Building Efficient Wireless Sensor Networks with Low-Level Naming*

John Heidemann† Fabio Silva† Chalermek Intanagonwiwat†
Ramesh Govindan† Deborah Estrin†‡ Deepak Ganesan†‡

† USC/Information Sciences Institute
4676 Admirality Way
Marina del Rey, CA, USA 90292

‡ Computer Science Department
University of California, Los Angeles
Los Angeles, CA, USA 90095

{johnh,fabio,intanago,govindan,estrin,gdeepak}@isi.edu

ABSTRACT

In most distributed systems, naming of nodes for low-level communication leverages topological location (such as node addresses) and is independent of any application. In this paper, we investigate an emerging class of distributed systems where low-level communication does not rely on network topological location. Rather, low-level communication is based on attributes that are *external* to the network topology and *relevant* to the application. When combined with dense deployment of nodes, this kind of named data enables *in-network processing* for data aggregation, collaborative signal processing, and similar problems. These approaches are essential for emerging applications such as sensor networks where resources such as bandwidth and energy are limited. This paper is the first description of the software architecture that supports named data and in-network processing in an operational, multi-application sensor-network. We show that approaches such as in-network aggregation and nested queries can significantly affect network traffic. In one experiment aggregation reduces traffic by up to 42% and nested queries reduce loss rates by 30%. Although aggregation has been previously studied in simulation, this paper demonstrates nested queries as another form of in-network processing, and it presents the first evaluation of these approaches over an operational testbed.

1. INTRODUCTION

In most distributed systems, naming of nodes for low-level communication leverages topological location (such as node addresses) and is independent of any application. Typically, higher-level, location-independent naming and communication is built upon these low-level communication primitives using one or more levels of (possibly distributed) binding services that map higher-level names to topological names and sometimes consider application-specific requirements.

An example of this is the Internet where IP addresses provide the low-level names suitable for routing. IP addresses are assigned topologically: the addresses for nodes that are topologically proximate are usually drawn from the same address prefix [18]. (By topology, we mean logical connectivity as distinct from physical geography.) This topological assignment is essential for scaling the routing system and was carried forward into IPv6 [30]. DNS provides a text-based hierarchical node naming system [26] that is implemented using IP. Above this system, the web and search engines provide a document and object naming system, and content distribution networks add geographic or application-level constraints. As an alternative, systems such as Jini [35] and INS [1] layer different approaches for resource discovery above IP for networks of devices.

In this paper, we investigate an emerging class of distributed systems where low-level communication does not rely on network topological location. Rather, low-level communication is based on names that are *external* to the network topology and *relevant* to the application; names can be based on capabilities such as sensor types or *geographic* location. Such an approach to naming allows two kinds of efficiencies. First, it eliminates the overhead of communication required for resolving name bindings. Second, because data is now self-identifying, it enables activation of application-specific processing inside the network, allowing data reduction near where data is generated.

These two benefits do not apply to the Internet as a whole, where, by comparison, bandwidth is plentiful, delay is low, and throughput (router processing capability) is the primary constraint. Technology trends suggest, however, that these conditions are *reversed* in wireless sensor networks. Sensor networks are predicated on the assumption that it will be feasible to have small form-factor devices containing significant memory resources, processing capabilities, and low-power wireless communication, in addition to several on-board sensors. In sensor networks processing

*This work was supported by DARPA under grant DABT63-99-1-0011 as part of the SCAADS project, NSF grant ANI-9979457 as part of the SCOWR project, and was also made possible in part due to support from Cisco Systems.

SOSP 2001, Lake Louise, Banff, Canada

time per bit communicated is plentiful (CPUs are fast and bandwidths low), but bandwidth is dear. For example, in one scenario Pottie and Kaiser observe that 3000 instructions could be executed for the same energy cost of sending a bit 100m by radio [29]. This environment encourages the use of computation to reduce communication. In that context, fewer levels of naming indirection and the use of in-network, application-specific message processing (as opposed to opaque packet forwarding) are essential to the design of sensor networks.

Our thesis, then, is that the resource constraints of wireless sensor networks can be better met by an *attribute-based* naming system with an *external frame of reference* than by traditional approaches. There have been many attribute-based naming schemes, but most build over an underlying topological naming scheme such as IP [28, 10, 6, 38, 4, 27, 1, 20, 22]. Multiple layers of naming may not be a bottleneck with a few or even tens of nodes, but the overhead becomes unreasonable with hundreds or thousands of nodes that vary in availability (due to movement and failures). However, constrained, application-specific domains such as sensor networks can profit by eliminating multiple layers and naming and routing data directly in application-level terms. Efficient attribute naming is based on external frames of reference such as pre-defined attributes and geography. Pre-defined sensor types reduce the levels of run-time binding and geographic-aided routing reduces resource consumption.

In addition to attribute-based naming, application-specific, *in-network processing* is essential in resource-constrained sensor networks. As suggested by the above trade-off between computation and communication, application-specific caching, aggregation, and collaborative signal processing should occur as close as possible to where the data is collected. Such processing depends on attribute-identified data to trigger application-specific filters, pre-defined attributes and data types to allow pre-deployment of these filters, and hop-by-hop processing of the data. This kind of processing is similar to Active Networks [34], but differs by operating in the constrained, bandwidth-poor environment of sensor networks where an integrated, application-specific solution is appropriate.

As an illustration of attribute-based naming and in-network processing in a sensor network, consider a wireless monitoring system with a mixture of light or motion sensors (constantly vigilant at low-power), and a few higher-power and higher-bandwidth sensors such as microphones or cameras. To conserve energy and bandwidth the audio sensors would be off (or not recording) at most times, except when triggered by less expensive light sensors. Instead this computation can be distributed throughout the network. Queries (user requests) are labeled with sensor type (audio or light) known to the system at design time. Queries diffuse through the network to be handled by nodes with matching sensors in the relevant geographic region. The application will hear from whatever relevant sensors respond. Moreover, the decision of one sensor triggering another can be moved into the network to be handled directly between the light and audio sensors. The alternative Internet-based architecture would have a central directory of active sensors and a central application that interrogates this database, monitors specific sensors, and then triggers others. Our goal is to eliminate the communication costs of maintaining this central information to provide more robust and long-lived networks in spite of changing communications, moving nodes, and limited battery power. (We explore exactly how these approaches work in Section 5 and quantify potential savings in Section 6.)

In this paper, we demonstrate that there exists a simple architecture that uses topology-independent naming for low-level communications to achieve flexible, yet highly energy efficient application designs. The key contributions of this work are therefore:

- Identifying the building-blocks of this architecture, specifically an *attribute-based naming scheme with flexible matching rules* grounded in a shared framework of attributes (such as sensor types and geography).
- Showing how this approach to naming enables *application-specific, in-network processing* such as localized data aggregation, and to quantify these benefits in a running system.

In previous work [23], we have discussed the low-level communication primitives that constitute directed diffusion. This work focused on understanding the design space of the network protocols underlying directed diffusion. It also evaluated their performance through simulation, finding that scalability is good as numbers of nodes and traffic increases. However, this work did not develop the software architecture necessary for realizing attributes and in-network processing in an operational system (for example, it employed a simplified attribute scheme and hard-coded aggregation methods). In addition, simulations necessitate approximating environmental effects such as radio propagation, and many parameters of those simulations were not set to match the sensor networking hardware that is only now becoming available. By contrast, this paper evaluates the design questions concerning naming and in-network processing encountered in deploying a sensor network, and it presents the first experimental results of data diffusion in a testbed (reflecting the details of an implementation such as non-idealized radios, propagation, MAC protocols, etc.).

Numerous early systems have developed attribute-based naming systems, for general use [28, 10, 6], as an approach to software design [9, 4, 27, 17, 25] and for sensor networks [1, 22]. Our work is unique in that it replaces rather than augments the underlying networking routing layers, and that it provides matching rules that allow efficient implementation and yet are expressive enough to cover a wide range of applications, and provides in-network processing.

2. RELATED WORK

Our work builds on prior work in attribute-based naming, in-network processing, and sensor networks.

2.1 Attribute-based naming systems

There has been a large amount of work on attribute-based naming, both for general purpose use over Internet-style networks, for special domains (such as the web), and as an internal structuring mechanism for services.

Research and industry have developed numerous attribute-based naming systems layered on top of general-purpose networks. Univers and yellow-pages naming at the University of Arizona [6, 28] were designed to provide service discovery for groups of computers (for example, print to an unloaded postscript-capable printer). Like our work, they include attributes and operators, but they build over standard Internet protocols for communications. Commercial attribute-based naming systems such as X.500 [10] and LDAP [38] also operate over Internet or Internet-like routing and provide a primarily hierarchical organization. Dependence on IP-level addressing and routing limits adds substantial overhead when applying these systems to highly resource-constrained environments such as sensor networks. (For example, some approaches to service location for smart spaces require services for IP assignment, IP-level routing, host name lookup, and service registration and lookup.) With end-to-end processing only, these systems also do not provide in-network processing.

