

NeTS-NOSS: Creating an Architecture for Wireless Sensor Networks

Wireless Sensor Networks (WSNs) are an important new class of networked system. Simultaneously presenting intellectually deep CISE research challenges and promising tremendous societal impact through scientific progress, better engineering, improved productivity, and enhanced security, research in this area has progressed substantially in the past decade. Early efforts to build the requisite hardware began in earnest around 1999. Before these efforts, researchers used simulation and small laptop/PC104 testbeds to study relatively narrow algorithmic questions, such as redundancy management and topology formation. One notable exception is Directed Diffusion [25], which provided a more general framework for querying sensor network nodes. In 2000, a hardware/software platform [22] became widely available that more accurately captured the salient attributes of this domain - limited resources, significant scale, and deep integration with a noisy physical world. The TinyOS mote enabled pilot deployments and helped researchers focus on some of the most limiting factors for deployments, such as robust, stable multihop routing in the presence of time varying lossy links, holistic power management, time synchronization and in-network query processing. Today, many individual network system components have been researched, designed, developed and evaluated in stand-alone form; in addition, limited collections have been integrated into complete applications. Much more work remains in developing these components.

However, this steady accretion of components, while necessary, is not sufficient for continued progress. The protocols and subsystems that have been developed make a wide range of differing assumptions about the other parts of the system and how the parts should interact. The extent to which these components can be combined to build usable systems is quite limited. This is a very unfortunate state of affairs. In order to produce running systems, various research groups have produced “vertically integrated” designs in which their own set of components are specifically designed to work together, but are unable to interoperate with components from other groups. This greatly reduces the synergy between research efforts, and has impeded progress in the field. It is the central tenet of this proposal that the primary factor currently limiting progress in the field is not any specific technical challenge (though many remain, and deserve much further study) but is instead *the lack of an overall sensor network architecture*. Such an architecture would help identify the essential components and, most importantly, provide a framework in which to compose them. This proposal is intended to help the community move towards an architecture for wireless sensor networks.

It is important to remember that an architecture is not an end in itself: it is merely a tool that helps us accomplish the task at hand. For wireless sensor networks, we define the task at hand to be:

the dense monitoring and analysis of sizable extents of the physical world.

As with the microscope and the telescope, the ability to see the previously unseen often transforms science. That is the promise of sensor networks - a macroscope. However, sensor networks are not providing individual pieces of data that were previously unavailable. Their novelty is that they provide data at a high density, over a large geographic extent, for significant periods of time. Previous monitoring techniques were typically able to achieve one or the other, but sensor networks provide density and scale simultaneously. This capability enables a new understanding of the *systemic* behavior of various phenomena. Settings such as ecosystems and contaminant propagation can only be understood through the monitoring of a large area at a sufficiently fine granularity such that one can determine how the local behavior results in the overall aggregate result. However, such macroscopic monitoring confronts the resource limitations of WSNs; naively extracting all the raw data from the sensors would consume too much bandwidth and power. Thus, in WSNs each sensor node is also a networked computer, allowing substantial data processing to be performed within the network and thereby reducing the bandwidth and power requirements.

Dealing with both scale and density is hard enough in ideal environments. Unfortunately, we don't have the luxury of ideal environments with sensor networks. Because sensor networks are intended to monitor the physical world, they must often be deployed in natural and uncontrolled environments. No longer can we assume the carefully controlled temperature, abundant power, and human monitoring of server rooms and data centers. Instead, wireless sensor networks must be designed to operate while untethered (no external power), unattended (no manual configuration or management), intermittently connected (radios may be turned off for substantial periods of time to conserve power), and uncontrolled environment (operating conditions are variable). Thus, one of our primary design constraints is that the sensor network must adapt (at design time and at run time) to its physical environment, operate within the restrictions that environment imposes, and cope with inherent sources of noise and variation. These issues have been well-articulated in

the literature and we won't belabor them here except to note that these constraints drive many of the design decisions in sensor networks.

While the implications of the various constraints imposed on sensor networks are explicitly addressed in most studies, the issue of heterogeneity is somewhat less explored.¹ One of the primary goals, and accomplishments, of the Internet architecture was to organize the Internet architecture around the "narrow waist" of the Internet Protocol (IP). By requiring all network technologies to support IP, and all applications to run on top of IP, the Internet could accommodate, even encourage, a vast degree of heterogeneity and diversity in both applications and underlying technologies. We have an analogous goal for WSNs; in both the application and device arenas we are in the midst of extremely rapid developments. WSNs will only flourish if we can identify a narrow waist in the architecture that will allow device and protocol developments to proceed apace, while permitting significant optimization.

To this end, we propose the Sensor-net Protocol (SP) to be a best-effort single-hop broadcast. As we discuss later in this proposal, the resource limitations of WSNs result in the SP being closer in nature to the OSI data link layer than the network layer at which IP resides. This requires a careful architecting of the layers above SP to provide a reasonably general platform on which to build various sensor network applications efficiently; that task is the major focus of this proposal.

With these general words behind us, we now discuss more specifically what we mean by the term "architecture."

1 What is a Sensor Network Architecture (SNA)?

The need to build composable components is hardly unique to wireless sensor networks; it is a standard challenge in building large software systems. However, an architecture is more than just the design of reusable software components; it is a set of principles that guide where functionality should be implemented along with a set of interfaces, functional units, protocols, and physical hardware that (roughly) follows those guidelines. For instance, the Internet architecture demonstrated powerfully how a properly chosen set of guiding principles can shape the evolution of a complex system over vast changes in technology, scale, and usage [11]. The philosophy of designing for heterogeneity, change and uncertainty was a radical shift from classical systems design, which more traditionally seeks a near optimal assembly of near optimal parts. Faced with integrating several existing networks with widely varying characteristics, the end-to-end principle and the focus on interoperability led to a design that has successfully coped with unprecedented scale, incorporated a vast array of new underlying technologies, and supported a dizzying variety of applications. However, it was not free of costs; the use of rigid layering sacrificed efficiency in various regards in return for increased interoperability.

The power of this architecture is revealed not so much in the elegance or efficiency of its individual components, but in the overall ability to encompass tremendous growth in scale and in diversity as usage and technology rapidly evolved. This is our goal for developing an architecture for wireless sensor networks. We must be extremely mindful of any loss of efficiency for particular tasks as we seek to greatly enhance the interoperability between components and ability to advance.

To better describe what we propose to undertake, it is helpful to delineate the various aspects of an architecture.

Design Principles: This is the least easily characterized but perhaps the most important part of an architecture. As stated earlier, design principles give guidance about where functionality is implemented, where state is kept, and other essential design decisions. (For example, end-to-end pushed reliability into the hosts, rather than routers.) In so doing, they help constrain the design task to a more tractable set of possibilities.

Functional Architecture: This is the analog of "machine organization" in computer design or system decomposition in software systems. It includes a description of the logical building blocks or functional units, the capabilities of each, and their interconnection. In the TCP/IP stack, this is the suite of protocols and their interdependencies. While these Internet protocols are largely arranged in a layered manner, it appears that the component services in sensor networks are more deeply interrelated, and the current set of designs have wide variations in both the decomposition and the interconnection.

¹Clearly we expect that not all nodes within a sensor network are identical; some may have substantially more capability than others (bigger battery, different radio, more storage, etc.); the various protocols and algorithms should adapt to and take advantage of this heterogeneity. However, here our focus is on the role accommodating heterogeneity plays as a driving rationale of the architecture.

Programming Architecture: This is the classical view of “instruction set architecture” in computer design and API in the networking (e.g., sockets) and software areas. It defines the logical data types that can be expressed, the operations that can be performed upon them, and the semantics of these operations. Of course, this notion of programming interface occurs at multiple levels in complex systems. There is an inherent tension as node constraints, network scale, and reliability drive toward extremely simple networking capabilities in sensor networks, while their application-specific nature and aggregate operation pull for richer APIs.

