Storage Management in Wireless Sensor Networks

Sameer Tilak[†], Nael Abu-Ghazaleh[†] and Wendi B. Heinzelman[‡]

[†]Department of Computer Science
Binghamton University
Binghamton, NY, USA

[‡]Department of Electrical and Computer Engineering
University of Rochester
Rochester, NY, USA

March 5, 2006

Abstract

Storage management is an area of sensor network research that is starting to attract attention. The need for storage management arises primarily in the class of sensor networks where information collected by the sensors is not relayed to observers in real-time (e.g., scientific monitoring applications). In such applications, the data must be stored, at least temporarily, within the network until it is later collected by an observer (or until it ceases to be useful). Therefore, storage becomes a primary resource which, in addition to energy, determines the useful lifetime and coverage of the network.

This chapter overviews issues and opportunities in storage management for sensor networks. We first motivate the need for storage management by overviewing sensor network applications that require storage management. We then discuss how the application characteristics and various resource constraints influence the design of storage management techniques. More specifically, we break down the storage management problem into the following components: (1) system support for storage management; (2) collaborative storage; and (3) indexing and retrieval. We then highlight the problems and issues in each of these components of storage management, and we describe proposed solutions.

Keywords - Storage management, Collaborative storage, Indexing and retrieval, Memory, Non-real-time applications

1 Introduction

Wireless Sensor Networks (WSNs) hold the promise of revolutionizing sensing across a range of civil, scientific, military and industrial applications. However, many battery-operated sensors have constraints such as limited energy, computational ability, and storage capacity, and thus protocols must be designed to deal efficiently with these limited resources in order to maximize the data coverage and useful lifetime of the network.

Storage management is an area of sensor network research that is starting to attract attention. The need for storage management arises primarily in the class of sensor networks where information collected by the sensors is not relayed to observers in real-time. In such applications, the data must be stored, at least temporarily, within the network until it is later collected by an observer (or until it ceases to be useful). An example of this type of application is scientific monitoring, where the sensors are deployed to collect detailed information about a phenomenon for later playback and analysis. Another example is that of sensors that collect data that is later accessed by dynamically generated queries from users. In these types of applications, data must be stored in the network, and thus storage becomes a primary resource which, in addition to energy, determines the useful lifetime and coverage of the network.

With the knowledge of the relevant application and system characteristics, a set of goals for sensor network storage management can be determined. These requirements are: (1) minimizing storage size to maximize coverage/data retention; (2) minimizing energy; (3) supporting efficient query execution on the stored data (note that in the reachback method where all the data must be sent to the observer, query execution is simply the transfer of the data to the observer); and (4) providing efficient data management under constrained storage. Several approaches to storage management have been proposed to meet the above requirements, with most approaches involving a tradeoff among these different goals.

One basic storage management approach is to buffer the data locally at the sensors that collect them. However, such an approach does not capitalize on the spatial correlation of data among neighboring sensors to reduce the overall size of the stored data (the property that makes data aggregation possible [11]). Collaborative storage management, on the other hand, can provide the following advantages over a simple buffering technique.

- More efficient storage allows the network to continue storing data for a longer time without exhausting storage space.
- Load balancing is possible. If the rate of data generation is not uniform at the sensors (e.g., in the case where a localized event causes neighboring sensors to collect data more aggressively), some sensors may run out of storage space while space remains available at others. In such a case, it is important for the sensors to collaborate to achieve load balancing for storage to avoid or delay data loss due to insufficient local storage.
- Dynamic, localized reconfiguration of the network (such as adjusting sampling frequencies of sensors based on estimated data redundancy and current resources) is possible.

Thus, collaborative storage is often better able to meet the goals of storage management.

This chapter overviews issues and opportunities in storage management for sensor networks. We first motivate the need for storage management by overviewing sensor network applications that require storage management. We then discuss how the application characteristics and various resource constraints influence the design of storage management techniques. Application characteristics define what data is useful and how the data will eventually be accessed; these features have significant implications on storage protocol design. In addition, hardware characteristics determine the capabilities of the system and the energy efficiency of the storage protocols. We overview the characteristics of flash memory, the prevalent storage technology in sensor networks, and we compare the cost of storage to that of computation and communication to establish the basic data manipulation costs using current technologies for subsequent tradeoff analyses. Then, as a case study, we discuss the design and implementation of the matchbox file system, which supports mote based applications.