As an alternative to providing attribute-based naming for end-user use, several systems have proposed attribute-based communications for structuring distributed systems. Linda proposed structuring distributed programs using several CPUs around an attribute-indexed common memory called a tuple space [9]. For the S/Net implementation this was the basic communication mechanism, but proposed implementations assume uniform and rapid communications between all processors. Later systems such as ISIS [4] and the Information Bus [27] provide a "publish and subscribe" approach where information providers publish information and clients subscribe to attribute-specified subsets of that information. These systems are designed to be robust to failure, but again assume reasonably fast, plentiful, and expensive communications between nodes. These approaches are not directly applicable to resource-constrained sensor networks. They do not use application-specific, in-network processing since all processes are reasonably close to each other; when they do use processing (such as at a wide-area gateway) it is manually configured.

More specific still is work that proposes attribute-based primitives as solutions to specific problems. SRM first suggested using named data as the fundamental data unit for reliable multicast communication, and it demonstrated this approach with a distributed whiteboard [17]. Our work is inspired by these approaches, but it differs by providing a wider range of matching operators (rather than just equality), adding in-network processing to leverage CPU-communications trade-offs for sensor networks, and operating directly over low-level (hop-by-hop) communications protocols instead of the Internet multicast infrastructure.

2.2 In-network processing

Recent work in active networks [34] and active services [2] has examined ways to provide in-network processing for the Internet. Sample applications include information transcoding, network monitoring, and caching. This work is built over an Internet-like infrastructure, often augmented with an extended run-time environment, and assumes nodes are individually addressable. We instead build directly over hop-by-hop communications primitives and identify data instead of nodes. Our work differs from active services in that we assume that communications costs between nodes vary greatly while currently proposed active services assume roughly equivalent distances between all service-providing nodes. We differ from active networks primarily in the target domain: we target sensor networks where bandwidth is limited, energy is expensive, and compute power is comparatively plentiful and inexpensive. Instead, active networks typically considers Internet-like domains where bandwidth is plentiful, the ratio of compute power to bandwidth is much lower, and energy is not an issue. All of these approaches distribute application-specific code throughout the network, raising questions about code safety and portability. These problems are not central to some sensor networks (such as those that are devoted to a single application), but more complex networks would benefit from active-networks-style execution environments to support in-place upgradability.

Recent work on adaptive web caching [25] and peer-to-peer file sharing systems such as Freenet [12] explore application-specific, hop-by-hop processing. Unlike active networks and our work, these approaches emphasize protocols designed for a particular application. In addition, our work runs directly over hop-by-hop communication rather than over a virtual network layered over the Internet.

2.3 Sensor-network-specific systems

Sensor networking research has seen increasing activity in the last few years, with advances in sensor node and radio hardware [33, 29]. This work has been instrumental in clarifying the trade-off between computation and communication and the need for in-network processing. Our focus on in-network processing is motivated by this work. This work is however based on topographically-addressed sensor nodes; the primary difference in our work is the use of attribute-based naming for structure and data diffusion for communication.

Internet ad hoc routing (Broch et al. survey several protocols [7] such as DSR and AODV) can also be used in sensor networks. Since ad hoc routing recreates IP-style addressing, it would require some kind of directory service to locate sensors, unlike our approach where they are named by attributes. Ad hoc routing does not support in-network processing.

Jini is an example of a resource discovery system built over Internet protocols [35]. It provides a directory service and uses Java to distribute processing to user nodes, making it well suited to a local-area network with high bandwidth and multicast. By contrast, we distribute the directory across the network and allow application-specific processing at intermediate system nodes, addressing problems of resource-constrained, multi-hop wireless networks. The Ninja Service Discovery Service [15] locates XML-named objects through a network of collaborating servers but again targets high bandwidth local-area resources.

The Piconet work has presented fundamental advances in energy-conserving network communications for networks of devices [3]. Their work focuses on static hierarchies of networked devices, concentrators, and hosts. While similar to our tiered architecture with full and micro-diffusion, they do not consider attribute-named data or dynamic in-network processing.

SPIN evaluates several variants of flooding for wireless sensor networks [20]. Data in SPIN is identified by application-specific metadata that appears to assume individual sensors are addressable. We instead use attributes to name data alone; globally unique identifiers are not used. SPIN does not consider application-specific in-network processing.

The Intentional Naming System is an attribute-based name system operating in an overlay network over the Internet [1]. Its use of attributes as a structuring mechanism and a method to cope with dynamically locating devices is similar to our approach in motivation and mechanism. The primary difference is that we assume that attribute-based communication (data diffusion) is the basic communications primitive (above hop-by-hop messaging), while they construct an overlay network over an IP-based Internet. Architecturally this implies that we distribute name matching across many small communications nodes while they manage names at a few resolvers that cooperatively manage parts of the namespace. Finally, the details of matching are different in the two systems. Their work provides a sophisticated hierarchical attribute matching procedure. Our approach is much more modest by comparison (targeting smaller embedded devices) but adds comparative operators in addition to equality.

LEACH analyzes the performance of cluster-based routing mechanism with in-network data compression [19]. They emphasize how intermediate-range communication via cluster-heads and how compression can reduce energy consumption. Their in-network compression is one example of the kind of in-network processing that we would like to support. They do not specify how flows and opportunities for aggregation would be activated, while our work focuses on the naming mechanisms that allow such activity.

DataSpace describes an attribute based naming mechanism for querying physical objects that produce and store local data [22]. The DataSpace is divided into smaller administrative and logical *datacubes*, which are logically grouped into *dataflocks*. Dataflocks are addressed at the network level through IPv6 multicast addresses that correspond to their geographic coordinates, and their values for certain attributes that serve as network indices. Query results may involve aggregation of more specific queries addressed to *sub-datacubes*. At a high-level their naming approach is similar to ours, but instead of mapping attributes and geometry to a very large number of multicast groups we route directly on attributes themselves without this indirection. In addition, they do not explore in-network processing.

The COUGAR device database system proposes distributing database queries across a sensor network as opposed to moving all data to a central site [5]. Sensor data is represented as an Abstract Data Type attribute, the public interface to which corresponds to specific signal processing functions supported by a sensor type. They then perform joins or aggregation in the network as specified by a centrally computed query plan. Their work is common with ours in its emphasis on in-network processing, and our study of nested queries (Section 5.2) was inspired by their work. The primary difference between their work in ours is how placement of in-network processing is determined. We emphasize the use of filters and nested queries to enable either ad-hoc or sensor-specific placement of in-network processing, while COUGAR centrally translates the query and assigns processing to the distributed system, incurring overhead to centrally collect network information for query optimization.

Declarative Routing from MIT's Lincoln Labs is closest to our work [14]. The publish/subscribe-oriented API we use was defined in collaboration with them [13] and they have developed an independent implementation. The primary difference between their work and ours is our focus on in-network processing. We evaluate their work more completely in Section 4.2.

3. ARCHITECTURE

Our communications architecture is based on three components: directed diffusion, matching rules, and filters. *Directed diffusion* is used to disseminate information in the distributed system. Data is managed as a list of *attribute-value-operation* tuples. *Matching rules* identify when data has arrived at its destination, or if intermediate *filters* should process the data. This approach to naming comes together to provide an external framework relevant to the application. These components balance the generic services of diffusion and matching rules with application-provided attributes and filters. We next describe each of these components.

3.1 Directed Diffusion

Directed diffusion is a data communication mechanism for sensor networks [23]. Data sources and sinks use attributes to identify what information they provide or are interested in. The goal of directed diffusion is to establish efficient n-way communication between one or more sources and sinks. Directed diffusion is a data-centric communication paradigm that is quite different from host-based communication in traditional networks. To describe the elements of diffusion, we take the simple example of a sensor network designed for tracking animals in a wilderness refuge.

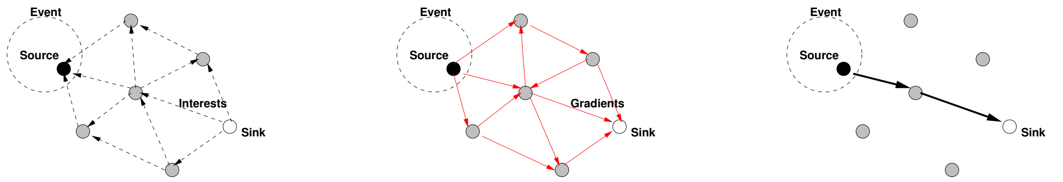
Suppose that a user in this network would like to track the movement of animals in some remote sub-region of the park. In directed diffusion, this tracking task represents an *interest*. An interest is a list of attribute-value pairs that describe a task using some task-specific naming scheme (we describe the details of these attributes in the next section). Intuitively, attributes describe the data that is desired by specifying sensor types and possibly some geographic region. They are then used to identify and contact all relevant sensors. We use the term *sink* to denote the node that originates an interest and therefore is the destination of data.

