, which is stationary or nomadic, or moves around continuously.

D. Component-grain management messaging

A solution to Problem 3 is the design of a protocol data unit (PDU) and an architectural framework in resource-constrained sensors, given the identified management patterns and the physical characteristics of sensors.[4]

TNMP uses the operations defined in Table I as protocol primitives to introduce a management messaging service. A component-grain addressing and interaction model is introduced to provide a systematic way of building management and coordination applications that are most likely to be built component-by-component [1]. A component can be a module implementing a sensor board, a module implementing a middleware service, or a management application itself. Therefore, TNMP operates at any network layer, and provides a communication service that delivers management and coordination information between networked components.

The message format of TNMP is given in Fig. 6. `pduId` represents the management operation defined in Table I. For unconfirmed operations, `pduIds` are `mStart`, `mStop`, `mSet`, `mAct`, `mRpt`, and `mErr`. For confirmed operations, `pduIds` are `mGetReq`, `mGetRes`, `mSetReq`, `mSetRes`, `mActReq`, and `mActRes`.

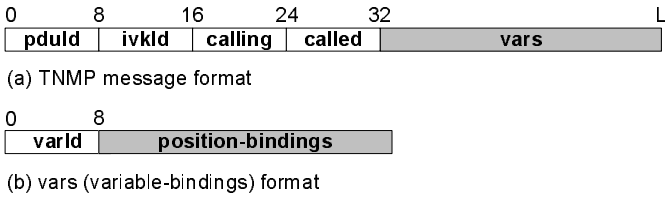


Fig. 6. TNMP PDU format of length L bits over TinyOS-based sensors. Bit alignments are tailored for TinyOS-based sensors and the actual value of L depends on the underlying routing primitives.

Associated with `mErr` are both an error code and an invocation context. Currently, such an error code is defined to have `accessDenied` and `processingError`. `accessDenied` is used to indicate that a requested operation cannot be applied temporarily due to the target component's deactivation. `processingError` indicates an error encountered during the processing of a received confirmed request. The invocation context includes information about the calling component such as its network address and requested operation. Both the error code and the invocation context will be carried in `vars`, which will be described below.

`ivkId` is used to differentiate between outstanding requests by associating each request with a unique invocation ID which will be referenced in case an `mErr` is returned or a timeout is signaled. The `calling` and `called` are used to address calling and called components, respectively. Since the `calling` and `called` do not have to be the same, a general many-to-one consumer and producer relationship can be established between components.

`vars` represents variable bindings. Typical variable bindings in SNMP or CMIP are a list of variable names and their corresponding values. They are also encoded in a machine-independent transfer syntax when transmitted across the network. However, such canonical variable bindings in a low-power wireless sensor are too costly for a limited payload size.[5] For example, a mica Mote is known to have a payload of 29 bytes in the TinyOS [4] MAC-level frame.

Instead of using canonical (name, value) pairs, TNMP uses a design-time agreed-on message structure that consists of a list of values only, wherever appropriate. This marshalling scheme is called *position-bindings* since the starting position of a variable in the received `vars` byte streams is known in advance, and its structure is agreed on *a priori*, as well. As shown in Fig. 6(b), the only tag information, represented by `varId`, is carried to identify a correct message structure. Note that such position-bindings are a common practice in TinyOS-based sensor programming to get around the limited payload size.

[3]As of this writing, the attribute-based scope constraint is not fully studied regarding how to specify it within a limited payload size of less than 20 bytes—physical limitation of our current implementation. However, the CMISFilter in CMIP may help readers understand what an attribute-based scope constraint looks like. Albeit less than perfect, either hop- or region-based scope constraints are still most popular with many of current sensor network aggregation and monitoring applications due to the localized computation.

[4]We assume use of TinyOS-based mica Mote sensors.