Protocol Architecture: This is the design of the distributed algorithms used to provide each of the component services and the definition of the information exchanged between instances of these components. These are the mechanisms which collectively realize the network behavior.

System Support Architecture: This refers to the capabilities within a node to support the implementation of network protocols. A node can potentially provide systemic support for functionality such as encapsulation/extraction, multiplexing/demultiplexing, buffer management, event notification, interface management, and communication scheduling. The network architecture is defined independently of an particular system architecture, but it is realized upon those system capabilities.

Physical Architecture: This is the set of nodes, interconnects, and communication fabrics upon which the network is ultimately constructed. Clearly, a network system architecture should encompass a wide variety of current and future technologies, but it is essential to understand what technological limits constrain the overall design from below and requirements make certain technologies inappropriate to the design.

The situation today in sensor networks is that none of these six levels of network system architecture are ‘solved’ or even clearly established. The vast majority of the studies fall in the category of protocol architecture. They deal with particular aspects, such as topology formation, data dissemination, time synchronization, or routing, in an isolated manner without a clear context either above or below (in the suite or architectural components). Other aspects of the system are chosen to suit the needs of the particular protocol component under study. There are very strong reasons why the best architecture for sensor nets unlikely to be a simple modification of an established architecture. Indeed, when we designed TinyOS as a platform for empirical exploration of this space, we elected to build a framework for defining abstractions, rather than a particular set of abstractions, to allow the right abstractions to emerge through the efforts of an entire community. Today there are over 500 research groups using the TinyOS mote platform for experimentation, reasonably high quality stacks of network components have been built, and we are seeing abstractions begin to gel [29]. In some cases they have been general (e.g., local packets with dispatch and tree-based multihop routing), while in others they have been application specific (e.g., power scheduling and time synchronization), but in several important areas the abstractions are completely in flux. In lieu of guiding framework, the most effective progress has been made by taking a vertical slice all the way from particular application to particular platform. These initial slices have been invaluable for identifying key components, their interactions, and, more generally, what it really means to program the various levels of a sensor network. But, with no general design principles guiding the work, no particular decomposition of functionality frames the investigation, and consensus on technological constraints has not been achieved. Unsurprisingly, the commonality across current solutions is limited. We have learned a great deal in letting chaos reign, and it is now time to harness these learning and rein in the chaos in the form of a coherent overall architecture.

There have been previous attempts to sketch architectural frameworks for sensor networks. For instance, there is a rough “layering” of the components in sensor networks, as depicted in [44]. There is also a sketch in [19] of how users interact with various sensor network components. Both of these only attempt to provide a conceptual framework for how to think about sensor networks, and are not intended to provide detailed guidance about how to build them. A description of typical physical structures appears in [7] and a general framework is extracted from habitat monitoring deployments in [33]. The construction of a more complete architecture is the challenge of this proposal.

We now proceed with a more detailed discussion of the various layers of an architecture.

2 Physical Architecture

The physical architecture of sensor networks has become fairly well established, although there is a huge and evolving range of constituent technologies and wide variation in the individual instances. The general structure of real deployments, in both natural and built environments, is illustrated in Figure 1. Large collections of resource constrained sensor nodes are typically deployed in dense patches. These may concentrate on a

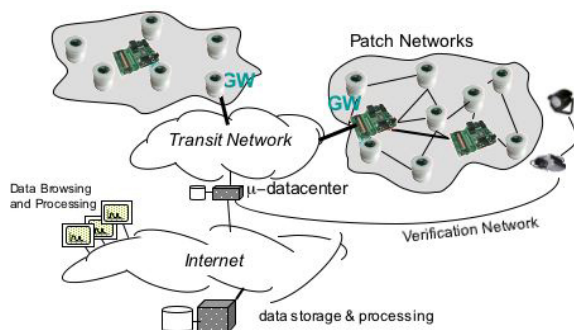


Figure 1: A Sample Physical Architecture of a Deployed Sensor Network

transect of a habitat, a tree in a forest, a set of joints in a structure, a security zone, or key machinery in a plant. Several physically separated patches, comprising the lowest tier of the network, are interconnected with some form of transit network, which may use specialized links or conventional data networking technology. At one or more points, this transit tier is connected to the Internet (or intranet) and thereby to powerful storage, processing, visualization, and access resources. In addition to this essentially stationary structure, there may be mobile nodes of a wide variety of sorts that move through and potentially between the patches.

The patches are generally not homogeneous (except in certain military scenarios demanding very rapid deployment). The vast majority of the nodes will be resource and energy constrained, but not necessarily uniform. They may have different sensing modalities, different roles, or different hardware or software components. Some nodes serve as gateways to the transit net, or directly to the Internet. Some nodes may have additional capabilities, such as an abundant source of power, greater energy storage, larger data storage, faster processor, or special sensors. Higher bandwidth sensors tend to require greater storage and processing capacity. Greater processing, storage, and communication capabilities are of little value without also providing additional power capacity. However, merely providing additional power to some of the otherwise resource constrained nodes provides value to many nodes.

Since Moore’s law has brought rapidly improving processing power, capacity, and bandwidth to other areas of information technology, the question arises as whether the resource constraints of the lowest tier of sensor nets is merely a short term phenomena. The answer is “no”, for several reasons. Generally, the Moore’s law advance assumes power consumption is essentially unlimited. (Indeed, the primary limit to performance is becoming dissipation of the heat produced by the incredibly power-hungry microprocessors.) At fixed power, decreasing feature size does provide more performance, but the rest of the system does not scale as well. Wireless communication requires a certain amount of power, which increases rapidly with distance, in the transceivers. Increased processing may improve the bit rates, but does little for range. Leakage in memories dominates the standby current, and storage needs increase with processing rate. Today, the microcontrollers used in motes have orders of magnitude more processing capacity per unit of storage or communication than PCs for these reasons[21]. Energy density of batteries and fuel cells improves slowly. Clearly, in indoor environments there are many sources of energy, however, wiring costs become dominant. Finally, what improvements are obtained are as likely to be utilized to reduce cost, increase density, and improve lifetime of the lowest tier as to increase the capacity of each node. The greater the ability of the network architecture to scale down to constrained nodes the greater its technological reach. At the same time, where greater resources are available they should be utilized and should reduce the impact of constraints elsewhere.

3 Programming Architecture

This physical architecture suggests that there are two kinds of programming interfaces to sensor networks. One form of programming determines how processing is performed across the numerous nodes that make up the field of intelligent sensors. This requires one or more levels of *internal* programming interface. The other is the *external* programming interface, which deals with how information extracted from the sensor net is processed on conventional networked computing resources. Real sensor network applications involve both, but external interfaces are fairly conventional. Our focus is on the internal programming interfaces. How are

applications within the sensor network constructed and how are each of the levels of capability that those applications utilize constructed?

Substantial guidance can be gained from the pilot applications of recent years. The most common, and simplest, usage pattern is data collection: collecting time series data from essentially all sensors into a small set of sink nodes, typically gateways. This invariably occurs in early stages of a deployment where properties of the data must be understood before algorithms for in-network processing can be selected, but it occurs throughout the life of the application as new in-network algorithms are developed and for basic network administration. Local processing at each node can filter, compress, or distill the data, reducing the overall data collection load. An extreme form of local filtering occurs in alarms, which only communicate detection events. Collection is typically performed as a tree (or forest) rooted at the sink(s). Several such sense-and-send applications have been constructed, where the external programming interface is essentially setting parameters, such as sample rate and filter selection. The internal programming interface issues involve determining which nodes participate in the operation, what processing is performed, formulation of the collection tree and how results are collected[33, 47] and the underlying network evolves due to changing connectivity and node population.