The interest is propagated from neighbor-to-neighbor towards sensor nodes in the specified region. A key feature of directed diffusion is *that every sensor node is task-aware*—by this we mean that nodes store and interpret interests, rather than simply forwarding them along. In our example, each sensor node that receives an interest remembers which neighbor or neighbors sent it that interest. To each such neighbor, it sets up a *gradient*. A gradient represents both the direction towards which data matching an interest flows, and the status of that demand (whether it is active or inactive and possibly the desired update rate). After setting up a gradient, the sensor node redistributes the interest to its neighbors. When the node can infer where potential sources might be (for example, from geographic information or existing similar gradients), the interest can be forwarded to a subset of neighbors. Otherwise, it will simply broadcast the interest to all of its neighbors.

When a sensor node that matches the interest is found, the application activates its local sensors to begin collecting data. (Prior to activation we expect the node's sensors would be in a low-power mode). The sensor node then generates *data* messages matching the interest. In directed diffusion, data is also represented using an attribute-based naming scheme. A sensor node that generates such an event description is termed a *source*.

Data is cached at intermediate nodes as it propagates toward sinks. Cached data is used for several purposes at different levels of diffusion. The core diffusion mechanism uses the cache to suppress duplicate messages and prevent loops, and it can be used to preferentially forward interests. (Since the diffusion core is primarily interested in an exact match, as an optimization, hashes of attributes can be computed and compared rather than complete data.) Cached data is also used for application-specific, in-network processing. For example, data from detections of a single object by different sensors may be merged to a single response based on sensor-specific criteria.

The initial data message from the source is marked as *exploratory* and is sent to all neighbors for which it has matching gradients. If the sink has multiple neighbors, it chooses to receive subsequent data messages for the same interest from a preferred neighbor (for example, the one which delivered the first copy of the data message). To do this, the sink *reinforces* the preferred neighbor, which, in turn reinforces its preferred upstream neigh-



(a) Interest propagation

(b) Initial gradients set up

(c) Data delivery along reinforced path

Figure 1: A simplified schematic for directed diffusion.

bor, and so on. Finally, if a node on this preferred path fails, sensor nodes can attempt to *locally repair* the failed path. The sink may also *negatively reinforce* its current preferred neighbor if another neighbor delivers better (lower latency) sensor data. This negative reinforcement propagates neighbor-to-neighbor, removing gradients and tearing down and existing path if it is no longer needed [23]. Negative reinforcements suppress loops or duplicate paths that may arise due to network dynamics.

After the initial exploratory data message, subsequent messages are sent only on reinforced paths. Periodically the source sends additional exploratory data messages to adjust gradients in the case of network changes (due to node failure, energy depletion, or mobility), temporary network partitions, or to recover from lost exploratory messages. Recovery from data loss is currently left to the application. While simple applications with transient data (such as sensors that report their state periodically) need no additional recovery mechanism, we are also developing retransmission scheme for applications that transfer large, persistent data objects.

Even this simplified description points out several key features of diffusion, and how it differs from traditional networking. First, diffusion is data-centric; all communication in a diffusion-based sensor network uses interests to specify named data. Second, all communication in diffusion is neighbor-to-neighbor or hop-by-hop, unlike traditional data networks with end-to-end communication. Every node is an "end" in a sensor network. A corollary to this previous observation is that there are no "routers" in a sensor network. Each sensor node can interpret data and interest messages. This design choice is justified by the task-specificity of sensor networks. Sensor networks are not general-purpose communication networks. Third, nodes do not need to have globally unique identifiers or globally unique addresses for regular operation. Nodes, however, do need to distinguish between neighbors. Fourth, because individual nodes can cache, aggregate, and more generally, process messages, it is possible to perform coordinated sensing close to the sensed phenomena. It is also possible to perform in-network data reduction, thereby resulting in significant energy savings. Finally, although our example describes a particular usage of the directed diffusion paradigm (a query-response type usage, see Figure 1), the paradigm itself is more general than that; we discuss several other example applications in Section 5.

3.2 Attribute Tuples and Matching Rules

Diffusion messages and application interests are composed of attribute-value-operation tuples. Attributes are identified by unique *keys* drawn from a central authority. (In practice we implement these as simple 32-bit numbers and assume out-of-band coordination of their values, just as Internet protocol numbers are assigned.) Attributes implicitly have a data format (integers and floating point values of different sizes, strings, and uninterpreted binary data are currently supported).

one-way match:

```
given two attribute sets A and B
for each attribute a in A where a.op is a formal {
    matched = false
    for each attribute b in B where a.key = b.key and b.op is an actual
        if a.val compares with b.val using a.op, then matched = true
    if not matched then return false (no match)
}
return true (successful one-way match)
```

Figure 2: Our one-way matching algorithm.

The operation field defines how data messages and interests interact. Operations are the usual binary comparisons (EQ, NE, LE, GT, LE, GE, corresponding to equality, inequality, less than, etc.), "EQ_ANY" (which matches anything), and IS. "IS" allows users to specify an *actual* (literal or bound) value, while all the other operations specify *formal* (a comparison or unbound) parameters for comparison. A *one-way match* compares all formal parameters of one attribute set against the actuals of the others (Figure 2). Any formal parameter that is missing a matching actual in the other attribute set causes the one-way match to fail (for example, "confidence GT 0.5" must have an actual such as "confidence IS 0.7" and would not match "confidence IS 0.3", "confidence LT 0.7", or "confidence GT 0.7"). Two sets of attributes have a *complete match* if one-way matches succeed in both directions. In other words, attribute sets A and B match if the one-way match algorithm succeeds from both A to B and B to A.

This matching style is similar to the rules used in other attribute-based languages (for example, Linda [9] and INS [1]), but we add two-way matching and a range of operators in addition to equality. When multiple attributes and operators are present they are effectively "anded" together; all formals must be satisfied for a match to be successful. This approach strikes a balance between ease of implementation and flexibility. The simple bounded set of operators can be implemented in tens of lines of code and yet supports, for example, rectangular regions.

To see how diffusion and attribute matching interact, we continue the example from Section 3.1 where a user asks a sensor network to track four-legged animals. The user's query translates into an interest with the attributes (type EQ four-legged-animal-search, interval IS 20ms, duration IS 10 seconds, x GE –100, x LE 200, y GE 100, y LE 400). Also, an implicit "class IS interest" attribute is added to identify this message as an interest (as opposed to data). This interest specifies five conditions: detection of animals in a particular region specified by a rectangle. It also provides information about how frequently data should be returned and how long the query should last.

Sensors in the network are programmed with animal search routines (either by pre-programming at deployment time or by downloading mobile code). Such sensors would watch for interests in animals by expressing *interests about interests* with attributes (class EQ interest, type IS four-legged-animal-search, x IS 125, y IS 220). When the user's interest arrives at the sensor it would activate its sensor using the parameters provided (duration and interval) and reply if it detects anything.

When the sensor detects something the data message would include attributes (type IS four-legged-animal-search, instance IS elephant, x IS 125, y IS 220, intensity IS 0.6, confidence IS 0.85, timestamp IS 1:20, class IS data). This message satisfies the original interest. It encodes as attributes additional information about what was seen and what confidence the sender has in its detection.

This example illustrates the details of a specific query. It shows how named data provides a convenient way of encoding information, and how geometry and well-known attributes allow simple matching rules work for this application. Although this example uses several attributes, some applications may use only a subset of these methods, omitting geographic constraints (in a small sensor network) or using a single attribute (when there is only one sensor type). We have found that these primitives provide good building blocks for a range of applications; we describe these in Section 5.

Although matching is reasonably powerful, it does not perfectly cover all scenarios or tasks. Simple matching in these cases can approximate what is required, and application-specific code can further refine the choice. For example, perfect rectangles aligned with the coordinate system are insufficient to describe arbitrary geometric shapes. Non-rectangular shapes can be accomplished either by multiple queries, or by using the smallest bounding rectangle and having the application ignore requests inside the rectangle but outside the required region. Similarly, applications can use general attributes that are clarified with sub-attributes or parameters (type IS animal-search, subtype IS four-legged). Filters (described next) also allow applications to influence processing.

3.3 Filters

Filters are our mechanism for allowing application-specific code to run in the network and assist diffusion and processing. Applications provide filters before deployment of a sensor network, or in principle filters could be distributed as mobile code packages at run-time. Filters register what kinds of data they handle through matching; they are then triggered each time that kind of data enters the node. When invoked, a filter can arbitrarily manipulate the message, caching data, influencing how or where it is sent onward, or generating new messages in response. Filters have access to internal information about diffusion, including gradients and lists of neighbor nodes.