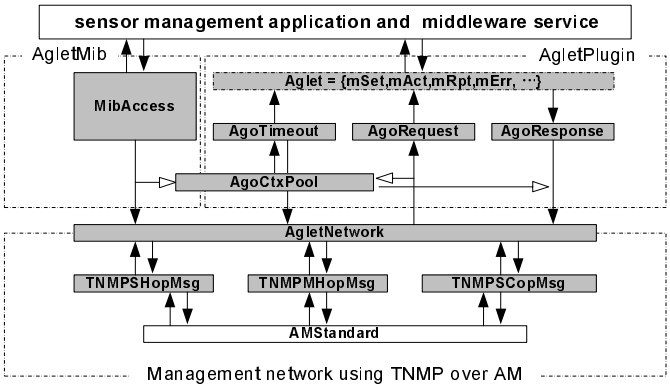


Fig. 7. Runtime structure of the implemented system. Hidden underneath `AgletNetwork` are the control-plane messages for `mSetup`, `mHello`, `mRRequest`, and `mRReply`. `Aglet` is implemented as a collection of the TinyOS/nesC parameterized interfaces [4], whereas `AgletMib` and `AgletPlugin` are implemented as the TinyOS/nesC configuration and module [4].

IV. Implementation

Implementing TNMP requires (i) application programming interfaces (APIs) for binding components, accessing remote attributes and functions, (ii) TNMP's finite state machine (FSM) for dispatching incoming TNMP PDUs and handling requests & timeouts, and (iii) cross-layer interaction interfaces (CIIs) between TNMP and the underlying routing protocols not only to build and maintain the management connectivity, but also to deliver TNMP PDUs using the connectivity. We implemented these on mica Mote sensors. Fig. 7 outlines the runtime structure of the implemented architectural elements. Due to space limitation, only a brief account of each of them is given below.

Implementation of APIs. `MibAccess` and `Aglet` are main elements that provide APIs. `MibAccess` offers uniform management abstractions, as shown in Fig. 8, that enable applications to access attributes and methods in the management information base (MIB) regardless of whether they are local or remote.

As shown in Fig. 8, these management abstractions are classified into four categories: confirmed basic operations (O1), confirmed scoped operations (O2), unconfirmed scoped operations (O3), and unconfirmed basic operations (O4). The O1 operations are returned with either a valid/error response from remote sensors or a timeout if the response is not received from remote sensors within the specified time limit. Depending on the value of `node` parameter, a requested operation may be invoked on local components. If `node` is multiple hops away, the request will be sent across the management network. For those scoped operations such as in (O2) and (O3), only the hop-based scope constraint is implemented. Support for other scope constraints is left for future work. Note that the delegated operations do not require any mechanisms other than these basic and scoped operations.

`Aglet` is just a collection of parameterized interfaces for each of the management operations defined in Table I. The parameterized interface [19] in TinyOS is a heavily-used programming abstraction that makes it easy to build a dispatching framework with dynamically-instantiated interfaces. For instance, what an application component wishing to receive TNMP `mRpt` has to do is to bind this `mRpt` operation. Then, a dispatching framework is automatically built to handle incoming TNMP `mRpt` messages. Likewise, other management operations are easily incorporated into a dispatching framework. The reference [19] details how to build a dispatching framework by using the prameterized interface.

Implementation of TNMP FSM. Four key elements are related to handling an `Aglet` operation (*Ago*) carried in TNMP PDUs: `AgoCtxPool`, `AgoTimeout`, `AgoRequest`, and `AgoResponse`.

`AgoCtxPool` keeps track of the invocation contexts associated with either incoming or outgoing TNMP messages. The invocation contexts are maintained only for confirmed operations. The information contained in the invocation context includes the `timeout` period, and TNMP header information such as `msgId`, `ivkId`, `called` and `calling`. Routing information such as the destination, `node`, and the casting mode, `cast`, is also saved for future reference.

`AgoTimeout` signals a timeout event to the local application component if a response from remote sensors is not received within the specified timeout period. It also signals a timeout event to remote sensors if a reply to the received confirmed request is not generated after the specified timeout period. In the former case, a timeout event is signaled to the local application component through the bound `mAgoTimeout` operation via the `AgoTimeout` interface. In the latter case, an `mErr` TNMP message with the `processingError` error code is delivered to the remote application component. The timeout event also frees the associated invocation context from `AgoCtxPool`.

`AgoRequest` utilizes the dispatching framework to pass the received TNMP message up to a target application component. Recall that the target application component is identified by the `called` in the incoming TNMP message. Before calling the target application component, `AgoRequest` saves the invocation context for later reference by a valid response or by a timeout response.

`AgoResponse` provides an interface through which applications reply to the received request of confirmed operations. It looks up the saved invocation context to build a response TNMP message, as well as the passed-down reply payload from the application.