2 Motivation – Application Classes

In this section, the storage management problem is motivated by describing two application classes that require effective use of storage resources. In general, storage is required whenever data is not relayed to observers outside the network in real-time; the data must be stored by the sensors until it is collected or discarded (it ceases to be useful, or it is sacrificed/compressed to make room for more important data). The following two applications illustrate different situations where such a requirement arises.

2.1 Scientific Monitoring – Playback Analysis

In scientific monitoring applications, the sensor network is deployed to collect data. This data is not of real-time interest; it is collected for later analysis to provide an understanding of some on-going phenomenon. Consider a wildlife tracking sensor network that scientists could use to understand the social behavior and migratory patterns of a species. Suppose sensors are deployed in a forest and collect data about nearby animals. In such an application, the sensors may not have an estimate regarding the observer’s schedule for accessing the data. Furthermore, the data generation rate at the sensors may be unpredictable, as it depends on the observed activity, and the sensor density may also be non-uniform [5]. The observer would like the network to preserve the collected data samples, and the collection time should be small since the observer may not be in range of the sensors for very long. An example of such a network is the ZebraNet project [13].

Scientists (observers) collect the data by driving around the monitored habitat, receiving information from sensors as they come in range with them. Alternatively, the data may be relayed by the sensors in a multi-hop fashion towards the observer. Data collection is not pre-planned: it might be unpredictable and infrequent. However, the data “query” model is limited (one time collection in a known fashion). This information about data access patterns may be exploited; for example, data may be stored at sensors closer to the observer. The long term availability of the data allows aggregation/compression that can be significantly more efficient than that achieved by real-time data collection sensor networks.

This type of situation also occurs in sensor networks that report their data in near real-time when the network becomes partitioned. For example, in the Remote Ecological Micro-Sensor Network [22], remote visual surveillance of federally listed rare and endangered plants is conducted. This project aims to provide near-real-time monitoring of important events, such as visitation by pollinators and consumption by herbivores, along with monitoring a number of weather conditions and events. Sensors are placed in different habitats, ranging from scattered low shrubs to dense tropical forests. Environmental conditions can be severe; e.g., some locations frequently freeze. In such applications, network partitioning (relay nodes becoming unavailable) may occur due to the extreme physical conditions (e.g., deep freeze). Important events that occur during disconnection periods should be recorded and reported once the connection is reestablished. Effective storage management is needed to maximize the partitioning time that can be tolerated without loss of data.

2.2 Augmented Reality – Multiple Observers, Dynamic Queries

Consider a sensor network that is deployed in a military scenario and collects information about nearby activity. The data is queried dynamically by soldiers to help with mission goals or with avoiding sources of danger, and by commanders to assess the progress of the mission. The queried data is both real-time as well as long-term data about enemy activity (e.g., to answer a question such as where are the supply lines). Thus, data must be stored at the sensors to enable queries that span temporally long periods, such as days or even months. One can envision similar applications with sensor networks deployed in other contexts that answer questions about the environment using both real-time as well as recent or even historical data. The data is stored at the sensors and used, perhaps collaboratively, to answer queries.

In addition to efficiently using data storage to allow the sensor network to retain more data and higher resolution data, one of the issues in this application is effective indexing and retrieval of the data. While the data may be addressed by content (e.g., searching for information about a certain type of vehicle), since the observers do not know where the data are stored, or even what data exist, the queries can be quite inefficient. Furthermore, data may be accessed multiple times, by different observers. Efficient indexing and retrieval of the data is desirable.

3 Preliminaries: Design Considerations, Goals and Storage Management Components

The applications described in the previous section provide insight into the range of issues that must be addressed to achieve effective storage management. In this section, these issues are described more directly. We first outline the factors that influence the design of storage management and then discuss the design goals of such a system. We also break the storage management problem into different components that require efficient protocols.

3.1 Design Considerations

Energy is a precious resource in all wireless micro-sensor networks; when the battery energy at a sensor expires, the node ceases to be useful. Therefore, preserving energy is a primary concern that permeates all aspects of sensor network design and operation. For storage bound applications, there is an additional finite resource: the available storage at the sensors. Once the available storage space is exhausted, a sensor can no longer collect and store data locally; the sensors that have run out of storage space cease to be useful. Thus, a sensor network's utility is bound by two resources: its available energy and its available storage space. Effective storage management protocols must balance these two resources to prolong the network's useful lifetime.