Filters are typically used for in-network aggregation, collaborative signal processing, caching, and similar tasks that benefit from control over data movement. In addition to these applications, we have found them very useful for debugging and monitoring.

Continuing our example, a filter can be used to suppress concurrent detections of four-legged animals from different sensors. It would register interest in detection interests and data with attributes (type IS four-legged-animal-search). It could then record what the desired interval is, then allow exactly one reply every interval units of time, suppressing replies from other sensors. A more sophisticated filter could count the number of detecting sensors and add that as an additional attribute, or it could generate some kind of aggregate "confidence" rating in some application-specific manner. In this example filtering may discard some data, but by reducing unnecessary communication it will greatly extend the system's operational lifetime.

We describe some application of filters in Section 5, and quantify the benefits of aggregation in one scenario in Section 6.1.

4. IMPLEMENTATIONS

There are currently three implementations of all or part of this architecture. Our current reference implementation *SCADDS diffusion version 3* provides all components. MIT-Lincoln Labs has implemented "declarative routing" that provides attribute matching but no filters [14]. Both of these implementations run on Linux on desktop PCs and PC/104-based sensor nodes [11] (embedded x86 machines, ours with a 66MHz CPU and 16MB of RAM and flash disk, Figure 3(a)), and on WINSng 1.0 sensor nodes [29] (Windows-CE-based nodes with custom low-power radios, Figure 3(b)). We have also implemented *micro-diffusion*, a bare subset of these services designed to run on Motes with tiny 8-bit processors and only 8KB of memory (Figure 3(c)).

Source code to our implementations can be found on our web site `http://www.isi.edu/scadds`.

All of our implementations build upon a simple radio API that supports broadcast or unicast to immediate neighbors. Neighbors must have some kind of identifier, but it is not required to be persistent. We can use persistent identifiers (for example, Ethernet MAC addresses) or operate with ephermally assigned identifiers [16].

4.1 Basic diffusion APIs

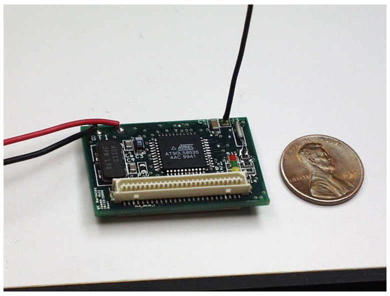
Our reference implementation includes C++ *Network Routing* APIs summarized in Figure 4 (see [13] for a complete specification and example source code). The APIs define a publish/subscribe approach to data handling. To receive data, nodes *subscribe* to particular set of attributes. A subscription results in interests being sent through the network and sets up gradients. A callback function is then invoked whenever relevant data arrives at the node.

Applications that generate information *publish* that fact, and then *send* specific data. The attributes specified in the publish call must match the subscription. If there are no active subscriptions, published data does not leave the node. As a further optimization sensor nodes may wish to avoid generating data that has no takers. In this case the application would *subscribe for subscriptions* and would be informed when subscriptions arrive or terminate.

Filter-specific APIs are shown in Figure 5. A filter is primarily a callback procedure (the `cb` specified in addFilter) that is called when matching data arrives. Rather than operate only on attribute vectors, filters are given direct access to messages that include identifiers for the previous and next immediate destinations. We

(a) Our PC/104 node

(b) WINSng 1.0 node



(c) UCB Rene Mote

Figure 3: Diffusion operational platforms.

```
handle NR::subscribe(NRAttrVec *subscribeAttrs,
                     const NR::Callback * cb);
int NR::unsubscribe(handle subscription_handle);
handle NR::publish(NRAttrVec *publishAttrs);
int NR::unpublish(handle publication_handle);
int NR::send(handle publication_handle,
             NRAttrVec *sendAttrs);
```

Figure 4: Basic diffusion API.

```
handle addFilter(NRAttrVec *filterAttrs,
          int16_t priority, FilterCallback *cb);
int NR::removeFilter(handle filter_handle);
void sendMessage(Message *msg, handle h,
                 int16_t agent_id = 0);
void sendMessageToNext(Message *msg, handle h);
```

Figure 5: Filter APIs.

are currently evaluating using this additional level of control to optimize diffusion, for example using geographic information to avoid flooding exploratory interests. We expect these interfaces to be extended as we gain more experience with how filters are used and what information they require.

Finally, these APIs have been designed to favor an event-driven programming style, although they have been successfully used in multi-threaded environments such as WINSng 1.0. We have targeted event-driven programming to avoid synchronization errors and to avoid the memory and performance overheads of multi-threading. Evidence is growing that event-driven software is well suited to embedded programming, particularly on very memory-constrained platforms [21].

Also we allow filters and applications to run in the same or different memory address spaces from each other and the diffusion core. Single-address space operation is necessary for very small sensor nodes that lack memory protection and as a performance optimization. Multiple address spaces may be desired for robustness to isolate filters of different applications from each other.

4.2 MIT-LL declarative routing

Dan Coffin helped define the basic diffusion APIs (Figure 4 and [13]) and developed an independent implementation in MIT-Lincoln Lab's Declarative Routing system [14]. In principle all applications that do not depend on filters will run over either implementation. This level of portability has been demonstrated with Cornell's query proxy [5] that runs over both implementations.[1]

Declarative routing and data diffusion are far more similar than they are different. Both name data rather than end-nodes. Differences are in how routes and transmission are optimized, both by applications and the core system. The primary difference is that declarative routing does not include filters to allow applications to directly influence diffusion. We see filters as a critical necessary component to enable general in-network data processing. Second, Lincoln Lab's declarative routing includes direct support for energy and geography-aided routing so that routes are selected to avoid energy-poor nodes and generally move "towards" a target geographic area. In our current implementation interests and exploratory messages are flooded through the network before gradients are set up for direct communication. We are currently exploring using filters to optimize diffusion (avoiding flooding) with geographic information [39].

4.3 Micro-diffusion

Micro-diffusion is a subset of our approach implemented on very small processors (8-bit CPU, 8KB memory). It is distinguished by its extremely small memory footprint and a complementary approach for deployment to our full system.

Micro-diffusion is a subset of our full system, retaining only gradients, condensing attributes to a single tag, and supporting only limited filters. As a result it adds only 2050 bytes of code and 106 bytes of data to its host operating system. (By comparison, our full system requires a daemon with static sizes of 55KB code, 8KB data, and a library at 20KB code, 4KB data.) Micro-diffusion is implemented as a component in TinyOS [21] that adds 3250B code and 144B of data (including support for radio and a photo sensor), so the entire system runs in less than 5.5KB of memory. Micro-diffusion is statically configured to support 5 active gradients and a cache of 10 packets of the 2 relevant bytes per packet.

[1] No changes were required to our diffusion implementation, although the port required one change to the application to accommodate a case where MIT's implementation was less strict about attribute matching.

Although reduced in size, the logical header format is compatible with that of the full diffusion implementation and we are implementing software to gateway between the implementations. Although we do not currently provide filters in micro-diffusion, they are an essential component of enabling in-network aggregation in diffusion, and we plan to add them. We intend to leverage on the ability to reprogram motes over the air [21] to program filters dynamically.

Motes and micro-diffusion can be used in regions where there is need for dense sensor distribution, such as distributing photo sensors in a room to detect change in light or temperature sensors for fine grained sensing. They provide the necessary sensor data processing capability, with the ability to use diffusion to communicate with less resource-constrained nodes (for example, PC/104-class nodes). Motes can also be used to provide additional multi-hop capability under adverse wireless communication conditions.

We thus envisage deployment of a tiered architecture with both larger and smaller nodes. Less resource-constrained nodes will form the highest tier and act as gateways to the second tier. The second tier will be composed of motes connected to low-power sensors running micro-diffusion. Most of the network "intelligence" is programmed into the first tier. Second-tier nodes will be controlled and their filters programmed from these more capable nodes.

4.4 Implementation discussion

We draw two observations from our experiences with these implementations. First, the range of diffusion implementations suggests that both the ideas and the code are portable since there are three independent implementations (our main implementation, micro-diffusion, and MIT-LL's declarative routing) and our primary implementation runs on multiple platforms (PC/104s and WINSng 1.0 as of June 2001, with ports in progress to two new radios and platforms). The requirements for diffusion are quite modest in terms of CPU speed (a 15MHz 32-bit processor is sufficient), memory (a few megabytes supports diffusion, an OS, and applications), and radio (10–20kb/s bandwidth is sufficient). Several low-power radio designs have packet sizes as small as 30B. We require moderate size packets (100B or more) and use code for fragmentation and reassembly when necessary. Second, micro-diffusion demonstrates that it is possible to implement a subset of diffusion on an embedded processor. A common preconception is that fully custom protocols are needed for embedded systems; these observations suggest that use of diffusion should not be precluded due to size or complexity.