Implementation of CIIs. A TNMP PDU can be delivered via three different types of communication: single-hop broadcast, multi-hop unicast, multi-hop broadcast. To provide a transparent transport mapping of TNMP PDU, a facade [20] design pattern is implemented in `AgletNetwork`. As shown in Fig. 7, `AgletNetwork` multiplexes three different communication types `TNMPSHopMsg` (single-hop broadcast),

[5]It may still need to use a canonical machine-independent transfer syntax in a network composed of heterogeneous sensors at the expense of wasted byes.

```
O1: command result_t {get,set,act}(uint16_t node, uint8_t called, void *pVar, uint8_t timeout);
O2: command result_t {get,set,act}Scoped(uint8_t called, void *pVar, uint8_t hops, uint8_t timeout);
O3: command result_t {set,act,start,stop}ScopedOneway(uint8_t called, void *pVar, uint8_t hops);
O4: command result_t {set,act,start,stop,rpt,err}Oneway(uint16_t node, uint8_t called, void *pVar);
```

Fig. 8. Management abstractions.

`TNMPMHopMsg` (multi-hop unicast), and `TNMPSCopMsg` (multi-hop broadcast), each of which is assigned a unique identifier in the TinyOS MAC-level Active Message (AM) frame.

Applications inform `AgletNetwork` of `cast` mode only, which specifies the type of TNMP message frame. Conversely, `AgletNetwork`, upon reception of an AM message from below, can tell which type of TNMP message frame is used by the unique identifier in the AM header. Then, it passes the received TNMP message up to the application with a pointer to the starting position of the TNMP message in the AM payload.

V. Evaluation

A. Quantitative evaluation

We would like to answer the following questions: (i) how large is the memory footprint that TNMP incurs? (ii) what are the delivery performance and overhead on real sensors or an equivalent simulation environment? (iii) what are the delivery performance and overhead, as compared to other algorithms?

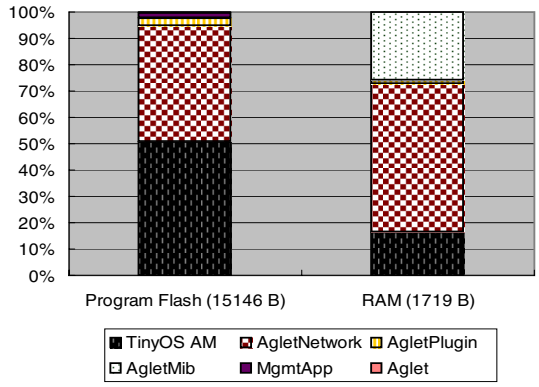


Fig. 9. Code size breakdown into key architectural elements.

1) Compile-time code-size distribution: Since this architectural framework is expected to evolve over time, either through code optimization of an existing implementation or the addition of new features, any quantitative number may soon become obsolete. Nonetheless, such a number can be useful to establishment of a comparative feel for our architectural framework against others. Thus, we calculate the code size for architectural implementation without any substantial management application logic.

Our current implementation on the TinyOS mica Mote sensor requires a total of 15146 bytes out of 128 KB program Flash memory and 1719 bytes out of 4KB RAM. These numbers are broken down into architectural elements, as shown in Fig. 9. The networking part of the implementation is a major source of this space consumption that is unavoidable in any networked sensor. About 50% of the Flash memory consumed is used by the intrinsic TinyOS AM and physical-layer implementations, and about 43% of it by `AgletNetwork`, a TNMP-specific routing extension to the existing TinyOS `Surge` and `MultihopRoute` networking abstractions.

Thus, the remaining 7% can be thought of as the net overhead incurred by TNMP-specific elements. This overhead is broken down as follows. First, the `AgletPlugin` binding and dispatching element accounts for about 3% overhead or 450B, whereas the overhead incurred by the `AgletMib` is less than 1%. The sample application skeleton `MgmtApp` incurs about 2% overhead. This application skeleton has only two incoming TNMP messages, `mGetRes` and `mRpt`, as well as one `MibAccess get` and an associated timeout `mAgoTimeout`. `Aglet`, which accounts for less than 1% overhead or about 42B, was written to measure the incremental overhead per incoming TNMP message. Thus, every addition of one incoming TNMP message handler requires an additional 42B of the Flash memory due to the additional comparison and branch code in a dispatching module.