The two limiting resources – storage and energy – are fundamentally different from each other. Specifically, storage is re-assignable while energy is not: a node may free up some storage space by deleting or compressing data. Furthermore, storage at other nodes may be utilized, at the cost of transmitting the data. Finally, the use of storage consumes energy. However, storage devices currently consume less energy than wireless communication devices.

The tradeoff between storage and energy is intricate. Sensors may exchange their data with nearby sensors. Such exchanges allow nearby sensors to take advantage of the spatial correlation in their data to reduce the overall data size. Another positive side effect is that the storage load can be balanced even if the data generation rates or the storage resources are not. However, the exchange of data among the sensors consumes more energy in the data collection phase, as in current technologies, the cost of storage is significantly smaller than the cost of communication. On the surface, it may appear that locally storing data is the most energy efficient solution. However, the extra energy spent in exchanging data may be counterbalanced by the energy saved by storing smaller amounts of data and, more importantly, by the smaller energy expenditure when replying to queries or relaying the data back to observers.

The above discussion pertains to data collection and storage. Another aspect of the storage problem is how to support observer queries on the data – the indexing and retrieval problem. In one of the motivating applications (scientific monitoring and partitioning), data will be relayed once to a possibly known observer. For these applications, storage may be optimized if the direction of the observer is known: while conducting collaborative storage, data can be moved towards the observer, making the storage-related data exchange effectively free. However, in the second application type, dynamic queries for the data from unknown observers can occur. This problem is logically similar to indexing and retrieval in peer-to-peer systems.

3.2 Storage Management Goals

In light of the above discussion, a storage management approach must balance the following goals.

- **Minimize Size of Stored Data.** Since sensors have limited storage available to them (state of the art sensors have approximately 4 Mb of flash memory), minimizing the size of data that need to be stored leads to improved coverage/data retention since the network can continue storing data for longer periods of time. Furthermore, query execution becomes more efficient if the data size is small.

- **Minimize Energy Consumption.** Most of the sensors are battery powered and thus energy is a scarce resource, requiring that storage management be as energy efficient as possible.
- **Maximize Data Retention/Coverage.** Collecting data is the primary goal of the network. If storage is constrained, data re-allocation must be carried out efficiently: in certain cases, if no more storage space is available to store new data, some of the less important data that are already stored need to be deleted. The management protocol should attempt to retain relevant data at an acceptable resolution, where relevancy and acceptable resolution are application dependent.
- **Efficient Query Execution.** Whether the queries are dynamically generated or static (e.g., the data relayed to an observer once), storage management can influence the efficiency of query execution. For example, query execution efficiency can be improved by effective data placement and indexing. Query execution efficiency can be measured in terms of the communication overhead and energy consumption required to get the requested data to the observer.

3.3 Storage Management Components

We break down the storage management problem into the following components: (1) system support for storage management; (2) collaborative storage; and (3) indexing and retrieval. Each of these components have a unique design space to consider, and some preliminary research has been done to improve each of these three components from the perspective of energy and storage efficiency. Furthermore, each of these components exhibit tradeoffs in the design goals described above. In the following sections, we highlight the problems and issues in each of these components of storage management, and we describe proposed solutions.

4 System Support for Storage

Because of the limited available energy on sensors, energy efficiency is a primary objective in all aspects of their design. In this section, the focus is on the system design issues that are most relevant to storage. These include design of hardware components and the sensor network file system.

4.1 Hardware

Magnetic disks (hard drives) are the most widely used persistent storage devices for desktop and laptop computers. However, power, size and cost considerations make hard disks unsuitable for micro-sensor nodes. Compact flash memories are the most promising technology for storage in sensor networks, as they have excellent power dissipation properties compared to magnetic disks. In addition, their prices have been dropping significantly. Finally, they have a smaller form factor than magnetic disks.

Table 1: Energy Characteristics MICA-2 components.