More sophisticated in-network processing operates on data across multiple nodes, rather than locally to a node. The natural extension of collection is to perform aggregation (reduction, histograms, compression) within the collection tree or DAG directed at the sinks. For example, TinyDB[32] and Cougar[54] place a small query processor at each node which can perform a restricted set of SQL-like queries on streams of sensor readings to produce partial results and can collect or aggregate partial results through the collection tree. The external interface is a data structure describing the query, allowing a fixed set of in-network processing algorithms to be used for a fairly wide variety of external applications - much wider than the sense-and-send data collection. The internal programming interface includes the naming of queries and partial results, as well as the intercepts for aggregation within the tree. Generalized, in-network query processing is essential for health monitoring of the network, an addition to physical data. Directed diffusion[25] provides essentially select operators at the nodes, but allows multiple internal sinks which may feed partial results into nested queries to perform aggregation. This requires a more elaborate external interface than network-wide queries.

Observe that all of these programming abstractions rely on a more basic capability to disseminate information throughout all or part of the network. Whether it be configuration parameters, queries, or the machine code that executes on the nodes, dissemination is fundamental. Typically, each dissemination establishes a stream of collection or aggregation operations. The programming interface involves the scope of the dissemination and its semantics in the presence of disconnection, node failures, and joins. Dissemination rooted at internal points is also extremely important. For example, information from particular sensors may establish thresholds for processing at many other nodes.

Several richer forms of in-network processing have been proposed where various parts of the network process information generated in other parts, rather than distill information bound for external use. Two primary forms have evolved: neighborhoods and data-centric storage. Neighborhoods involve processing data over geographically and topologically contiguous portions of the sensor net. For example, in tracking applications the set of nodes in sensor range of the target perform local processing and exchange information to cooperatively identify the position of the target[51]. In environmental monitoring applications, regions bounded by particular isobars may need to be identified. Localization algorithms perform distributed shortest path algorithms on ranging data using a few anchor positions[6]. A common internal interface for these algorithms is a “reflected tuple-space”[51, 50], in which each node maintains a set of attribute-value pairs and mirrors its tuples onto node in a local neighborhood, defined either geographically or topologically.

Data-centric storage is especially important when the information to be combined is not geographically or topologically localized[44]. This occurs, for example, in aggregates over narrow selections. If only a few widely distributed nodes are producing partial results, an aggregation tree built over the entire network is inefficient. Introducing an additional level of indirection provides a way to gather disparate data into known or even consolidated regions so that additional processing can be performed efficient. Typically, the internal interface provides *put*, which stores a value v according to a key k , and *get*, which retrieves the values bound to a key. Data-centric storage is essentially passive. In Section 4.5 we propose an extension where nodes can associate active triggers with the *get*.

The dominant communication abstraction in conventional computer networks, universal point-to-point exchange, is not widely used as an application programming interface in sensor networks. This appears to

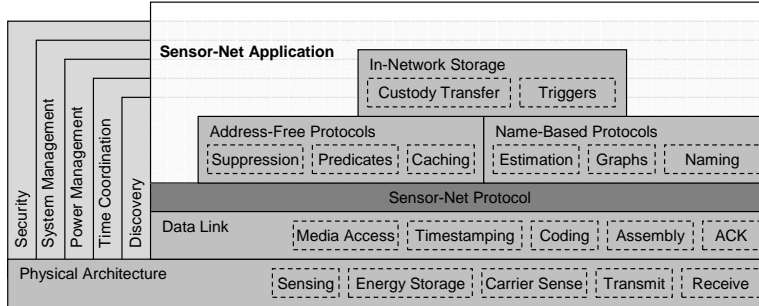


Figure 2: Sensor Network Functional Layer Decomposition

be for several reasons. Implementing topologically localized collection, aggregation, and dissemination using any-to-any routing is quite inefficient. Sensor nodes are essentially servers of physical data. Users are seldom associated with the individual nodes generating streams of requests, as in client-server, web, or file transfer contexts. The nodes are deployed over a physical space for a particular set of purposes and act as a coherent ensemble. The clients are other nodes interested data over regions of space, time, or value ranges, rather than in particular servers per se. However, point-to-point routing is likely to grow in importance, at least at layers below the application programming interface. As applications become more sophisticated, it will not be uncommon for particular nodes to provide specific information to other nodes and network management requires the ability to probe particular nodes. While these situations may be addressed with limited point-to-point routing, general any-to-any routing occurs as mobile nodes interact with one another and in support of data-centric storage abstractions.

The SNA will define consistent application programming interfaces that encompass these seven communication abstractions: collection, aggregation, dissemination, neighborhoods, data-centric storage and attribute-based routing.

4 Functional Architecture

Designed to interconnect multiple existing networks into a larger service, the Internet architecture is based upon a single, shared protocol – the Internet Protocol – at the network layer. Above IP, there are many transport and application protocols, such as TCP, HTTP, and SIP; below it, there are many data link protocols, such as Ethernet, ATM, and Frame Relay. The Internet’s “narrow waist” comes directly from its goal of universal end-to-end connectivity and the dominant usage model of pairwise communication between often separated entities.

The definition of a narrow waist is essential for sensor networks as well, in order to decouple the diversity of higher level protocols and diverse, evolving underlying technology. However, as a result of design characteristics and application requirements – in-network processing, cross-layer control, a wireless medium, a wide range of communication patterns – a best-effort single-hop broadcast emerges as the unifying abstraction. This is consistent with what has been observed in the TinyOS context, where the narrow waist is essentially the active message layer[29]. A wide range of network protocols build on top of this primitive, including flooding, dissemination, collection trees, and virtual coordinate routing, each with its own traffic characteristics and addressing. However, the reason that the waist moves down is fairly fundamental. Applications differ dramatically in their communication patterns and are intimately tied to the associated network protocol. They generally do not require and often do not benefit from a common, universally routable addressing scheme. Instead, a single, simple interface needs to be provided to implement a range of routing protocols, independent of the underlying link layers. Moving the point of narrow, universal abstraction downward present new issues that we do not typically concern ourselves about in the Internet architecture. Here we identify what we see as the key elements of the layers in the emerging sensor net architecture.

As the Internet’s narrow waist is called the Internet Protocol, we refer to a sensor network’s as the Sensor-net Protocol, or SP. Starting with SP, we discuss the design issues and research questions in the sensor network equivalents to the physical and data link layers, then work up to the network layer. At this point, sensor networks diverge enough from their Internet analogues that we use an alternative layer decomposition, shown in Figure 2.

4.1 The Sensor-net Protocol

SP dispatches incoming packets to various higher level services and protocols should have the equivalent of the IP protocol identifier (the AM handler ID in TinyOS). In IP, the destination address controls downward dispatch to an outgoing link. However, in sensor networks many useful communication models are address free; we discuss several in Section 4.2. The SP send arguments must contain sufficient information to determine the out-bound interface(s). Often, devices have single interface, a radio, but some have one or more wired links, or multiple radio channels. One option for interface dispatch is to require explicitly stating the outbound channel, placing the burden for the first routing step on upper layers. Alternatively, the default channel can be a given interface, and SP can allow higher layers to select other interfaces through optional addressing.

Being a local communication abstraction, SP is much closer to the OSI data link layer than the network layer. However, as it is a unifying abstraction with a common format and semantics across many physical layers, how functionality divides across the packet boundary is a key question. We believe that the interface needs to be more expressive than current data link interfaces (such as Ethernet) for higher level control, yet have decomposable functionality below for greater flexibility.

Underlying mechanisms over which a caller of the send operation should be able to exert control include generation of link level acknowledgments, performance of initial and collision-avoidance backoffs, post-media arbitration timestamping, degree of FEC, retransmission and power management. These mechanisms need to be optimized for each particular link technology, rather than performed in a general fashion above the SP interface. Additionally, they should not be encapsulated in an inflexible and predefined manner. As sensor networks have a small number of widely distributed applications, a greater ability to allow customization and application interaction can lead to improvements without the traditional OS issues hard protection boundaries through virtualization.

For example, in addition to the traditional question of whether fragmentation occurs below or above the basic communication interface, when designing SP we must consider where protocol exchanges to reduce hidden terminals and improve fairness, such as RTS/CTS, belong. Exposing control of the underlying mechanisms allows middle layers to provide the additional functionality when it is needed at performance that is competitive (or even exceeding) the general solution [40]. Simple defaults allow easy composition and use in basic operation.