5. APPLICATION TECHNIQUES FOR SENSOR NETWORKS

We next consider application techniques in more detail. These techniques illustrate how topology-independent low-level naming and in-network processing can be used to build efficient applications for sensor networks. The first approach we examine is filter-driven data aggregation, an example of how in-network processing can reduce data traffic to conserve energy. We also consider two approaches to provide nested queries where one sensor cues another. Finally, we briefly describe several other applications that have been implemented.

5.1 In-network data aggregation

An anticipated sensor application is to query a field of sensors and then take some action when one or more of the sensors is activated. For example, a surveillance system could notify a biologist if an animal enters a region. Coverage of deployed sensors will overlap to ensure robust coverage, so one event will likely trigger multiple sensors. All sensors will report detection to the user, but communication and energy costs can be reduced if this data is aggregated as it returns to the user. Data can be aggregated to a binary value (there was a detection), an area (there was a detection in quadrant 2), or with some application-specific aggregation (seismic and infrared sensors indicate 80% chance of detection).

Although details of aggregation can be application-specific, the common systems problem is the design of mechanisms for establishing data dissemination paths to the sensors within the region, and for aggregating responses. Consider how one might implement this kind of data fusion in a traditional network with topologically-assigned low-level node names. First, in order to determine which sensors are present in a given region, a binding service must exist which, given a geographical region, lists the node identifiers of sensors within that region. Once these sensors are tasked, an election algorithm must dynamically elect one or more network nodes to aggregate the data and return the result to the querier.

Instead, our architecture allows us to realize this using *opportunistic* data aggregation. Sensor selection and tasking is achieved by naming nodes using geographic attributes. As data is sent from the sensors to the querier, intermediate sensors in the return path identify and cache relevant data. This is achieved by running application-specific filters. These intermediate nodes can then suppress duplicate data by simply not propagating it, or they may slightly delay and aggregate data from multiple sources. We are also experimenting with influencing the dynamic selection of aggregation points to minimize overall data movement.

Opportunistic data aggregation benefits from several aspects of our approach. Filters provide a natural approach to inject application-specific code into the network. Attribute naming and matching allow these filters to remain inactive until triggered by relevant data. A common attribute set means that filters incur no network costs to interact with directory or mapping services.

In prior work we analyzed the performance of diffusion with and without aggregation through simulation [23]. In Section 6.1 we evaluate our implementation of this over real sensor nodes and validate our initial results with laboratory tests.

5.2 Nested queries

Real-world events often occur in response to some environmental change. For example, a person entering a room is often correlated with changes in light or motion, or a flower's opening with the presence or absence of sunlight. Multi-modal sensor networks can use these correlations by triggering a secondary sensor based on the status of another, in effect nesting one query inside another. Reducing the duty cycle of some sensors can reduce overall energy consumption (if the secondary sensor consumes more energy than the initial sensor, for example as an accelerometer triggering a GPS receiver) and network traffic (for example, a triggered imager generates much less traffic than a constant video stream). Alternatively, in-network processing might choose the best application of a sparse resource (for example, a motion sensor triggering a steerable camera).

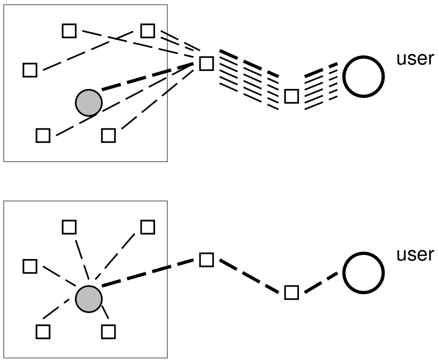


Figure 6: Two approaches to implementing nested queries. Squares are initial sensors, gray circles are triggered sensors, and the large circle is the user. Thin dashed lines represent communication to initial sensors; bold lines are communication to the triggered sensor.

Figure 6 shows two approaches for a user to cause one sensor to trigger another in a network. In both cases we assume sensors know their locations and not all nodes can communicate directly. Part (a) shows a direct way to implement this: the user queries the initial sensors (small squares), when a sensor is triggered, the user queries the triggered sensor (the small gray circle). The alternative shown in part (b) is a nested, two-level approach where the user queries the triggered sensor which then sub-tasks the initial sensors. This nested query approach grew out of discussions with Philippe Bonnet and embedded database query optimization in his COUGAR database [5].

The advantage of a nested query is that data from the initial sensors can be interpreted directly by the triggered sensor, rather than passing through the user. In monitoring applications the initial and triggered sensors would often be quite close to each other (to cover the same physical area), while the user would be relatively distant. A nested query localizes data traffic near the triggering event rather than sending it to the distant user, thus reducing network traffic and latency. Since energy-conserving networks are typically low-bandwidth and may be higher-latency, reduction in latency can be substantial, and reductions in aggregate bandwidth to the user can mean the difference between an overloaded and operational network. The challenges for nested queries are how to robustly match the initial and triggered sensors and how to select a good triggered sensor if only one is desired.

Implementation of direct queries is straightforward with attribute-addressed sensors. The user subscribes to data for initial sensors and when something is detected he requests the status of the triggered sensor (either by subscribing or asking for recent data). Direct queries illustrate the utility of predefined attributes identifying sensor types. Diffusion may also make use of geography to optimize routing.

Nested queries can be implemented by enabling code at each triggered sensor that watches for a nested query. This code then sub-tasks the relevant initial sensors and activates its local triggered sensor on demand. If multiple triggered sensors are acceptable but there is a reasonable definition of which one is best (perhaps, the most central one), it can be selected through an election algorithm. One such algorithm would have triggered sensors nominate themselves after a random delay as the "best", informing their peers of their location and election (this approach is inspired by SRM repair timers [17]). Better peers can then dispute the claim. Use of location as an external frame of reference defines a best node and allows timers to be weighted by distance to minimize the number of disputed claims.

In Section 6.2 we evaluate nested queries with experiments in our testbed.

5.3 Other applications

In addition to these approaches we have explored at ISI, our system has been used by several other research efforts.

Researchers at Cornell have used our system to provide communication between an end-user database and application that represents and visualizes a sensor field and query proxies in each sensor node [5]. This application used attributes to identify sensors running query proxies and to pass query byte-codes to the proxies. They also originated the idea of using a nested approach for nested queries. Future work includes understanding what network information is necessary for database query optimization and alternative approaches for nested queries.

Researchers at BAE Systems and Pennsylvania State University have used our system for collaborative signal processing. BAE systems contributed signal processing code and systems integration, while PSU provided sensor fusion algorithms [8]. The combined system used our system to communicate data between sensors using named data and diffusion. At the time our filter architecture was not in place; interesting future work is to evaluate how sensor fusion would be done as a filter.

6. EVALUATION

The approaches described in this paper are useful if they can be efficiently implemented and improve the energy-efficiency of distributed systems such as sensor nets. In Section 5 we described several applications that employ these techniques. In this section, we measure the benefits of aggregation and nested queries and verify raw matching performance.

6.1 Aggregation benefits

In Section 5.1, we argued that it is relatively easy to build sensor network applications using attribute-based naming, and in-network filters. In earlier work, we have observed that in-network aggregation is important to the performance of data diffusion [23]. In this section, we validate these results with an actual implementation of a simple surveillance application using attribute-based names and filters.

We examined in-network aggregation in our testbed of 14 PC/104 sensor nodes distributed on two floors of ISI (Figure 7). These sensors are connected by Radiometrix RPC modems (off-the-shelf, 418 MHz, packet-based radios that provide about 13kb/s throughput) with 10dB attenuators on the antennas to allow multi-hop communications in our relatively confined space. The exact topology varies depending on the level of RF activity, and the network is typically 5 hops across.

To evaluate the effect of aggregation we placed a sink on one side of the topology ("D" at node 28) and then placed data sources on the other side ("S" at nodes 25, 16, 22, and 13), typically 4 hops apart. All sources generate events representing the detection of

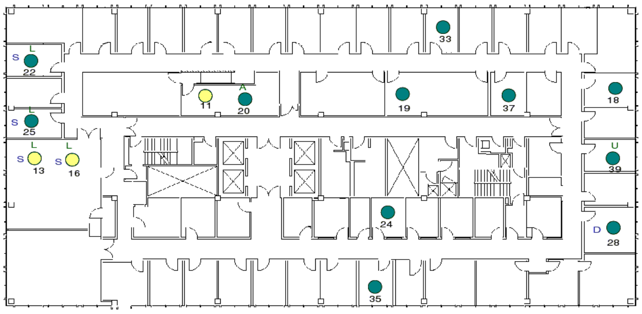


Figure 7: Node positions in our sensor testbed. Light nodes (11, 13, 16) are on the 10th floor; the remaining dark nodes are on the 11th floor. Radio range varies greatly depending on node position, but the longest stable link was between nodes 20 and 25.