Component	Current	Duty Cycle (%)
Processor		
Full operational	8mA	1
Sleep	$8\mu A$	99
Radio		
Receive	8mA	0.75
Transmit	12mA	0.25
Sleep	$2\mu A$	99
Logger Memory		
Write	15mA	0
Read	4 mA	0
Sleep	$2\mu A$	100

As a case study, we consider the family of Berkeley motes [27], MICA-2, MICA2DOT and MICA nodes, which feature a 4-Mbit serial flash (non-volatile) memory for storing data, measurements, and other user-defined information. TinyOS [10] supports a micro file system that manages this flash/data logger component. The serial flash device supports over 100,000 measurement readings, and this device consumes 15 mA of current when writing data. Table 1 gives the energy characteristics of important hardware components of Mica-2 motes [27].

4.2 File System Case Study: Matchbox

Traditional file systems (such as FFS [17] and Log structured file system [24]) are not suitable for sensor network environments. They are designed and tuned for magnetic disks that have different operational characteristics than Flash memories that are typically used for storage in sensors. For example, traditional file systems employ clever mapping and scheduling techniques to reduce seek time since this is the primary cost in disk access. In addition, these file systems support a wide range of sophisticated operations making them unsuitable for a resource constrained embedded environment. For example, typically file systems support hierarchical filing, security (in terms of access privilege and data encryption in some cases) and concurrent read/write support. Finally, sensor storage disk access patterns are not typical of traditional file system workloads. New technological constraints, low resources and different application requirements demand the design of a new file system for sensor network applications.

One component of the TinyOS [10] operating system is an evolving micro file system called *Matchbox* [6] [7]. Matchbox is designed specifically for mote-based applications and has the following design goals:

1. Reliability: Matchbox provides reliability in the following two ways.
 - (a) Data corruption detection: Matchbox maintains CRCs to detect errors.

- (b) Metadata updates are atomic and therefore the file system is resilient to failures such as power down. Data loss is limited only to files being written at the time of failure.

2. Low Resource consumption.

3. Meeting technological constraints: Some flash memories have constraints that must be met. For example, the number of times a memory location can be written to may be limited.

These design goals translate into a file system that is very simple. Matchbox stores files in an unstructured fashion (as a byte stream). It only supports sequential reads and append only writes. The current design does not aim at providing security, hierarchical file-system support, random access or concurrent read/write access. Typical clients of Matchbox are TinyDB [15], generic sensor kit for data logging applications, and the virtual machine for storing programs.

Matchbox divides the flash memory into sectors (mostly of 128K) and then each sector is divided into pages. Each page is of size 264 bytes, divided into 256 bytes of data and 8 bytes of meta-data. Free pages are tracked using a bitmap.

At present, both the file system and sensor network applications are evolving. Sensor network applications offer a different workload than traditional applications. We believe that as these applications mature and become better understood, the design of the file system will evolve to support these canonical applications. The file system evolution may include changes in the way the files are stored (extending the byte stream view), and supporting a new set of file operations for real-time streaming data (extending append only write operations).

Unfortunately, existing collaborative storage management studies have not considered their interaction with the file-system. It would be interesting to study how different data manipulations can be supported by Matchbox. For example, in one of the storage management protocols, old data is gracefully degraded in storage-constrained conditions [4]. With append-only writes, deleting intermediate data might require rewriting much of (even most of) the existing data. This might result either in extending append-only writes to other techniques or modification of the application itself. Still, implications of such operations on life-time of flash memory (recall that a given memory location can be written only a certain number of times) and energy-consumption are worth exploring. Moreover, file system designers should consider the requirements of collaborative storage when designing sensor file systems.

5 Collaborative Storage

A primary objective of storage management protocols is to efficiently utilize the available storage space to continue collecting data for the longest possible time without losing samples in an energy efficient manner. In this section, we describe collaborative storage management protocols that have been proposed and discuss the important design tradeoffs in these different approaches.

5.1 Collaborative Storage Design Space

Storage management approaches can be classified as:

1. *Local storage*: This is the simplest solution whereby every sensor stores its data locally. This protocol is energy efficient during the storage phase since it requires no data communication. Even though the storage energy is high (due to all the data being stored), the current state of technology is such that storage costs less than communication in terms of energy dissipation. However, this protocol is storage inefficient since the data are not aggregated and redundant data are stored among neighboring nodes. Furthermore, local storage is unable to balance the storage load if data generation or the available storage varies across sensors.
2. *Collaborative storage*: Collaborative storage refers to any approach where nodes collaborate. It can provide the following advantages over a simple buffering technique: (1) More efficient storage allows the network to continue storing data for a longer time without exhausting storage space; (2) Load balancing is possible: if the available storage varies across the network, or if the rate of data generation is not uniform at the sensors (e.g., in the case where a localized event causes neighboring sensors to collect data more aggressively), some sensors may run out of storage space while space remains available at others. In such a case, it is important for the sensors to collaborate to achieve load balancing for storage to avoid or delay data loss due to insufficient local storage; and (3) Dynamic, localized reconfiguration of the network (such as adjusting sampling frequencies of sensors based on estimated data redundancy and current resources) is possible.