In addition to the address-free broadcast, SP also has a directed send operation. Returning to the issue of dispatch to physical layers, the exact form this addressing scheme takes directed communication is an open research question. In the opposite direction of data flow, reception should provide an upward path for additional metadata and physical information, such as time of arrival, signal strength, and source. Additionally, reception demultiplexes packets across multiple network services and protocols.

SNA will define SP with the view that it is the narrow universal abstraction of the overall architecture. It will be implemented over multiple link layers and utilized by a range of network layer protocols.

Having presented the design issues and constraints of the data-link and physical layers, we start moving up the network stack, starting with the simplest slice of the network layer, address-free protocols.

4.2 Address-Free Protocols

Because applications are distributed over a physical space of interest and the network overlays that space, many data processing abstractions can be naturally expressed without referencing the nodes involved by addresses. One basic example is “neighbor communication” to all (or a subset) of the nodes in direct physical connectivity, which approximately corresponds to physical proximity. By comparing neighbor communications, nodes can establish bidirectional neighbor relations. Although names may be used to distinguish potential neighbors, these names are not needed for the communication itself. Whole-network data dissemination can also be an address-free protocol. [30] However, once a protocol introduces retransmission, it must also handle the issues of suppression and redundancy detection, or it will face problems such as the broadcast storm. [36]

Fundamentally address-free protocols provide communication over a connected subset of nodes defined by a *retransmission predicate*. The two examples represent the edge cases – the neighbor communication predicate always resolves to false except the source, while the dissemination predicate always resolves true – but there is a wealth of intermediate points, based on characteristics such as hop count or sensor values. Using the union and intersection of several more expressive predicates, an application can refine the retransmission

scoping, and routing (discussed later) can distribute across disconnected regions.

As address-free protocols depend on a small subset of the SNA, but provide useful application-level services, an immediate question is how nodes express retransmission predicates. Static predicates allow design-time analysis and behavior characterization, while dynamic predicates provide much richer capabilities and can allow more efficient operation based on run-time information. Additionally, there is a question on how these can protocols specify caching and data lifetime. Theoretically, as long as one node maintains a copy of the original data, new nodes satisfying the predicate or joining the connected region can participate. Resource constraints may prevent every node from caching a copy, requiring consistency and versioning protocols. However, there is distinction a between sufficiency and efficiency; the more nodes that store the data, the fewer hops it will be to a source. Alternatively, the propagated data may be transient, as in the case of simple floods.

Numerous higher level communication models proposed by the research community essentially boil down to defining the data structure and rules for maintaining the predicates. For example, directed diffusion [25], attribute-based routing, and various tuple-space abstractions [51] utilize a list of (attribute,value) pairs to define the predicate algebra. Exploration and reinforcement involve adjusting predicate selectivity.

In developing SNA, we will build a collection address-free protocols over SP, focusing on general, yet efficient techniques for defining the forwarding predicate and reusable mechanisms for duplicate detection, suppression, and transmission scheduling.

4.3 Name-based Protocols: Topology Management and Routing

A second, important class of network-layer services provides multi-hop communication based on directly addressable nodes. An appropriate choice of routing protocol and associated addressing structure can enable more efficient communication. Protocols belonging to this class include data collection, aggregation, and point-to-point routing. Key architectural issues that arise in designing these protocols include (1) what naming scheme to use, (2) how does a node performs packet forwarding and processing relative to the naming scheme (data path), and (3) how does a node discovers and maintains routing and processing state (control path).

At their basis all these protocols are implemented by having nodes differentially processing packets based on one or more *names*² or identifiers in the packet headers. A name can have either local or global scope, and can be used to identify an individual node, a set of nodes, or a communication structures such as a tree. From the point of view of a node, a name represents a handler that can be used to process an incoming packet. Upon receiving a packet, a node extracts the name from the packet header and then, if it maintains state associated to that name, it process the packet accordingly. If it doesn't know anything about that name, the node will perform a default operation such broadcast the packet to all its neighbors or dropping the packet. An IP address in the Internet is an example of name. Indeed, upon a packet arrival, an IP router uses the destination address in the IP header to perform a lookup in its routing table, and then based on the result forwards the packet to the next hop. In the reminder of this section, we discuss several examples of names in the context of sensornets.

A fundamental requirement for building these name-based protocols is the formation and maintenance of a connectivity graph, represented by routing tables in each of the nodes. At any point in time, each node can receive transmissions from a set of neighboring nodes with a non-zero packet success probability. Individual nodes perform link estimation to judge connection quality, using passive or active techniques, and build a table of well-defined links that act as an approximation to the underlying physical communication graph. As wireless links are often asymmetric [18, 56], a good incoming link from a node does not necessarily mean there is a good outgoing link to that node. Therefore, to form a bidirectionally-consistent mesh for the whole network, connectivity information must be exchanged between neighboring nodes. This connectivity exchange requires each node to be able to uniquely identify (name) its neighbors, in order to accurately associate estimates with specific neighbors. Note that in this case a name need only be unique within the neighborhood of a node, it does not need to be unique in the entire network.

Current sensor networks used for monitoring typically maintain a modest number of simultaneous collection trees. Trees are named by the unique MAC-style address of the root node, by the query id (in TinyDB), or by a hash of the attribute set describing the data to be collected (in Directed Diffusion). Each node

²By abuse of notation in this section we use the term “name” to refer to “address” as well; indeed, at the limit an address can be viewed as a name that encodes location semantics.

maintains a parent, or small collection of candidate parents, in each tree. Routing is performed along tree edges by selective forwarding at each hop. Most often, the transmitter specifies the neighbor responsible for forwarding, but different routing structures can enable receivers to make this determination. Multipath routing and multiple roots naturally extend these trees into DAGs, while maintaining similar naming and estimation techniques.

Although not the predominant traffic pattern, as it is in convention computer networks, any-to-any routing has important uses, such as communication between mobile nodes moving within a sensor field. Many naming schemes and associated routing techniques have been proposed. Classical AODV approaches build a tree rooted at one of the endpoints and prune branches that do not reach the other endpoint [38]. Mobility and changing connectivity require continuous tree reconstruction and pruning. Alternatively, mobility can be supported by building a tree from an arbitrary landmark and then reinforcing paths from the endpoints to the root. When connectivity is relatively stable, the relationship between the landmark and each node, as encoded in the tree structure, can be used to generate routable names for each of the constituent nodes (cf, GEM [35]).

Given that many sensor networks are distributed over space, spatial coordinates are often an attractive naming scheme. With an established connectivity graph, spatial naming can enable greedy routing. At each hop, the neighbor closest to the ultimate destination is selected, with GPRS-style heuristics used to cope with obstructions. However, this requires that all nodes have established geographic coordinates, which is often not the case. Recently, a family of pseudo-geographic routing schemes have been developed that produce a virtual coordinate system based solely on the connectivity graph [41]. We have developed a technique where the virtual position and name of each node is simply a vector of its hop-count distance from each of a small set of landmarks. These naming and routing techniques are powerful enough to support content-based storage within the sensor network, which requires arbitrary point-to-point routing with low stretch [42].

Unlike Internet routing, sensor network routing does not generally follow the end-to-end principle. In addition to packet forwarding, a node along a path can inspect received data and make local decisions with it, possibly transforming the data before forwarding it, or suppressing it. This in-network processing can greatly reduce communication while keeping higher-level semantic requirements. For example, when collecting a MAX query over a network (which returns the maximum sensor reading), nodes need only forward the highest reading they receive, and if a node hears a neighbor transmit a higher reading than its own, it can suppress its subsequent transmission.