Similar interpretations are also made for RAM space consumption. The networking part is still a major consumer of RAM. Note that a major portion of RAM space is usually used by statically-allocated data or table in the code. For instance, about 72% of the RAM consumption by `AgletNetwork` is associated with the size of a default neighbor table, a routing table, and packet pool. All these values are configurable for sensors with restricted RAM size. About 26% or 450B goes to the `AgletMib` which also maintains catalogs of the exposed attributes, methods, and variables (or arguments in the method). The remaning 2% goes to all the other parts.

In summary, our architectural framework except both `AgletNetwork` and TinyOS AM components incurs about 500B in both program Flash and RAM spaces, with an incremental overhead of 42B whenever a new management operation is added.[6] Although these numbers may change with code optimization or feature addition, they indicate the small memory footprint of the proposed architectural framework.

2) Runtime performance evaluation via PowerTOSSIM: A sample management scenario was written to understand the dynamic aspects of the proposed architectural elements by using PowerTOSSIM [21]. PowerTOSSIM and its base TOSSIM [22] have been known to provide an accurate and scalable simulation by running the *same* actual implementation

[6]We exclude the space consumption by `AgletNetwork` and TinyOS AM for the following reasons. First, the space consumption by an extension of the existing TinyOS networking component to `AgletNetwork` is negligible. Second, both the existing TinyOS networking and AM components are so intrinsic they should be shared by not only management applications but also other applications.

code as on real TinyOS-based sensors. Thanks to this property, a study with PowerTOSSIM is helpful when there is no real sensor or there are not enough sensors available for meaningful evaluation of scalability.

The sample management scenario is composed as follows. A mobile manager enters the sensor field and sets up its management network by broadcasting `mSetup` every 100 seconds with its interests and new self-encounter. Other nodes in the network then participate in building and maintaining a management network rooted at the manager. They will check their encounters with the manager every 10 seconds to determine whether to send `mHello` or not. One source node is chosen to emit `mRpt` to a mobile manager every 4 seconds.

Given this scenario, the following performance metrics are evaluated.[7]

- *Delivery ratio*: the ratio of the total number of packets received by the mobile manager to that sent by the source.
- *Delivery overhead*: the ratio of the total number of packets generated in the network to that received by the manager, normalized with the measured average number of hops between the manager and the source. In other words, it is the overhead per received packet and per hop. Also, it indicates the energy consumption involved in the communication, which is a major source of energy consumption in wireless networks.
- *End-to-end delay*: average end-to-end (e2e) packet delivery delay, in simulation seconds, between the source and the manager.
- *E2e hop count*: average hop count between the source and the manager.

Two distinct overheads contribute to the packet-generation overhead in the network: *data overhead* and *protocol overhead*. The data overhead is mainly associated with retransmissions, whereas the protocol overhead is associated with the routing protocol itself. In our case, `mHello` is counted toward the protocol overhead.

Fig. 10 shows a screen shot from the simulation study. A 5x5 grid topology with the *stationary* manager 0 at the upper-left corner and the source 24 at the lower-right corner is used. The transmission range of a sensor is nominally set to 10, which is the side length of a grid cell. The plot in the screen shot shows the cumulative performance evaluation results up to about 1000 simulation seconds. After a warm-up period of building the management connectivity, the performance metrics under study are shown to be stable. For example, we achieved an almost 99% or high packet delivery ratio with the delivery overhead of 2. The delay turns out to be about 2 simulation seconds in reporting an event to the manager, which is 8 hops away from the source.

3) Runtime performance evaluation via ns-2: Although the simulation study via PowerTOSSIM enables programmers to debug the code and obtain accurate and dependable information before any real-world deployment, its study in the context of node mobility is limited due in part to the inconsistency

[7]We extended PowerTOSSIM to measure our performance metrics.