It is important to consider the energy implications of collaborative storage relative to local storage. Collaborative storage requires sensors to exchange data, causing them to expend energy during the storage phase. However, because they are able to aggregate data, the energy expended in storing this data to a storage device is reduced. In addition, once connectivity with the observer is established, less energy is needed during the collection stage to relay the stored data to the observer.

5.2 Collaborative Storage Protocols

Within the space of collaborative storage, a number of protocols have been proposed. One such protocol is the Cluster Based Collaborative Storage (CBCS) protocol [26]. CBCS uses collaboration among nearby sensors only: these have the highest likelihood of correlated data and require the least amount of energy for collaboration. Wider collaboration was not considered because the collaboration cost may become prohibitive, due to the fact that the energy cost of communication is significantly higher than the energy cost of storage under current technologies. The remainder of this section describes CBCS operation.

In CBCS, clusters are formed in a distributed, connectivity-based or geographically-based fashion — almost any one-hop clustering algorithm would suffice. Each sensor sends its observations to the elected Cluster Head (CH) periodically. The CH then aggregates the observations and stores the aggregated data. Only the CH needs to store aggregated data, thereby resulting in low storage. The clusters are rotated periodically to balance the storage load and energy usage. Note that only the CH needs to keep its radio on during its tenure, while a cluster member can turn off its radio except when it has data to send. This results in high energy efficiency, since idle power consumes

significant energy in the long run if radios are kept on. Furthermore, this technique reduces the reception of unnecessary packets by cluster members, which can also consume significant energy.

Operation during CBCS can be viewed as a continuous sequence of rounds until an observer/base station is present and the reach-back stage can begin. Each round consists of two phases. The first phase is called the CH Election phase. In this phase, each sensor advertises its resources to its one hop neighbors. Based on this resource information, a cluster head (CH) is selected according to a clustering protocol. The remaining nodes then attach themselves to a nearby CH. The second phase is the data exchange phase. If a node is connected to a CH, it sends its observations to the CH; otherwise, it stores its observations locally.

The CH election approach used in CBCS is based on the characteristics of the sensor nodes such as available storage, available energy or proximity to the “expected” observer location. The criteria for CH selection can be arbitrarily complex; experiments presented used available storage as the criteria. CH rotation is done by repeating the cluster election phase with every round. The frequency of cluster rotation influences the performance of the protocol, as there is overhead for cluster formation due to the exchange of messages. Thus cluster rotation should be done frequently enough to balance storage/energy yet not so often to make the overhead of cluster rotation prohibitive.

5.3 Coordinated Sensor Management

Using a clustering approach such as the one for CBCS enables coordination that can be used for redundancy control to reduce the amount of data actually generated by the sensors. Specifically, each sensor has a local view of the phenomenon, but cannot assess the importance of its information given that other sensors may report correlated information. For example, in an application where 3 sensors are sufficient to triangulate a phenomenon, 10 sensors may be in a position to do so and be storing this information locally or sending it to the cluster head for collaborative storage. Through coordination, the cluster head can inform the nodes of the degree of the redundancy, allowing the sensors to alternate triangulating the phenomenon. Coordination can be carried out periodically at low frequency, with a small overhead (e.g., with CH election). Similar to CH election, the nodes exchange meta data describing their reporting behavior, and it is assumed that some application specific estimate of redundancy is performed to adjust the sampling rate.

As a result of coordination, it is possible that a significant reduction in the data samples produced by each sensor is achieved. We note that this reduction represents a portion of the reduction that is achieved from aggregation. For example, in a localization application, with 10 nodes in position to detect an intruder, only 3 nodes are needed. Coordination allows the nodes to realize this and adjust their reporting so that only 3 sensors produce data in every period. However, the three samples can still be aggregated into the estimated location of the intruder once the values are combined at the cluster head.