Besides the actions taken on the data path, a name-based protocol needs also to specify the control plane, that is, how do nodes maintain the routing and processing state. With nodes distributed densely over space, irregular wireless connectivity, and limited storage resources, a node will typically receive packets from many more nodes than it is able to record as possible neighbors. Effective algorithms exist for maintaining useful subsets, but these should be guided by the utility of potential links to the network-level services. For example, many tree-based collection and aggregation protocols use underlying estimators to form a spanning tree over region of the network, rooted at a particular node, and choose links based on minimizing expected costs along each leaf-to-root path. Node heterogeneity can be encapsulated by a cost metric that includes expected transmission count or energy consumption.

For example, in a standard tree-structured collection routing protocol each node chooses one link to a parent, based on a cost metric such as hop distance. This builds a spanning tree over the mesh. Other routing algorithms, such as Diffusion's sink-source pairing [24], or TinyDB's result-splitting [31] layer other topologies, such as a directed acyclic graph, over the mesh.

Our goal in developing SNA is to define a simple set of primitives at the SP layer that can be shared by these various network-layer services. In addition, defining a set of network-layer services allows nodes from different applications with shared administrative domains to provide each other with routing. As each node is effectively a router, it would be natural for nodes from one network to participate in application-independent portions of the neighbor discovery and forwarding associated with a partner network.

4.4 In-Network Storage

The role of storage is a critical aspect of the sensor network architecture. Whereas the Internet architecture was greatly simplified by maintaining an end-to-end model without explicit internal storage, in-network processing, high loss rates, and intermittent connectivity strongly pull toward incorporating storage into the network architecture. The core question is how to do this while preserving the ability to operate while nodes

come and go and to restart the network from scratch.

The storage requirements are closely associated with the particular network protocols. Data collection can be realized, and is currently, for low data rates with essentially no internal storage. Connectivity and routing information is soft state and simple buffering occurs at each hop providing a best effort (with possible link-level retransmission) collection path. However, to support high sample rate applications and to maintain data quality in the presence of disconnection, rather than link loss, more persistent forms of storage, such as custody transfers, are needed [15]. Aggregation also requires greater lifetime of the in-network storage in order to provide time windows over which children may feed data into the reduction operator. Dissemination protocols require local storage of the disseminated data in order to update neighboring nodes that have been disconnected for any length of time. Hierarchical caching may be necessary for large pieces of data. Various kinds of caches and attributes stores have been used to date in a wide variety of ways. *A critical element of the architecture is to provide a soft-state storage abstraction as a building block for a variety of the network protocols, both address-free and name-based. In particular, it will be necessary to identify where storage within the network layer should be used to provide greater reliability, persistence, and support for bulk transfer, and where it is better addressed as a transport layer above.*

Incorporating richer forms of storage into the architecture can support more complex usage patterns and enable transport-level protocols that provide varying degrees of reliable transfer [17, 42]. In particular, in-network storage and query processing capabilities are deeply interrelated. Requesting data directly from nodes can be more efficient if the user knows what data is needed as it is generated. Persistent local storage can allow historical queries based on observed phenomena. By decoupling the locations of data storage and production, an architecture can support further decentralization of query processing and better resilience to node failure. This is useful, for example, to query the history of a node which failed, to try to determine the cause of failure. Decentralization can use characteristics of the data itself to determine the most effective storage location. With data stored at well-defined locations, queries for data that may require information collection from multiple sources can instead be routed to a unique rendezvous point. If data sources have previously transmitted information to the rendezvous point, querying can be inexpensive, as it requires only contacting a few specific nodes, rather than the entire network. While there is much current work on supporting a hash-table-like put/get interface, it is too early to know if they will work in typical sensor network systems.

We don't take a position on the viability of these approaches here, except to note that if indeed these kinds of sophisticated in-network data structures become practical and are of broad utility, then they should be accommodated within the overall architecture. One of the key questions is how one builds these data structures without detailed knowledge of the underlying routing algorithm. For instance, Geographic Hash Tables [42] provide the necessary put/get interface but rely on geographic (or pseudo-geographic) routing. Can one build the data structures on other routing algorithms?

Our goal is to study the trade-offs between the reliability and consistency semantics associated to the stored data, and the complexity and the overhead of the algorithms to implement these semantics.

4.5 Active In-Network Storage

The higher-level use of in-network storage proposed so far are largely passive; data is stored at a rendezvous point, waiting to be accessed. One could imagine making a more active use of in-network storage in which when data reached a rendezvous point, it triggered some action. This is similar in spirit to the approach taken in the Internet Indirection Infrastructure (*i3*) [46].

At its basis, *i3* offers a rendezvous-based communication abstraction. Instead of explicitly sending a packet to a destination, each packet is associated with an identifier; this identifier is then used by the receiver to obtain delivery of the packet. One way to generalize this approach is to associate an "action" with each trigger. When a packet matches a trigger the action associated to the trigger is invoked on the packet. By specifying proper actions, higher level layers or applications can use triggers to implement a multitude of communication abstractions such as aggregation, dissemination, or continuous query processing. For example, a node can use action-based triggers to send a packet to all nodes that measure a temperature less than say 60 degrees, or to gather the light intensity from all nodes that measure a temperature greater than 30 degrees. Our approach is complementary to the Direct Diffusion approach, which can provide similar functionality. More precisely, while the trigger-based approach is more efficient in the cases in which only a subset of nodes exchange information, Direct Diffusion is more efficient in the cases in which a large fraction of the nodes in

the network communicate.

One natural question we plan to address in our research is what is the trade-off between the generality of the actions associated to triggers, and the expressiveness of the abstractions that can be implemented. While this is an open question, next we give a particular example in which we show that even very simple actions can implement powerful functionalities such as predicate evaluation. Consider the example in which a node, $node_id$, wants to receive packets (1) from each node that either measures a light intensity higher than 1000 candles *and* a temperature between 60 and 80 degrees Fahrenheit, *or* (2) from each node that measures a temperature lower than 30 degrees. This predicate can be written as $((id_t : [60, 80]) \wedge (id_light : (1000, \infty))) \vee (id_t : (-\infty, 30))$, where id_t and id_light denote the temperature attribute and the light attribute, respectively. To evaluate this predicate, the node can insert the following triggers (the meaning of the trigger fields will be clear below):

$$\begin{aligned}
 t_1 &= ((id_t : [60, 80]), id_1, "xor"); \\
 t_2 &= ((id_light \otimes id_1 : (1000, \infty)), node_id, "replace"); \\
 t_3 &= ((id_t : (-\infty, 30)), node_id, "replace");
 \end{aligned}
 \tag{1}$$

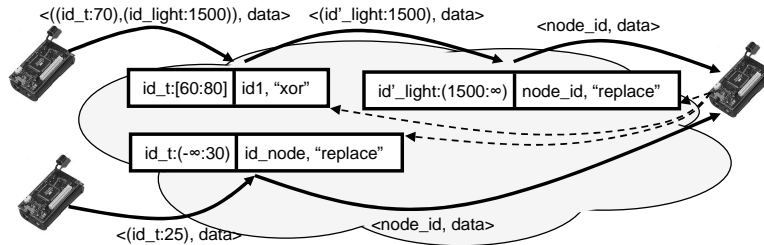


Figure 3: Example of a trigger-based implementation of a predicate.

Each node periodically sends the values measured locally as a list of attribute-value pairs. Figure 3 shows how a packet carrying a list of attribute-value pairs $((id_t : 70), (id_light : 1500))$ and a packet carrying a list $(id_t : 25)$ are both delivered to node, $node_id$, as they both satisfy the predicate inserted by the node. A packet matches a trigger when the attribute at the head of the packet's list is the same as the trigger's attribute, and the value of the packet's attribute falls in the range of the trigger's attribute. The last two fields in the trigger specify how the attribute-range list in the packet header is updated. Packet, $((id_t : 70), (id_light : 1500))$, first matches trigger t_1 . As a result, the attribute-value pair $(id_t : (60, \infty))$ at the head of the list is removed, and the identifier of the second attribute id_light is updated, according to the action "xor", to $id_light = id_light \otimes id_1$. Next, the packet, now containing the list $(id_light : 1500)$, matches trigger t_2 . In turn, the attribute-range at the head of the list is replaced with attribute $node_id$ (this is specified by action "replace"). Finally, the packet is forwarded to the node identified by attribute $node_id$.