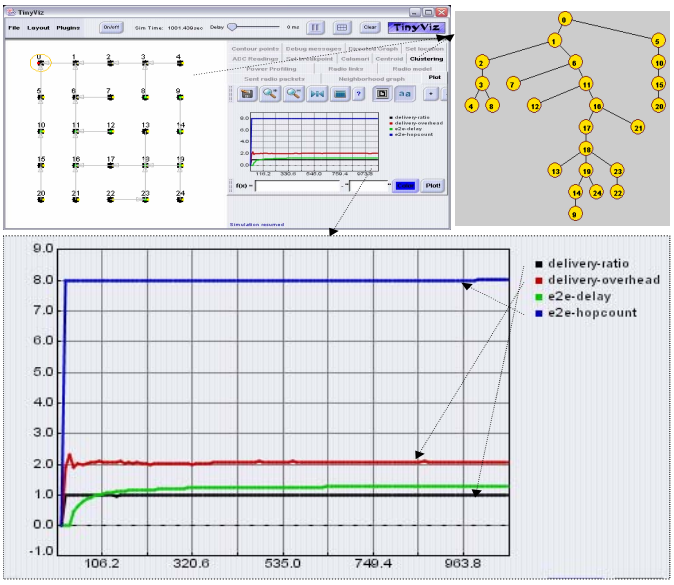


Fig. 10. Runtime performance evaluation via PowerTOSSIM.

in the link loss probability during the manger's movement.[8] Also it cannot offer a comparative perspective of the proposed maintenance of the management connectivity against others. Thus, we use *ns-2* [23] for detailed and comparative aspects of the proposed connectivity maintenance under various manager mobility scenarios.

In the *ns-2* simulation study, nodes remain stationary after they are randomly deployed in the field of size 1500m $\times$ 300m. The number of nodes deployed in the field is 100. Each simulation runs 1000 seconds. The *ns-2* default AT&T WaveLAN wireless interface card specification is used as a physical wireless interface. The default IEEE 802.11 CSMA MAC model is used as the MAC protocol. The transmission range is homogeneously set to 150m.

While other nodes are stationary, a manager node moves around the field following the *random waypoint* model [24]. In the random waypoint model, a node moves from one random location to another random location at a randomly-chosen speed, which is uniformly distributed between 0 and maximum speed—the maximum 20m/s is commonly used in the literature. Once the destination is reached, another random location is chosen after a pause. We varied the pause period from 0 (continuous mobility) to 999 (no mobility in 1000 simulation seconds). The source node is randomly chosen from among the stationary nodes.

Given the same management scenario and the simulation setting as above, we run and compare our connectivity maintenance against SWR (Single path With Repair) [25], which claims to outperform other well-known schemes, DD and GRAB. SWR maintains the connectivity with sink based on hop count only. Note that SWR, DD, and GRAB are all sink-driven network formation protocols without considering the

[8]As of this writing, PowerTOSSIM does not update the link loss probability after a node movement.

sink mobility. To the best of our knowledge, there is no sink-driven many-to-one network formation algorithm that takes the sink mobility into account. Nevertheless, evaluation against SWR gives a comparative perspective of ours since SWR has a mechanism to cope with a link failure with the sink regardless of the cause of failure.[9] For ease of presentation, TNMP's management-connectivity creation and maintenance is abbreviated as MODV (Many-to-One Distance Vector).

Fig. 11 shows the evaluation results under the above-defined performance metrics, which are averaged over 10 different topologies. As for the delivery ratio, MODV is shown to outperform SWR regardless of the mobile manager's speed as shown in Fig. 11(a). When the manager's mobility is very low (e.g., characterized by a model with a pause of 700 sec or more), they are comparable to each other. However, in case of a high mobility model where the pause time is close to 0, MODV achieves a much higher packet delivery ratio than SWR.

As for the delivery overhead, MODV still outperforms SWR; the faster the manager moves, the larger its performance gain becomes. However, MODV turns out to traverse longer or more circuitous paths than SWR—see Fig. 11(d). Therefore, MODV suffers a longer delay as seen in Fig. 11(c). The MODV's increased path length does not necessarily imply that it incurs more overhead; as already seen in the delivery ratio and overhead evaluations, it still achieves the highest packet delivery ratio with the minimum overhead. Our evaluation also confirms the fact [26] that the minimum hop path routing may not be the best in wireless networks.

We also investigated the case of multiple sources. In the multiple sources scenario, each node makes a probabilistic decision on its `mRpt`. The probability model intentionally favors distant nodes from the manager so that the evaluation result is not obscured by the behaviors of nodes close to the manager. Specifically, the actual report of `mRpt` is given as $\min\{0.2 \times H, 1\}$, where H is the hop count from a node to the manager. As can be seen in the figure, conclusions drawn upon the single source scenario still hold for the multiple source case.