Coordination can be used in conjunction with local storage or collaborative storage. In Coordinated Local Storage (CLS), the sensors coordinate periodically and adjust their sampling schedules to reduce the overall redundancy, thus reducing the amount of data that will be stored. Note that the sensors continue to store their readings locally. Relative to Local Storage (LS), CLS results in

a smaller overall storage requirement and savings in energy in storing the data. This also results in a smaller and more energy efficient data collection phase.

Similarly, Coordinated Collaborative Storage (CCS) uses coordination to adjust the sampling rate locally. Similar to CBCS, the data is still sent to the cluster head where aggregation is applied. However, as a result of coordination, a sensor can adapt its sampling frequency/data resolution to match the application requirements. In this case, the energy in sending the data to the cluster head is reduced because of the smaller size of the generated data, but the overall size of the data is not reduced.

In the next section, we discuss some experimental results for CLS and CCS, and we compare these techniques to their non-coordinated counterparts, LS and CBCS.

5.4 Experimental Evaluation

We simulated the proposed storage management protocols using the NS-2 simulator [19]. We use a CSMA based MAC layer protocol. A sensor field of 350×350 meters² is used, with each sensor having a transmission range of 100 meters. We considered three levels of sensor density: 50 sensors, 100 sensors and 150 sensors deployed randomly. We divide the field into 25 zones (each zone is 70×70 meters² to ensure that any sensor in the zone is in range with any other sensor). The simulation time for each scenario was set to 500 seconds and each point represents an average over five different topologies. Cluster rotation and coordination are performed every 100 seconds in the appropriate protocols.

We assume sensors have a constant sampling rate (set to one sample per second). Unless otherwise indicated, we set the aggregation ratio to a constant value of 0.5. For the coordination protocols, we used a scenario where the available redundancy was on average 30% of the data size — this is the percentage of the data that can be eliminated using coordination. We note that this reduction in the data size represents a portion of the reduction possible using aggregation. With aggregation, the full data is available at the cluster head and can be compressed at a higher efficiency.

5.4.1 Storage and Energy Tradeoffs

Figure 1(a) shows the average storage used per sensor as a function of the number of sensors (50, 100 and 150 sensors) for the four storage management techniques : (1) local storage (LS); (2) Cluster-Based Collaborative Storage (CBCS); (3) Coordinated Local Storage (CLS); and (4) Coordinated Collaborative Storage (CCS). In the case of CBCS, the aggregation ratio was set to 0.5. The storage space consumption is independent of the network density for LS and is greater than the storage space consumption for CBCS and CCS (roughly in proportion to the aggregation ratio). CLS storage requirement is in between the two approaches because it is able to reduce the storage requirement using coordination (we assumed that coordination yields improvement uniformly distributed between 20% and 40%). Note that after data exchange, the storage requirement for CBCS and CCS are roughly the same since aggregation at the cluster head can reduce the data to a minimum size, regardless of whether coordination took place or not.

(a) Storage space vs. Network Density

(b) Energy consumption vs. Network Density

Figure 1: Comparison between local storage (LS), cluster based collaborative storage (CBCS), coordinated local storage (CLS) and coordinated collaborative storage (CCS).

Surprisingly, in the case of collaborative storage, the storage space consumption decreases slightly as the density increases. While this is counter-intuitive, it is due to the higher packet loss observed during the exchange phase as the density increases; as density increases, the probability of collisions increases. These losses are due to the use of a contention based unreliable MAC layer protocol. The negligible difference in the storage space consumption between CBCS and CCS is also an artifact of the slight difference in the number of collisions observed in the two protocols. A reliable MAC protocol such as that in IEEE 802.11 (which uses four-way handshaking) or a reservation based protocol such as the TDMA based protocol employed by LEACH [9] can be used to reduce or eliminate losses due to collisions (at an increased communication cost). Regardless of the effect of collisions, one can clearly see that collaborative storage achieves significant savings in storage space compared to local storage protocols (in proportion to the aggregation ratio).

We assumed Transfer Energy/MB $0.055J$ for flash memory. In the case of ratio, we assumed Transmit Power = $0.0552W$, Receive Power = $0.0591W$, and Idle Power = $0.00006W$. Figure 1(b) shows the consumed energy for the protocols in Joules as a function of network density.