Our goal is to identify and explore the set of minimalist actions that are flexible enough to enable applications or higher level layers to implement expressive predicates and queries, and to study the feasibility of such an approach in a large scale sensor net.

5 Cross-Layer Issues

Several critical architectural issues cut across layers or arise in distinct forms within multiple layers. We have seen that cross-layer control, as well as cross-layer optimization, are far more important in the Sensor Net regime than in traditional networking. As part of defining the new architecture, we will need to establish clear guiding principles and specific interfaces and mechanisms to address these cross layer issue. Here we briefly discuss five critical setting in which these issues arise.

5.1 Discovery

Discovery is fundamental requirement for network self-organization. Deployments cannot require a full-time administrator to configure the network. However, remote reconfiguration, management, and inspection must be possible. Although a deployment site may be studied in advance, actual network structure and connectivity information can only be determined by deployment, and may change due to environmental conditions. Once

deployed, an application may evolve, nodes may die or be added, and nodes with different resource tradeoffs may be introduced; discovery should consider these changes. New deployments of different applications may introduce additional resources, such as routing fabrics or collection points, creating the possibility that logically distinct sensor networks may interact or support one another.

Establishing a connectivity mesh, discussed in Section 4.3, is one example of discovery. Depending on the link technology, there are various mechanisms for detecting devices and estimating link quality. For example, software link estimators use packet-level passive listening or active probing [52], Bluetooth has a complex discovery process that filters out nodes below a threshold [5], and 802.15.4 provides simple discovery through hardware link estimation based on observed bit error rates [23]. The SP interface must present an abstraction of connectivity discovery so routing protocols can establish a mesh which filters links according to higher-level criteria [52, 32, 16, 26]. A connectivity abstraction requires that these connections have names. One naming approach is to assume all nodes have a unique identifier, similar to an Ethernet address. However, it is also possible to negotiate locally unique names or short nicknames [12].

Once nodes can be discovered, the next step is discovering their key characteristics, such as energy, storage, processing, and sensors. For example, many energy-aware routing protocols seek to route traffic through powered nodes or try to avoid depleted nodes [8, 53, 9, 55, 1]. Certain nodes may be deployed with special links, e.g., gateway nodes bridging a patch low-power link to a transit long-range link [33, 2, 14] or nodes may be dedicated to performing a backbone [1, 43] role. Reliable transport protocols need to identify storage resources that can support custody transfers [15, 37]. Protocols at different layers have particular characteristics to which they are sensitive: there need to be interfaces and mechanisms for protocol peers to exchange relevant characteristics. One important question is how to efficiently share and coordinate the various on-going discovery processes from many layers. We see this emerging as a cross layer service that serves to consolidate and multiplex discovery inquiries.

In addition to detecting when and where resources are, the discovery service must also be able to quickly detect their disappearance. For example, if connectivity to a collection tree root breaks temporarily, nodes may wish to locally store data before forwarding it. Simple approaches, such as periodically contacting the resources, scale badly, as many nodes may be continually generating redundant traffic. A tension exists between responsiveness, accuracy and the local state storage requirements.

In addition to discovery within the protocols, there is a need to discover the protocols themselves. As nodes are not homogeneous and may be deployed incrementally, not all nodes will support all services. For example, a reliable dissemination protocol may need to determine which of its neighbors, according to link connectivity, are able to participate. This requires a protocol naming scheme, a registry, and a management protocol. It is another (attribute,value) list that can be extended for higher level discovery, such as applications and cooperating services.

One test of the SNA will be the design of a discovery service that consolidates information shared between peer services across the SP links, eliminating redundancy.

5.2 Time Coordination

Time coordination represents a rather different kind of cross layer challenge, partly because there are several very different kinds of temporal relationships to establish and partly because of strong dependences between protocol implementations and time. Time synchronization in sensor nets is different than in the Internet as the entire mesh, rather than solely the end points, can participate.

At a low-level, very good techniques exist for obtaining temporal relationships between nodes. Post-CSMA timestamping, introduced in the TinyOS MAC, provides microsecond granularity references between transmitter and receiver [20]. RBS provides tight receiver-receiver synchronization [13]. Combining these with an NTP-like round-trip protocol can produce excellent time references among complete node neighborhoods. Open issues include how offset and skew relationships propagate over multiple hops and time adjusts locally.

Many network protocols have time synchronization needs. Having synchronized clocks can allow nodes to schedule with neighbors along a routing path, allowing the protocol to power down the radio when not actively communicating. Additionally, scheduled communication can proactively lessen contention by more evenly allocating bandwidth over time and space. Numerous fine-grained TDMA schemes have proposed as well. The architectural challenge is how network protocols, time synchronization protocols, and communication power-scheduling interact with one another to allow meaningful, non-conflicting composition.

At the application level, time coordination is primarily useful in correlating sensor data obtained across

different nodes. At low sample rates, such as in microclimate monitoring, typically the goal is to collect simultaneous readings across the network within a tolerance of tens of milliseconds to a few seconds. In high sample rate applications, such as structural monitoring, the goal is to instead start sampling with a precise (e.g., microseconds), known temporal offset and sample at a common fixed rate. Coarse relationships are efficiently established relative to a common or global reference. However, continuous correction to a reference introduces significant quantization error for very fine grained timing. Thus, we expect that time services will need to deal with both notions of virtual reference time and intermittently corrected local time. Furthermore, applications often have clear times when they can – and cannot – tolerate adjustments [29].

SNA will provide a configurable time service, probably with pieces at multiple layers in the architecture, providing interfaces to the other protocol components.

5.3 Power Management

Power management is fundamental in sensor networks and particularly challenging to abstract into a clean architectural concept. Traditionally, power aware networking has been dealt with at a single point in the stack in isolation. Power aware MACs attempt to turn off the radio invisibly to the stack above, for example after receiving the header of a packet not intended for the node in question [45]. Buffering transmissions can make power management appear as latency. Scheduled routing controls radio activity from a very high level, assuming that no other communication services are involved. Most published work only deals with powering the radio, not the actual processor beneath these power-aware protocols: dropping from active to full sleep reduces power draw a hundred-fold, while shutting off the radio only cuts it in half. Ultimately, power management requires a holistic approach, so potentially multiple protocols can establish a cooperative power schedule.

Long term sensor network deployments have made headway on this problem, using a vertical slice approach. In the Great Duck Island deployment, each TinyOS component include a standard power interface, allow to act and convey power transitions to other portions of the stack. The application level protocol goes into virtual sleep for long periods of time, the physical layer wakes periodically to sample for preambles, and the multihop routing layer between has an event-driven schedule. TinyDB adopts a related approach, but shuts down the whole stack between query processing epochs[29].

The SNA challenge is to generalize these techniques into a set of interfaces and a composable usage discipline that allows individual components of the network architecture to establish their portion of the overall power scheduling apparatus.

5.4 Network Management

Sensor networks must be able to monitor themselves, adapt to changes, and automatically self-manage resources. However, the network architecture should also provide a range of management capabilities that involve users, to assist in development and testing of protocols, identifying problems, and assess performance. These capabilities need to touch many layers of the network architecture so that low-level tools can test higher levels without depending on them. These tools will greatly depend on what protocols and systems are built; in this section we enumerate several we believe will be important and should be considered in the architecture.

Node-local failsafes can provide a stable means of responding to node failures. Without knowing failure specifics, watchdogs can reset or disable a node if regular operation seems to have been disrupted (e.g., it has stopped receiving packets). Services, protocols, and applications should be able to introduce watchdog checks, for validation and monitoring across the entire system. Fixed code segments and bootloaders can enable remote reprogramming even if an application has failed. Combined with failsafes, node enumeration allows an administrator to obtain a rough estimate of the state of network in terms of how many individual nodes are active or disabled.