Finally, we studied the impact of the encounter check interval on the performance metrics under study. Fig. 12 shows the evaluation results with the check interval ranging from 5, 10 to 20 seconds. Recall that `mHello` by nodes is not sent out at every check interval since it uses event-triggered advertisements, and that `mHello` by a manager is not affected by this check interval. Nevertheless, the shorter check interval, the higher chance a node has to detect the manager's movement early. The evaluation results in the figure show that the metrics we investigated are more or less insensitive to the check interval. Interestingly, frequent checks of the encounter in the high mobility scenario increase the delivery ratio, but incur the least overhead.

[9]We modified SWR to be mobility-aware by making it interpret `mSetup` and `mHello` accordingly. Other link or node failure recovery mechanism is implemented as described in [25].

B. Qualitative evaluation

We answer the following questions: (i) how similar or dissimilar are the proposed management operations to traditional network management operations? (ii) what unique features to wireless sensor management are introduced against other sensor network abstractions?

Aside from the need to create and maintain the management connectivity, TNMP's programming abstractions and architectural features are compared (Fig. 13) against existing network management protocols and other sensor network protocols in terms of programming abstractions. Existing sensor network programming abstractions like TinyDB, Hood, and Abstract Region are more or less favoring the applications that need to share data among neighbors or aggregate sensed data across the network. Since network management applications tend to require more than what data sharing or aggregation applications do, it is natural to follow or adopt existing network management protocols and abstractions.

We stress that the choice of a proper protocol and abstraction that matches a given application has a significant impact on the application performance [27]. Thus, the programming abstraction and paradigm of TNMP are influenced greatly by traditional network-management protocols. But it comes with the addition of sensor-network-specific protocol operations and constraints. For example, newly-introduced `mStart` or `mStop` reflects the need to support a holistic operation on sensors. It also comes with current limitations associated with WSNs. For example, the limitation of managed object representation and naming, and the lack of a proper marshalling scheme reflect less demand for the inter-operability in the sensor network. Of course, these issues need to be overcome as heterogeneous sensors with disparate hardware and software capabilities are likely to be used simultaneously in future.

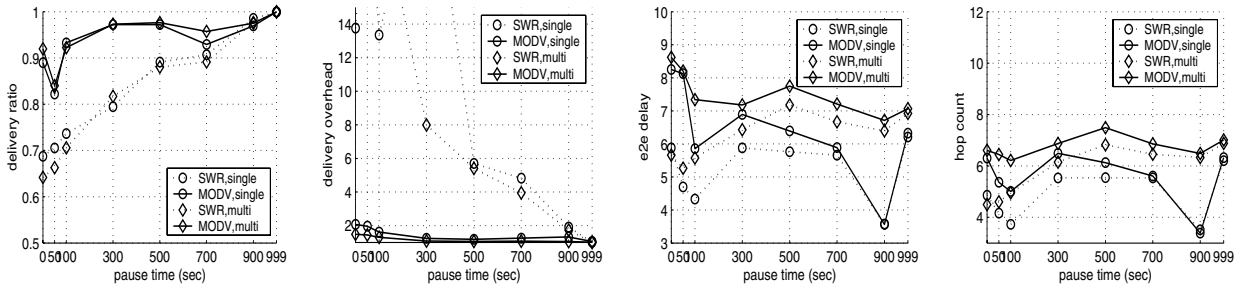
VI. Concluding Remarks

A. Conclusion

In this paper, we addressed issues associated with the management of wireless sensor networks (WSNs) that are fundamentally different from those associated with traditional network management. In case of networked autonomous sensors, management functions and roles must be distributed flexibly and dynamically across the network. Moreover, WSNs require maximal automation and in-network processing of management functions and services. Self-manageability of networked sensors is essential.