The X-axis represents protocols for different network densities: L and C stand for local buffering and CBCS respectively. L-1, L-2, and L-3 represent the results using the local buffering technique for network size 50, 100 and 150, respectively. The energy bars are broken into two parts: pre-energy, which is the energy consumed during the storage phase, and post-energy, which is the energy consumed during data collection (the relaying of the data to the observer). The energy consumed during the storage phase is higher for collaborative storage because of the data communication among neighboring nodes (not present in local storage) and due to the overhead for cluster rotation. CCS spends less energy than CBCS due to the reduction in data size that results from coordination. However, CLS has higher expenditure than LS since it requires costly communication for coordination. This cost grows with the density of the network because our coordination implementation has each node broadcasting its update and receiving updates from all other nodes.

Figure 2: Percentage of storage depleted sensors as a function of time.

For the storage and communication technologies used, the cost of communication dominates that of storage. As a result, the cost of the additional communication during collaborative storage might not be recovered by the reduced energy needed for storage except at very high compression ratios. This tradeoff is a function of the ratio of communication cost to storage cost; if this ratio goes down in the future (for example, due to the use of infra-red communication or ultra-low power RF radios), collaborative storage becomes more energy efficient compared to local storage. Conversely, if the ratio goes up, collaborative storage becomes less efficient.

5.4.2 Storage Balancing Effect

In this study, we explore the load-balancing effect of collaborative storage. More specifically, the sensors are started with a limited storage space, and the time until this space is exhausted is tracked. We consider an application where a subset of the sensors generates data at twice the rate of the others, for example, in response to higher observed activity close to some of the sensors. To model the data correlation, we assume that sensors within a zone have correlated data. Therefore all the sensors within a zone will report their readings with the same frequency. We randomly select zones with high activity; sensors within those zones will report twice as often as those sensors within low activity zones.

In Figure 2, the X-axis denotes time (in multiples of 100 seconds), whereas the Y-axis denotes the percentage of sensors that have no storage space left. Using LS, in the even data generation case, all sensors run out of storage space at the same time and all data collected after that is lost. In comparison, CBCS provides longer time without running out of storage because of its more efficient storage.

The uneven data generation case highlights the load-balancing capability of CBCS. Using LS, the sensors that generate data at a high rate exhaust their storage quickly; we observe two subsets of sensors getting their storage exhausted at two different times. In comparison, CBCS has much longer mean sensor storage depletion time due to its load balancing properties, with sensors exhausting their resources gradually, extending the network lifetime much longer than LS.

The sensor network coverage from a storage management perspective depends on the event

generate rate, the aggregation properties and the available storage. If the aggregated data size is independent of the number of sensors (or grows slowly with it), the density of the zone correlates with the availability of storage resources. Thus, both the availability of storage resources as well as the consumption of them may vary within a sensor network. This argues for the need of load-balancing across zones to provide long network lifetime and effective coverage. This is a topic of future research.

Collaborative storage reduces storage requirements by taking advantage of temporal and spatial correlations in data from nearby sensors. Furthermore, coordinated storage proactively exploits spatial redundancy in sensor data, thereby reducing not only storage requirements but also energy dissipation in communication of the data as well as in the sensing hardware, which can be turned off on nodes that are not assigned to sense data for the current period. Another approach that takes advantage of the correlations in the data to manage storage space is the multi-resolution technique, described next.

5.5 Multi-Resolution Based Storage

Much existing research, such as Directed Diffusion [12], Data Centric Storage (DCS) [21], TAG [16], Cougar [2], etc. has focused on in network aggregation and query processing when features of interest are known. For example, DCS stores named events at known locations to reduce query overhead. Since the events are constellations of low-level sensor observations, they are not storage intensive. Therefore DCS does not address the issue of managing limited storage space. TAG assumes knowledge of aggregation operators. Recently, Ganesan et al. [4] proposed an in-network wavelet-based summarization technique accompanied by progressive aging of these summaries to support data-intensive applications where features of interest are not known in advance and the networks are storage and communication-constrained. Their system strives to support storage and search for raw sensor data (low-level observations) in data-intensive scientific applications by providing a lossy, progressively degrading storage model. The key factors of this work are wavelet-based spatio-temporal data summarization (construction of multi-resolution summaries of data), a hierarchically decomposed distributed storage structure, drill-down queries, and progressive aging of summaries. We briefly describe each of these features in detail in next few sections.