Administrators must also be able to examine the performance of a deployment on a holistic, network-wide scale. Deployments have shown that nodes can enter into failure modes that cause them to broadcast meaningless data and impair the radio environment for nearby nodes [33]. For events and anomalous behavior that on-node watchdogs do not or cannot check for, administrators need higher level tools that collect and examine data from multiple nodes in the network.

One of our goals is to incorporate interfaces into the SNA that support this functionality.

5.5 Security

Sensor network security should include clear demarcation and enforcement of administrative domains, especially in situations where multiple different sensor networks may be physically co-located. Current approaches to this problem focus on establishment of shared keys, and then consequent use of these keys for authentication and encryption at the local packet link level. Future work is likely to include mechanisms for enforcing resource sharing policies across trust domains to enable greater efficiency.

Because the adaptability of a sensor network rests on a set of heuristics executing at local nodes designed to produce desired global behavior, attacker compromise of individual nodes can easily prevent effective action in the rest of the network. The deeply embedded nature of these networks, and the likelihood that they are deployed in inaccessible areas, provides even more opportunities for attacker node compromise. Security can be integrated into the topology management layer by using redundant routes to detect and correct unauthorized data modification in transit.

At the application level, the data gathering functionality of a sensor network can be subverted by introducing flawed data at points in the network. Using techniques similar to Byzantine fault tolerant communication, data can be cross-checked for consistency and malicious data detected. In-network aggregation introduces strong leverage points for data modification, and requires even more careful attention to trust and resiliency.

Effective security must be considered within the context of every component in the SNA.

6 System Support

The sensor network architecture should be independent of any particular operating system for the nodes so that it might be realized on a range of systems. However, high quality system support can greatly facilitate both the development and the efficacy of the architecture. In the Internet, designs were developed primarily in the context of Unix and ported to other systems and the impact of the system support has been felt primarily in high performance settings. In sensor nets the quality of the system support is much more important because the available resources are so constrained, devices are inaccessible, and applications are long lived.

We intend to extend the TinyOS environment in four primary ways to support the emerging network architecture: encapsulation, buffer management, robustness, and scheduling. Encapsulation is critical in the implementation of network stacks. As packets are transmitted down the stack, each layer effectively envelopes the payload provided from above. On the way back up, on receive, payloads are extracted from within lower level frames. We want to be able to encapsulate or de-encapsulate as packets move through the stack without copying or indirection (i.e., gather scatter). The application specificity and the constrained resources makes it possible to perform whole system analysis in the TinyOS setting so that allocation can occur in one place with offsets established for the intervening layers, while maintaining the information hiding needed to keep payloads of one layer opaque to the layer below. Efficient and safe buffer management is closely related to encapsulation, but in addition to size determination, management schemes that prevent overruns and dangling pointers, while permitting buffer reuse are desired. And, the behavior must be well defined as unanticipated rate mismatches and other phenomena occur. For example, TinyOS provides a strict buffer swapping discipline, so allocation for the worst-case is performed statically, except where sharing is explicitly implemented through buffer pooling components.

Robustness is improved by providing narrow, fully typed interfaces. Protocols work in peer relationships across nodes. Packet formatting and access is specified in the program. Support should be provided so that packets are sufficiently typed that all parsing and formatting is performed by the compiler, rather than by serial dynamic field extraction. In addition, the asynchronous and interleaved nature of network stack operation, as well as the use of timers and events make network stack scheduling intensive. Target scheduling support can greatly simplify protocol implementation and improve robustness. Our starting points are the bidirectional interfaces and event driven scheduling of TinyOS.

7 Research Plan

We envision the research in this proposal being comprised of three overlapping phases.

Phase 1: Layer Definition, Interface Design. We will design each of the layers of the SNA and the interfaces between the layers and their component protocols. The definition of SP forms the keystone. Relative to it, we will resolve the architectural questions italicized in each section above. During this phase, we will actively seek the insights and

feedbacks from other research groups, both through informal conversations and more formal workshops. A particular focus is articulating and applying design principles.

Phase 2: Testing the functional decomposition and set of interfaces through a series of test implementations. The focus here will be on trying to implement rather different versions of each component (*e.g.*, the various styles of aggregation, detection, and time synchronization) to a common interface, and trying to use them in widely varying applications. This will require the construction of a new testbed facility. We will again seek to engage other research groups in this stage of our work.

Phase 3: Building several fully composed prototype applications, with a large degree of code reuse across quite different uses. If we are the only group doing this, then we will have failed. The goal is that other research groups find the framework useful enough so that they too are actively using it to build their systems.

While at any one time the focus will be on one of these phases, parallel work on the other phases will help us peek into the future and check our past results. Year 1 focuses on phase 1. We will hold a graduate seminar at Berkeley that identifies a moderate number of internally consistent high level architectures and conducts protocol realization studies within each for comparison. At the end of year 1 we will host a workshop to present our findings and engage the NETS community in developing protocols within the resulting framework. Years 2 and 3 will focus on phase 2, while iterating back over phase 1. To this end we will construct a sizable testbed, including multiple link types, and supporting a range of network protocols. We will make the testbed and its network stacks available to the NETS community over the web. This will provide a competitive winnowing process for interface definitions and associated protocol designs. A second seminar will be held focused on protocol design relative to the architecture and a second workshop will be held focused on evaluation techniques. Years 3 and 4 will be increasingly devoted to phase 3. The success we have had in getting literally hundreds of research groups to build upon our TinyOS platform gives us confidence in the outcome, but here we must observe how the differing investigation use the overall architecture, not just the underlying platform.

It is difficult to evaluate an architecture. Its worth can only be demonstrated by observing, over time, whether the architecture has proven useful in its goals. That is, after several years one can ask question such as: Have the set of abstractions proven useful for organizing components, and clarifying their implementation? Have several quite different applications been realized within this framework with a high degree of code reuse? And the ultimate question is: Have others adopted this framework for their own systems?

8 Results for Prior NSF Support

David Culler was PI on NSF RI Grant “SimMillennium: A Large-Scale System of Systems Organized as a Computational Economy for Simulation and Modeling” (EIA-9802069). It provided a powerful research infrastructure for computational modeling and simulation of Internet scale systems and facilitated the design of *computational economies*[49, 10, 34]. Its widely used cluster software distribution served as the starting point for PlanetLab. He is co-PI on PlanetLab: An Overlay Testbed for Disruptive Network Services (ANI-0335289). It provides an international testbed for networking research containing 360 machines and 148 sites in 26 countries. Recent publications include [3, 39]. He is Senior Personnel on ITR/SI: Societal Scale Information Systems: Technologies, Design and Applications” (EIA-0122599) which has supported the application of sensor network technology, including habitat monitoring[33, 48], structural monitoring of the Golden Gate Bridge, and redwood ecophysiology using the TinyDB in-network query processor[32, 14].

Scott Shenker has been a PI or co-PI on eight NSF grants during the past five years. The one most closely related to the current proposal was IIS-0330178 (\$500,000, September 1, 2003 through August 31, 2006, co-PI Ramesh Govindan of USC) entitled “Robust and Efficient Data Dissemination for Data-Centric Storage.” Preliminary results have been obtained in several areas (*e.g.*, common pitfalls of geographic routing, reliable geographic routing, routing without geography, and routing with structural information) and have been reported in [41, 27, 4].

Ion Stoica has been PI and co-PI on three NSF proposals. The most closely related proposal to this one is the Associative Overlay Networks, NSF CAREER Award ANI-0133811, \$497,896, March 1, 2002-February 29, 07. This project led to the development of *i3* (the Internet Indirection Infrastructure) [46], an overlay infrastructure that provides support for the basic communication primitives: unicast, mobility [57], multicast, anycast, and service composition. This flexibility allows users not only to build new applications such as heterogeneous multicast, but also to better defend against denial of service attacks [28]. *i3* is available as a service on PlanetLab, and a publically available software is at <http://i3.cs.berkeley.edu>.