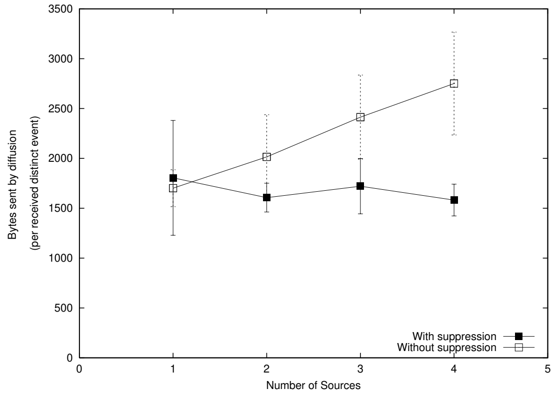


Figure 8: Bytes sent from all diffusion modules, normalized to the number of distinct events, for varying numbers of sources.

some object at the rate of one event every 6 seconds. For experiment repeatability events are artificially generated, rather than taken from a physical sensor and signal processing. Each event generates a 112 bytes message and is given sequence numbers that are synchronized at experiment start.[2] All nodes were configured with aggregation filters that pass the first unique event and suppress subsequent events with identical sequence numbers. Although this scenario abstracts some details of a complete sensor network (for example, real signal processing may have different sensing delays), we believe it captures the essence of the networking component of multi-sensor aggregation.

We would like to compare the energy expended per received event. Unfortunately, we cannot measure that directly for two reasons. First, we do not have hardware to directly measure energy consumption in a running system. Second, we have previously observed that choice of MAC protocol can completely dominate energy measurements. In low power radios, MAC protocols that do not sleep periodically are dominated by the amount of time spent listening, regardless of choice of protocol. Thus energy-conscious protocols like PAMAS [32] or TDMA are necessary for long-lived sensor networks. We are currently experimenting with power-aware MAC approaches.

Although we currently cannot measure energy consumption on an appropriate MAC, we can estimate the effectiveness of reducing traffic for MACs with different duty cycles. A simple model of energy consumption is:

$$p_a = dp_\ell t_\ell + p_r t_r + p_s t_s$$

where p and t define the relative power and time spent listening, receiving, and sending and d is defined as the required listen duty cycle (the fraction of time the radio must be listening to receive all traffic destined to it). We found our sensor network contained pockets of severe congestion, but in the aggregate, radios listen:receive:send times were about 1:3:40. Relative energy consumption of listen:receive:send has been measured at ratios from 1:1.05:1.4 to 1:2:2.5 [37]. For simplicity, assume energy consumption ratios of 1:2:2. With these parameters, energy usage for nodes with a duty cycle of 1 are completely dominated by energy spent listening. At duty cycle of 22% half of the energy is spent listening. Duty cycles of 10% begin to be dominated by send cost. Duty cycle for most radios today is 100%, but TDMA radios such as in WINSng nodes [29] may have duty cycles of 10–15% for non-base-stations. This analysis illustrates the importance of energy-conserving MAC protocols.

Since we cannot directly measure energy per event, Figure 8 measures bytes sent from diffusion in all nodes in the system normalized to the number of distinct events received. Each point in this graph represents the mean of five 30-minute experiments with 95% confidence intervals. Performance with one source is basically identical with and without suppression (this form of aggregation). As expected, suppression requires less data per event with multiple sources than experiments without suppression. With suppression the amount of traffic is roughly constant regardless of the number of sources. This application-specific data aggregation shows the benefit of in-network processing. It also shows that diffusion is useful for point-to-multipoint communication, since traffic represents both data and control traffic. Comparing traffic with and without suppression shows that suppression is able to reduce traffic by up to 42% for four sources. The network exhibits very high loss rates at that level of traffic. Our current MAC is quite unsophisticated, performing only simple carrier detection and lacking RTS/CTS or ARQ. Since all messages are broken into several 27-byte fragments, loss of a single fragment results in loss of the whole message, and hidden terminals are endemic to our multi-hop topology, this MAC performs particularly poorly at high load. We are currently working on a better MAC protocol.

We can confirm these results with a simple traffic model. We approximate all messages as 127B long and add together interest messages (sent every 60s and flooded from each node), reinforcement messages (sent on the reinforced path between the sink and each source), simple data messages (9 out of every 10 data messages, sent only on the reinforced path, and either aggregated or not), and exploratory data messages (1 out of every 10 data messages, sent from each source and flooded in turn from each node,

[2] An operational sensor network would use timestamps instead of sequence numbers. Both require synchronization, but time can be synchronized globally with GPS or NTP. We use sequence numbers because at the time of this experiment we had not synchronized our clocks. Experimentally, other than synchronization overhead, sequence numbers and timestamps are equivalent.

again possibly aggregated). If data messages are not aggregated, each source incurs the cost of the full path, while if data messages are aggregated after the first hop each incurs one hop cost to the aggregation point and then one message will travel on to the sink. Summing the message cost and normalizing per event we expect aggregation to provide a flat 990B/event independent of the number of sources, and we expect bytes sent per event to increase from 990 to 3289B/event without aggregation as the number of sources rise from 1 to 4.

The shape of this prediction matches our experimental results, but in absolute terms it underpredicts the B/event of aggregation and overpredicts the 4-source/no-aggregation case. We believe these differences are due to MAC-layer collisions in the experiment that tend to drive bytes-per-event to the middle. Only 55–80% of events generated in the experiment were delivered to the sink, so bytes-per-event in less congested portions of the experiment (with one source or aggregation) is high because traffic is normalized over fewer events. On the other hand, with four sources and no aggregation, we believe collisions happen very near the data sources and so the aggregate amount of data sent is lower that predicted. In addition, we sometimes observe longer paths in experiment than we expected.

These experimental measurements of aggregation are also useful to validate our previous simulation experiments that consider a wider range of scenarios. Previous simulation studies have shown that aggregation can reduce energy consumption by a factor of 3–5× in a large network (50–250 nodes) with five active sources and five sinks (Figure 6b from [23]). Although care must be used in comparing energy to bytes sent, a 3–5-fold energy savings with five sources is much greater than the 42% (or 1.7-fold) traffic savings we observe with four sources. The primary reason for this difference is differences in ratio of exploratory to data messages in these systems. Exploratory messages (called low-data rate messages in [23]) are used to select good gradients and so are flooded to all nodes. Data messages (called high-rate messages in [23]) are sent only on reinforced gradients forming a path between the sources and sinks. In simulation the ratio of exploratory to data messages sent from a source was about 1:100 (exploratory messages were sent every 50s, data every 0.5s, messages were modeled as 64B packets). In our testbed this ratio was about 1:10 (exploratory messages every 60s, data every 6s, with messages of roughly the same size). Increasing this ratio in experiment was not possible given our small radio bandwidth (13kb/s rather than 1.6Mb/s in simulation) while keeping reasonable experimental running times. This large difference in ratios is consistent with the large difference in energy or traffic savings.

A potential disadvantage of data aggregation is increased latency. The effect of aggregation on latency is strongly dependent on the specific, application-determined aggregation algorithm. The algorithm used in these experiments does not affect latency at all, since we forward unique events immediately upon reception and then suppress any additional duplicates (incurring only the additional negligible cost of searching for duplicates). Other aggregation algorithms, such as those that delay transmitting a sensor reading with the hope of aggregating readings from other sensors, can add some latency. Understanding aggregation and sensor fusion algorithms is an important area of future work.

Although we have quantified the benefits of in-network aggregation in a specific application, aggregation is one example of in-network processing. Other examples range from simple data caching to collaborative signal processing. As our experiments show, not only do attribute matching and filters make aggregation and similar services easy to provide, they also enable noticeable performance improvements.

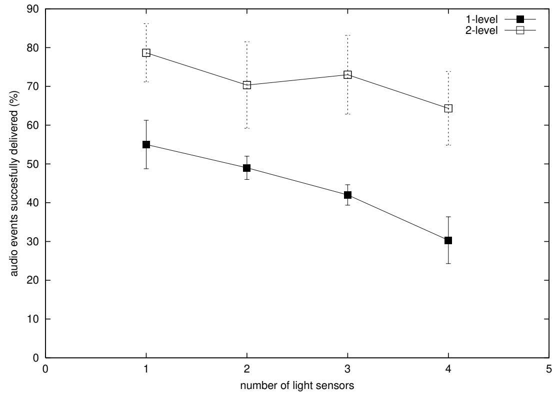


Figure 9: Percentage of audio events successfully delivered to the user.