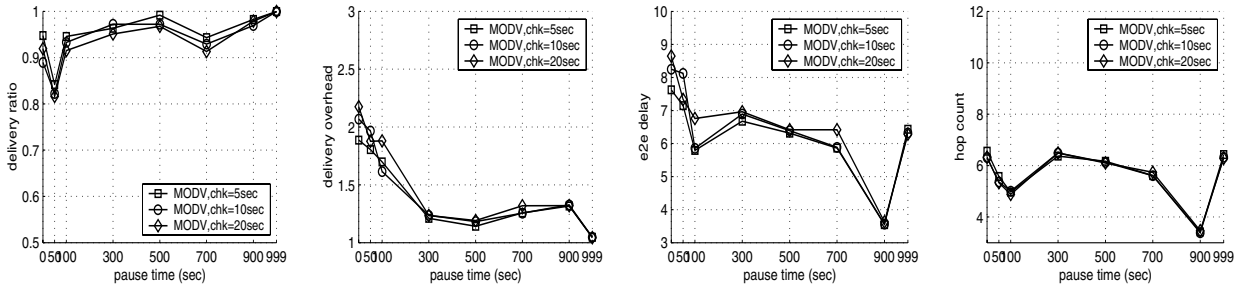
However, lack of proper management protocols and abstractions has hindered the progress in sensor network management. We addressed this deficiency by designing a tiny network management protocol (TNMP) and its related management abstractions at a modest—neither too low nor too specialized—level.

The proposed TNMP has a mechanism not only to cope with a mobile manager in the network but also to advertise its interests as needed. Upon such a new advertisement, nodes in the network will decide whether to join the management



(a) delivery ratio (b) delivery overhead (c) e2e delay (d) e2e hop count

Fig. 11. Performance comparisons under single source and multi sources.



(a) delivery ratio (b) delivery overhead (c) e2e delay (d) e2e hop count

Fig. 12. Impact of the encounter check interval on the performance metrics investigated.

	Internet	Telecomm. network	**Sensor network (TinyOS-community)**		
Target	Network management	Network management	**Network Management**	Neighboring / Data sharing	
Functional entity	manager - agent	manager - agent	**peer-to-peer**	peer-to-peer	
Protocol/Abstraction	SNMP	CMIP	**TNMP**	Hood	Abstract (Spatial) Region
Communication	Point-to-point (PtP)	Point-to-point (Ptp)	**Ptp/Broadcast**	Ptp/Broadcast	Ptp/Broadcast
Stack	TCP / IP	OSI	**AM**	AM	AM
Encoding	ASN.1/BER	ASN.1/BER	-	-	-
Managed objects	Attribute	Attribute / Function	**Attribute / Function**	Attribute	Attribute
representation	Scalar and Tabular	Any abstract data type	**Scalar**	Scalar	Scalar
naming	Hierarchically organized	Hierarchically organized	**Enumerated**	Enumerated	Enumerated
Protocol primitives					
attribute retrieval	GET, GETNEXT	M-GET	**mGet**	GET	GET
attribute update	SET	M-SET	**mSet**	SET	PUT
function invocation	SET with ScriptMIB	M-ACTION	**m{Act,Stop,Start}**	-	REDUCE with Aggr. Funcs.
event notification	TRAP	M-EVENT-REPORT	**mRpt, mErr(error noti)**	PUSH/UPDATED	PUT
object creation	SET	M-CREATE	-	-	-
object deletion	SET	M-DELETE	-	-	-
Multiple selections					
scoping/filtering	-	parameterized CMISFilter	**parameterized**	-	geographic filter
atomic operation	-	parameterized CMISSync	-	-	-

Fig. 13. Comparison of TNMP (data-plane only) against two standard network management protocols and other sensor network abstractions.

network or not, depending on the posted interests, and once joined, it continually manages the connectivity to the manager by keeping track of encounters with the manager. Thus, it enables a data-centric management paradigm in WSNs.

Moreover, our proposed architectural framework remedies the several limitations in existing abstractions at a small increase in the memory footprint. Especially, our approach makes best use of a component-oriented sensor software architecture with a component-grain addressing and interaction model. The proposed architectural framework has also been implemented and evaluated on real sensors.

B. Discussion and Future Work

Traditionally, a network management protocol does not care about the creation and maintenance of a management network, because (i) it is used for a wired network; (ii) the network connectivity is no concern to an application running on the network layer; (iii) there is no need for energy savings. However, the emergence of wireless systems and wireless sensors has changed all of these. A new design guideline has been developed for *cross-layer design and optimization* [28]. Also, the importance of application-aware communication has been emphasized by many researchers, especially in wireless sensors; the authors of [27] showed that application-specific refinements of an algorithm can affect application performance by 40–60%. It is, therefore, reasonable for management applications to be responsible for management connectivity by interacting with the network layer.