References

- [1] Ivy - a sensor network infrastructure for the college of engineering. <http://www-bsac.eecs.berkeley.edu/projects/ivy/>.
- [2] M. Allen, M. Hamilton, and J. Rotenberry. Research infrastructure: James reserve local area power system and network enhancements. <http://cens.ucla.edu>.
- [3] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating systems support for planetary-scale network services. In *First USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI)*, 2004.
- [4] F. Bian, R. Govindan, and S. Shenker. Using hierarchical location names for scalable routing and rendezvous in wireless sensor networks. Under submission.
- [5] Bluetooth SIG, Inc. <http://www.bluetooth.org>.
- [6] N. Bulusu, V. Bychkovskiy, D. Estrin, and J. Heidemann. Scalable, ad hoc deployable, rf-based localization. In *Proceedings of the Grace Hopper Conference on Celebration of Women in Computing*, Vancouver, Canada, October 2002.
- [7] 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 Workshop on Data Communications in Latin America and the Caribbean*, Apr. 2001.
- [8] A. Cerpa and D. Estrin. Adaptive self-configuring sensor networks topologies. In *Proceedings of the Twenty First International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2002.
- [9] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *International Conference on Mobile Computing and Networking (MobiCom 2001)*, pages 85–96, Rome, Italy, July 2001.
- [10] B. Chun and D. Culler. Market-based proportional resource sharing for clusters. Technical Report CSD-1092, 2000.
- [11] D. D. Clark. The design philosophy of the DARPA internet protocols. In *SIGCOMM*, pages 106–114, Stanford, CA, Aug. 1988. ACM.
- [12] J. Elson and D. Estrin. An address-free architecture for dynamic sensor networks. [cite-seer.nj.nec.com/elson00addressfree.html](http://citeseer.nj.nec.com/elson00addressfree.html).
- [13] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, USA., dec 2002.
- [14] W. H. et al. Tiny application sensor kit (task). <http://berkeley.intel-research.net/task/>, 2004.
- [15] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34. ACM Press, 2003.
- [16] R. Fonesca, D. Culler, S. Ratnasamy, S. Shenker, and I. Stoica. Beacon vector routing: Scalable point-to-point routing in wireless sensor networks. In submission.
- [17] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. An evaluation of multi-resolution storage for sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 89–102. ACM Press, 2003.
- [18] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. An empirical study of epidemic algorithms in large scale multihop wireless networks. UCLA Computer Science Technical Report UCLA/CSD-TR 02-0013, 2002.
- [19] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker. Difs: A distributed index for features in sensor networks. In *Proceedings of First IEEE International Workshop on Sensor Network Protocols and Applications*, 2003.
- [20] J. Hill and D. E. Culler. Mica: a wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, nov/dec 2002.
- [21] J. Hill, M. Horton, R. Kling, and L. Krishnamurthy. The Platforms Enabling Wireless Sensor Networks. In *CACM Special Issue on Sensor Nets*, June 2004. To appear.
- [22] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, Boston, MA, USA, Nov. 2000.

- [23] C. Inc. Cc2420 data sheet. http://www.chipcon.com/files/CC2420_Data_Sheet_1_0.pdf, 2003.
- [24] C. Intanagonwivat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the International Conference on Mobile Computing and Networking*, Aug. 2000.
- [25] C. Intanagonwivat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11(1):2–16, 2003.
- [26] R. E. Kahn, S. A. Gronemeyer, J. Burchfiel, and R. C. Kunzleman. Advances in packet radio technology. *Proceedings of the IEEE*, 66(11):1468–1496, Nov. 1978.
- [27] Y. Kim, R. Govindan, B. Karp, , and S. Shenker. Practical and robust geographic routing in wireless networks. Under submission.
- [28] K. Lakshminarayanan, D. Adkins, A. Perrig, and I. Stoica. Taming ip packet flooding attacks. In *Hotnets-II*, Cambridge, MA, Nov. 2003.
- [29] 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 *First USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI)*, 2004.
- [30] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code maintenance and propagation in wireless sensor networks. In *First USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI)*, 2004.
- [31] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data*, pages 491–502. ACM Press, 2003.
- [32] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the ACM Symposium on Operating System Design and Implementation (OSDI)*, Dec. 2002.
- [33] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications*, Sept. 2002.
- [34] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. Submitted for publication, 2003.
- [35] J. Newsome and D. Song. Gem: graph embedding for routing and data-centric storage in sensor networks without geographic information. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 76–88. ACM Press, 2003.
- [36] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162. ACM Press, 1999.
- [37] R. Patra and S. Nedeveschi. Dtnlite: Delay tolerant networking on constrained devices. www.cs.berkeley.edu/~rkpatra/cs294_deep/deep_proj/deep_rabin_sergiu/dtnlite_rabin_sergiu.pdf.
- [38] C. E. Perkins, E. M. Belding-Royer, and S. Das. Ad hoc on demand distance vector (AODV) routing. IETF Internet draft, draft-ietf-manet-aodv-09.txt, November 2001 (Work in Progress).
- [39] L. Peterson, D. Culler, T. Anderson, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets-I)*, 2002.
- [40] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In submission.
- [41] A. Rao, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 96–108. ACM Press, 2003.
- [42] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. Ght: a geographic hash table for data-centric storage. In *Proceedings of the first ACM international workshop on Wireless sensor networks and applications*, pages 78–87. ACM Press, 2002.
- [43] S. Rhee, D. Seetharam, S. Liu, N. Wang, and J. Xiao. i-Bean Network: An Ultra-Low Power Wireless Sensor Network. In *UbiComp 2003, the Fifth International Conference on Ubiquitous Computing*, Oct. 2003.
- [44] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin. Data-centric storage in sensornets. *SIGCOMM Comput. Commun. Rev.*, 33(1):137–142, 2003.

- [45] S. Singh and C. S. Raghavendra. Pamas: power aware multi-access protocol with signalling for ad hoc networks. *SIGCOMM Comput. Commun. Rev.*, 28(3):5–26, 1998.
- [46] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *SIGCOMM*, pages 73–86, Pittsburgh, PA, Aug. 2002.
- [47] S. Systems and C. Technology. Sensicast development software. <http://www.xbow.com>, 2004.
- [48] R. Szweczyk, J. Polastre, A. Mainwaring, and D. Culler. An analysis of a large scale habitat monitoring application. In submission.
- [49] UC Berkeley Ninja Project. The UC Berkeley Ninja Project. <http://ninja.cs.berkeley.edu>.
- [50] M. Welsh and G. Mainland. Programming sensor networks with abstract regions. In *First USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI)*, 2004.
- [51] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: a neighborhood abstraction for sensor networks. 2004.
- [52] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 14–27. ACM Press, 2003.
- [53] Y. Xu, J. S. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, 2001.
- [54] Y. Yao and J. Gehrke. The Cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002.
- [55] M. Yarvis and W. Ye. Tiered architectures in sensor networks. In M. Ilyas, editor, *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*. CRC Press, 2003.
- [56] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the First International Conference on Embedded Network Sensor Systems*, 2003.
- [57] S. Zhuang, K. Lai, I. Stoica, R. Katz, and S. Shenker. Host mobility using an Internet Indirection Infrastructure. In *MobiSys*, pages 129–144, San Francisco, CA, May 2003.

Contents

1	What is a Sensor Network Architecture (SNA)?	2
2	Physical Architecture	3
3	Programming Architecture	4
4	Functional Architecture	6
4.1	The Sensor-net Protocol	7
4.2	Address-Free Protocols	7
4.3	Name-based Protocols: Topology Management and Routing	8
4.4	In-Network Storage	9
4.5	Active In-Network Storage	10
5	Cross-Layer Issues	11
5.1	Discovery	11
5.2	Time Coordination	12
5.3	Power Management	13
5.4	Network Management	13
5.5	Security	14
6	System Support	14
7	Research Plan	14
8	Results for Prior NSF Support	15