6.2 Nested query benefits

In Section 5.2 we suggested that nested queries could reduce network costs and latency, and we argued that nested queries could be implemented using attributes and filters. To validate our claim about the potential performance benefits of this implementation we measure the performance of an application that uses nested queries against one that does not.

The application is similar to that described in Section 5.2 and Figure 6: a user requests acoustic data correlated with (triggered by) light sensors. We reuse our PC/104 testbed shown in Figure 7 placing the user "U" at node 39, the audio sensor "A" at node 20, and light sensors "L" at nodes 16, 25, 22, and 13. It is one hop from the light sensors to the audio sensor, and two hops from there to the user node. To provide a reproducible experiment we simulate light data to change automatically every minute on the minute. Light sensors report their state every 2s (no special attempt is made to synchronize or unsynchronize sensors). Audio sensors generate simulated audio data each time any light sensor changes state. Light and audio data messages are about 100 bytes long.

Figure 9 shows the percentage of light change events that successfully result in audio data delivered to the user. (Data points represent the mean of three 20-minute experiments and show 95% confidence intervals.) The total number of possible events are the number of times all light sources change state and a successful event is audio data delivered to the user. These delivery rates do not reflect per-hop message delivery rates (which are much higher), but rather the cumulative effect of sending best-effort data across three or five hops for nested or flat queries, respectively.

This system is very congested, and as described above (Section 6.1), our primitive MAC protocol exaggerates the impact of congestion. Missing events translate into increased detection latency. Although a sensor network could afford to miss a few events (since they would be retransmitted in the next time the sensor is measured), these loss rates are unacceptably high for an opera-

Set A: interest	**Set B: data**
class IS interest	class IS data
task EQ “detectAnimal”	task IS “detectAnimal”
confidence GT 50	confidence IS 90
latitude GE 10.0	latitude IS 20.0
latitude LE 100.0	longitude IS 80.0
longitude GE 5.0	target IS “4-leg”
longitude LE 95.0	
target IS “4-leg”	

Figure 10: Attributes used for matching experiments.

tional system.

However, this experiment sharply contrasts the bandwidth requirements of nested and flat queries. Even with one sensor the flat query shows significantly greater loss than the nested query because both light and audio data must travel to the user. Both flat and nested queries suffer greater loss when more sensors are present, but the one-level query falls off further. Comparing the delivery rates of nested queries with one-level queries shows that localizing the data to the sensors is very important to parsimonious use of bandwidth. In an uncongested network we expect that nested queries would allow operation with a lower level of data traffic than one-level queries and so would allow a lower radio duty cycle and a longer network lifetime.

6.3 Run-time costs of matching

Attribute matching is used in all communication between sensors, filters, and applications in our system. Although technology trends suggest rapid improvement in processor performance, price, and size, sensor nodes may chose to hold performance constant and leverage technology through reduced price and size, so run-time performance must be considered. A second constraint is memory storage, particularly in very small implementations.

To evaluate matching performance we examined the cost of matching data from a sensor. The basic matching in that case compares an 8-element interest against a 6-element data (attributes are shown in Figure 10). To evaluate the cost of larger data objects we increased the number of attributes in the data from 6 to 30 attributes. This experiment was done on our PC/104 sensor node with a 66MHz AMD 486-class CPU. To evaluate the cost of a single match we measured cost of many matches (5000 for matching or 10,000 for the non-matching case) in a loop and normalized, repeating this experiment 1000 times to avoid undue system effects such as interrupts. The order of attributes in each set is randomized each experiment. We also show 95% confidence intervals, although they are always less than 5% of the mean. Although memory caching will cause this approach to underestimate the cost of a match, the basic trends it identifies should be applicable to operational systems.

Our expectation is that the cost of matching is linear with the number of elements. This is confirmed in Figure 11 that shows the cost of matching as the number of attributes in one attribute set increases in different ways. The two lowest lines (no-match/IS and no-match/EQ) show the case where one of the attributes in set A is not matched by those in set B (specifically, the confidence value in set B is changed from 90 to 10). Because the two-way matching algorithm tests the formals in set A first, the incremental cost of additional attributes in set B is fairly small in this case, and it is insensitive to the type of attribute added. If the failing formal was in set B we would expect the cost to be higher (mid-

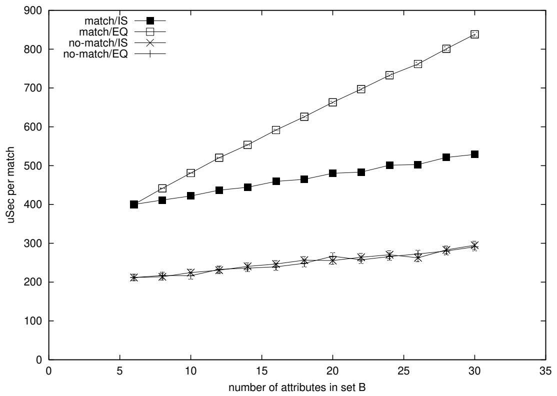


Figure 11: Matching performance as the number of attributes grow.

way between the measured data).

The two higher lines (match/IS and match/EQ) show the cost of matching when all attributes succeed. The difference in cost of additional attributes in these lines shows the cost of additional matching. In the match/EQ line all additional attributes are formals (additions of the “class EQ interest” attribute), so each new attribute must be matched against set A. For match/IS, additional attributes are actuals (repetitions of ‘extra IS “foo” ’) that must be examined but do not require searching.

Although our current implementation is completely unoptimized, the absolute performance of these operations is quite reasonable. At 500μs/match for small attribute sets our quite slow PC/104 can match 2000 sets per second. Although quite slow by Internet router standards, this is reasonable for sensor networks where we expect high-level events to happen with frequencies of 10Hz or less.

Finally, these measurements suggest several potential optimizations to matching performance. Segregating actuals from formals can reduce search time (since formals cannot match other formals there is no need to compare them). Attributes could be statically or dynamically optimized to move the attributes least likely to match to the front. We plan to explore these kinds of optimizations in the future.

6.4 Experiment Discussion

These experiments have provided new insight into sensor network operation, building substantially on our prior simulation studies [23].

These experiments are first examination of nested queries and matching performance. They suggest that the CPU overhead of matching should not be a constraint for reasonably powerful sensor nodes and that nested queries can greatly reduce contention by localizing data movement.

These experiments have explored low-bandwidth operation. Previous simulation studies of sensor networks often have not used the low-bandwidth radios we see in actual sensor-network hardware. Protocols and scenarios behave qualitatively different at 10–20kb/s for sensor networks rather than the 3–12Mb/s common to wireless 802.11 LANs. Even with our early operational experience in small-scale demonstrations and testing, we did not appreciate the difficulty of operating a 14-node sensor network at a relatively high utilization. Our observations suggest two areas of future work: first, sensor networks must adapt to local node densities (we are beginning to explore this area [11]). Second, more work is needed to understand how diffusion's parameters map to different needs, particularly the trade-offs between overhead and reliability present in the frequency of exploratory messages, interests, and reinforcements. Finally, the diffusion applications we currently use operate in an open loop; feedback and congestion control are needed.

Two aspects of radio propagation proved unexpectedly difficult. First, some experiments seemed to show asymmetric links (communication was fine in one direction but poor or impossible in the other). Diffusion does not currently work well with asymmetric links; we are considering how to best revise it. Second, some links provided only intermittent connectivity. A future direction for diffusion might send similar data over multiple paths to gain robustness when faced with low-quality links. Current simulation models, even with statistical noise, do not adequately reflect these observed propagation characteristics.

Finally, we were generally happy with our approach to attribute naming and filters. It was reasonably easy to build and adapt our sample applications and debugging software.

7. FUTURE WORK

This work describes our current approach to constructing robust distributed sensor networks for a few applications. It suggests several areas for future work including enhancing our testbed and protocols, applying them to additional applications, and understanding how to build sensor networks.

We have several planned changes to our testbed hardware. Most importantly, we plan to move to a different radio by RF Monolithics and to use a UCB Mote as the packet controller. The packet-level controller of our Radiometrix RPC was very helpful for rapid development, but this revised approach will give us complete control over the MAC protocol.

We have now explored diffusion performance both in simulation and with testbed experiments. In-network aggregation shows qualitatively the same results in both evaluations (Section 6.1). A next step is to use the experiments to parametrize the simulations.

In this work we were repeatedly challenged by the difficulty in understanding what was going on in a network of dozens of physically distributed nodes. Our current environment augments the radio network with a separate wired network for experimental data collection, but much more work is needed in developing analysis tools for these networks. Tools are needed to report the changing radio topology, observe collision rates and energy consumption, permit more flexible logging, and accurately synchronize node clocks. We have begun work on in-network monitoring tools [40], but more work is needed.