We plan to continue the development and experimentation of the proposed framework by adding key management functions. Furthermore, we will investigate the self-managed encounter-associated connectivity when all sensors are mobile as an extreme case study. Lastly, we will study the impact of the duty-cycling of sensors on the self-manageability of the connectivity. In the duty-cycling mode, sensors turn on and off their radio interfaces to prolong the lifetime of sensors. Due to this on-and-off duty-cycling, the routing stability and continuity are severely disrupted. Thus, we will investigate how to make our algorithm robust to the duty-cycling of sensors.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers and the shepherd, Alexander Keller, for their comments. The work reported in this paper was supported in part by the NSF under Grant CNS-0435023.

REFERENCES

[1] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler, "The emergence of networking abstractions and techniques in tinyos," in *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, March 2004.

[2] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis, "Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea," in *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*. ACM Press, 2005, pp. 64–75.

[3] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong, "A macroscope in the redwoods," in *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*. ACM Press, 2005, pp. 51–63.

[4] UCB, "Tinyos project," http://webs.cs.berkeley.edu/tos/.

[5] A. Kansal, A. A. Somasundara, D. D. Jea, M. B. Srivastava, and D. Estrin, "Intelligent fluid infrastructure for embedded networks," in *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*. ACM Press, 2004, pp. 111–124.

[6] W. Chen, N. Jain, and S. Singh, "Anmp: Ad hoc network management protocol," in *IEEE Journal on Selected Areas in Communications, Vol. 17, No.8, August*, 1999.

[7] C.-C. Shen, C. Srisatapornphat, and C. Jaikaeo, "An adaptive management architecture for ad hoc networks," in *IEEE Communications Magazine, February*, 2003.

[8] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *ACM/IEEE International Conference on Mobile Computing and Networks (MobiCom '00)*, 2000.

[9] L. B. Ruiz, J. M. Nogueira, and A. Loureiro, "Manna: A management architecture for wireless sensor networks," in *IEEE Communications Magazine, February*, 2003.

[10] G. Tolle and D. Culler, "Design of an application-cooperative management system for wireless sensor networks," in *Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN)*, 2005.

[11] S. Maden, M. Franklin, J. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," in *Symposium on Operating System Design and Implementation (OSDI 2002)*, Dec. 2002.

[12] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler, "Hood: A neighborhood abstraction for sensor networks," in *Proceedings of the Second ACM Conference on Mobile Systems, Applications, and Service (Mobisys)*, June 2004.

[13] M. Welsh and G. Mainland, "Programming sensor networks using abstract regions," in *Proceedings of the 1st symposium on networked system design and implementation (NSDI)*, March 2004.

[14] B. Greenstein, E. Kohler, and D. Estrin, "A sensor network application construction kit (snack)," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM Press, 2004, pp. 69–80.

[15] T. Liu and M. Martonosi, "Impala: a middleware system for managing autonomic, parallel sensor systems," in *PPoPP '03: Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM Press, 2003, pp. 107–118.

[16] sNMP, "Wireless sensor network management," http://www.research.rutgers.edu/∼bdeb/sensor_networks.html.

[17] F. Ye, G. Zhong, S. Lu, and L. Zhang, "Gradient broadcast: A robust data delivery protocol for large scale sensor networks," in *ACM Wireless Networks*, 2005.

[18] C. E. Perkins and E. M. Royer, "Ad hoc on-demand distance vector routing," in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[19] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to networked embedded systems," in *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*. ACM Press, 2003, pp. 1–11.

[20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns," Addison Wesley.

[21] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM Press, 2004, pp. 188–200.

[22] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*. ACM Press, 2003, pp. 126–137.

[23] "The vint project. the ucb/lbnl/vint network simulator - ns (version 2)," http://www.isi.edu/nsnam/ns/.

[24] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*. ACM Press, 1998, pp. 85–97.

[25] D. Tian and N. D. Georganas, "Energy efficient routing with guaranteed delivery in wireless sensor networks," in *Proceedings of the IEEE Wireless Communications and Networking Conference, 2003*.

[26] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*. ACM Press, 2003, pp. 134–146.

[27] J. Heidemann, F. Silva, and D. Estrin, "Matching data dissemination algorithms to application requirements," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*. ACM Press, 2003, pp. 218–229.

[28] V. Kawadia and P. Kumar, "A cautionary perspective on cross-layer design," in *IEEE Wireless Communications, February*, 2005.