Appropriate MAC protocols for sensor networks is a continuing challenge. In spite of published work in this area [3, 33] and ongoing activities, a freely available, energy aware MAC protocol remains needed. We and others are currently exploring alternatives here; we hope solutions will be forthcoming.

A balance of control and data traffic is particularly important in bandwidth-constrained systems such as sensor networks. Several known techniques to constrain control traffic exist for soft-state protocols in wired networks [24, 31, 36]; these approaches need to be applied to our system.

We have explored two applications of sensor networks and collaborated on other applications, but many other applications remain. One interesting direction is to explore how collaborative signal processing interacts with in-network processing and filters.

Finally, although we focus on wireless sensor networks, the techniques we develop are also relevant to wired sensor networks. Wired connections greatly reduce bandwidth constraints and and eliminate power constraints, but attribute-based naming can reduce system complexity by decoupling data sources and sinks, and in-network processing may reduce latency and improve scalability. Although prior systems have separately used these abstractions for virtual information systems, a future direction is to apply them to large, wired sensor networks that are coupled with the physical world.

8. CONCLUSION

This paper has described an approach to distributed systems built around attribute-named data and in-network processing. By using attributes with external meaning (such as sensor type and geographic location) at the lowest levels of communication, this approach avoids multiple levels of name binding common to other approaches. Attribute-named data in turn enables in-network processing with filters, supporting data aggregation, nested queries and similar techniques that are critical to reduce network traffic and conserve energy. We evaluated the effectiveness of these techniques by quantifying the benefits of in-network processing for data aggregation and nested queries. In one experiment we found that aggregation reduces traffic by up to 42% and nested queries reduces loss rates by 15–30%. Although aggregation has previously been studied in simulation, these experiments are the first evaluation of these techniques in an operational testbed. These approaches are important in the emerging domain of wireless sensor networks where network and power resource constraints are fundamental.

Acknowledgments

We would like to thank Dan Coffin for his work defining the network routing APIs. Van Jacobson first suggested applying directed diffusion to sensor networks.

Many people participated in developing our PC/104 testbed, including Alberto Cerpa, Jeremy Elson, Lewis Girod, Mohammad Rahimi, and Jerry Zhao. In particular, we thank Jerry Zhao for his help on setting up debug stations and PC/104s, and Jeremy Elson for his prompt modification of the RPC device driver. The measurements in this paper would have been impossible without all of their input.

We would also like to thank the TinyOS group (`http://tinyos.millennium.berkeley.edu`) at UC Berkeley for their support with TinyOS and Motes.

The discussion in this paper benefitted from contributions from numerous people, including Philippe Bonnet, Brian Noble (our paper shepherd), and the many SOSP reviewers.

9. REFERENCES

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proceedings of the 17th Symposium on Operating Systems Principles*, pages 186–201, Kiawah Island, SC, USA, Dec. 1999. ACM.

[2] E. Amir, S. McCanne, and R. H. Katz. An active service framework and its application to real-time multimedia transcoding. In *Proceedings of the ACM SIGCOMM Conference*, pages 178–189, Vancouver, Canada, Sept. 1998. ACM.

[3] F. Bennett, D. Clarke, J. B. Evans, A. Hopper, A. Jones, and D. Leask. Piconet: Embedded mobile networking. *IEEE Personal Communications Magazine*, 4(5):8–15, Oct. 1997.

[4] K. P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36–53, Dec. 1993.

[5] P. Bonnet, J. Gehrke, T. Mayr, and P. Seshadri. Query processing in a device database system. Technical Report TR99-1775, Cornell University, Oct. 1999.

[6] M. Bowman, S. K. Debray, and L. L. Peterson. Reasoning about naming systems. *ACM Transactions on Programming Languages and Systems*, 15(5):795–825, Nov. 1993.

[7] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparision of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 85–97, Dallas, Texas, USA, Oct. 1998. ACM.

[8] R. R. Brooks and S. S. Iyengar. Robust distributed computing and sensing algorithm. *IEEE Computer*, 29(6):53–60, June 1996.

[9] N. Carriero and D. Gelernter. The S/Net's Linda kernel. In *Proceedings of the Tenth Symposium on Operating Systems Principles*, pages 110–129. ACM, Dec. 1985.

[10] CCITT. The directory: Overview of concepts, models and service. Recommendation X.500, CCITT, 1988.

[11] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: Application driver for wireless communications technology. In *Proceedings of the ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, San Jose, Costa Rica, Apr. 2001. ACM.

[12] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage retrieval system. In *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, USA, July 2000.

[13] D. Coffin, D. V. Hook, R. Govindan, J. Heidemann, and F. Silva. *Network Routing Application Programmer's Interface (API) and Walk Through*. MIT/LL and USC/ISI, Dec. 2000.

[14] D. A. Coffin, D. J. V. Hook, S. M. McGarry, and S. R. Kolek. Declarative ad-hoc sensor networking. In *Proceedings of the SPIE Integrated Command Environments Conference*, San Diego, California, USA, July 2000. SPIE. (part of SPIE International Symposium on Optical Science and Technology).

[15] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An architecture for a secure service discovery service. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 24–35, Seattle, WA, USA, Aug. 1999. ACM.

[16] J. Elson and D. Estrin. Random, ephemeral transaction identifiers in dynamic sensor networks. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 459–468, Phoenix, Arizona, USA, Apr. 2001. IEEE.

[17] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *ACM/IEEE Transactions on Networking*, 3(4):365–386, Aug. 1995.

[18] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless inter-domain routing (CIDR): an address assignment and aggregation strategy. RFC 1519, Internet Request For Comments, Sept. 1993.

[19] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocols for wireless microsensor networks. In *Proceedings of the Hawaii International Conference on Systems Sciences*, Jan. 2000.

[20] W. R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 174–185, Seattle, WA, USA, Aug. 1999. ACM.

[21] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, Cambridge, MA, USA, Nov. 2000. ACM.

[22] T. Imielinski and S. Goel. DataSpace: Querying and Monitoring Deeply Networked Collections in Physical Space. *IEEE Personal Communications. Special Issue on Smart Spaces and Environments*, 7(5):4–9, October 2000.

[23] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 56–67, Boston, MA, USA, Aug. 2000. ACM.

[24] V. Jacobson. Compressing TCP/IP headers for low-speed serial links. RFC 1144, Internet Request For Comments, Feb. 1990.

[25] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson. Adaptive web caching: Towards a new global caching architecture. In *Proceedings of the 3rd International World Wide Web Conference*, Manchester, England, June 1998.

[26] P. Mockapetris. Domain names—concepts and facilities. RFC 1034, Internet Request For Comments, Nov. 1987.

[27] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The information bus—an architecture for extensible distributed systems. In *Proceedings of the 14th Symposium on Operating Systems Principles*, pages 58–68, Asheville, North Carolina, USC, Dec. 1993. ACM.

[28] L. L. Peterson. A yellow-pages service for a local-area network. *Proceedings of the ACM SIGCOMM Conference '87*, pages 235–242, Aug. 1987.

[29] G. J. Pottie and W. J. Kaiser. Embedding the internet: wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[30] Y. Rekhter, P. Lothberg, R. Hinden, S. Deering, and J. Postel. An IPv6 provider-based unicast address format. RFC 2073, Internet Request For Comments, Jan. 1997.

[31] P. Sharma, D. Estrin, S. Floyd, and V. Jacobson. Scalable timers for soft state protocols. In *Proceedings of the IEEE Infocom*, Kobe, Japan, Apr. 1997. IEEE.

[32] S. Singh and C. Raghavendra. PAMAS: Power aware multi-access protocol with signalling for ad hoc networks. *ACM Computer Communication Review*, 28(3):5–26, July 1998.

[33] K. Sohrabi, J. Gao, V. Ailawadhi, and G. Pottie. A self-organizing sensor network. In *Proceedings of the 37th Allerton Conference on Communication, Control, and Computing*, Monticello, Ill., USA, Sept. 1999.

[34] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, Jan. 1997.

[35] J. Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(10):76–82, Oct. 1999.

[36] L. Wang, A. Terzis, and L. Zhang. A new proposal for RSVP refreshes. In *Proceedings of the IEEE International Conference on Network Protocols*, Toronto, Canada, Oct. 1999. IEEE.

[37] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 70–84, Rome, Italy, July 2001. ACM.

[38] W. Yeong, T. Howes, and S. Kille. Lightweight directory access protocol. RFC 1777, Internet Request For Comments, Mar. 1995.

[39] Y. Yu, D. Estrin, and R. Govindan. Geographical and energy-aware routing for wireless sensor networks: A recursive data dissemination protocol. Work in Progress, Mar. 2001.

[40] Y. Zhao, R. Govindan, and D. Estrin. Residual energy scans for monitoring wireless sensor networks. Technical Report 01-745, May 2001.