5.5.1 Multi-Resolution Summarization

The authors proposed the use of a wavelet based data summarization technique that is carried out in two steps, namely temporal and spatial summarization.

Temporal summarization is done by every sensor node locally using techniques such as time series analysis to find redundancy in its own signal. Temporal summarization involves just computation—no communication is needed.

Spatial summarization involves constructing a hierarchy and using spatio-temporal summarization techniques to re-summarize data at each level. Also, data at lower levels can be summarized at higher levels at larger spatial scale but with higher compression (thereby it is more lossy).

5.5.2 Drill Down Queries

The basic idea behind drill-down queries is quite intuitive. Data are summarized at multiple resolutions across the network. Queries are injected at the highest level of hierarchy, which has the coarsest and most highly compressed summary of very large spatio-temporal data. Processing the query over this summary gives an approximate answer/pointer to the part of the network that is very likely to give a more accurate answer, since it has a more detailed view of the subregion. Further queries can then be directed to this region, if needed. This process is applied recursively until the user is satisfied with the accuracy of the result or the leaves of the hierarchy are encountered. Clearly, the accuracy of the result (query quality) improves with more drill-downs since finer data gets queried at lower levels.

Hierarchical summarization and drill down query together address the challenges in searching data in an efficient manner. After describing how to compute the summaries, we now discuss another challenging issue that deals with storage space allocation/reclaim of these summaries in a distributed fashion.

5.5.3 Aging Problem

In storage-constrained networks, a challenging question is to decide how long a summary is to be stored. The length of time for which a summary is stored is called an *age* of the summary. Let $f(t)$ be a monotonically decreasing user-specified aging function that represents the error the user is willing to accept as the data ages in the network. The authors argue that, typically the domain experts can supply this kind of function. As an example, the user might be willing to accept 90% query accuracy for data that is a week old data but only 50% accuracy for data that is a year old. Let us denote the instantaneous quality difference as $qdiff(t)$, which represents the user-specified aging function and the achieved query accuracy at give time t . The aging problem can be defined as follows. Find the ages of summaries, Age_i , at different resolutions such that the maximum instantaneous quality difference is minimized.

$$Min_{0 \leq t \leq T} (Max(qdiff(t))) \quad (1)$$

under the constraints:

- **Drill Down Constraints:** It is not useful to retain a summary at lower levels if its summary at higher levels is not present, since drill down queries will not be directed to the lower levels in that case.
- **Storage Constraints:** Each node has a finite storage space available to the summaries of each level.

The authors proposed three aging strategies, namely omniscient, training-based and greedy strategy, and evaluated their performance.

6 Indexing and Data Retrieval

The final challenge that we discuss in storage management is indexing and retrieval of the data. In the case of the first application type (Scientific Monitoring/Partitioning), the data is relayed back to an observer during reach-back one time. Accordingly, indexing and retrieval is not an important issue for this type of application. Nevertheless, reducing storage size leads to more efficient retrieval in terms of time and energy. Furthermore, it is possible to improve the retrieval performance if the expected location of the observer is known by favoring sensors closer to the observer.

Indexing and retrieval are more important issues in our second application model (Augmented reality), where data can be queried dynamically and by multiple observers. Such networks are inherently data-centric; observers often name data in terms of attributes or content that may not be topologically relevant. For example, a commander may be interested in enemy tank movements. This characteristic of sensor networks is similar to many peer-to-peer (P2P) environments [1] which are also often data-centric. Such a model is in contrast with traditional host-centric applications such as telnet, where the end user is communicating with a specific end-host at the other end. We first overview peer-to-peer solutions to data indexing and retrieval and then show that the properties of sensor networks significantly change the tradeoffs and invite different solutions.

6.1 Design Space – Retrieval in P2P Networks

In P2P networks, the problem of data indexing and retrieval has been attacked in several ways. These approaches can be classified in terms of data placement as: (1) structured: the data is placed at specific locations (e.g., using hashing on keys) to make retrieval more efficient [25, 20]. Using this approach, a node that is searching for data can use this structure to figure out where to look for the data. However, this structure requires extensive communication of data and may not be suitable for sensor networks. Further, related data may be stored at many different locations, making queries inefficient; (2) unstructured: the data is not forced to specific locations [23]. In this case, searching for data is difficult since it may exist anywhere; in the worst case the user must perform a random search. Replication improves the performance of retrieval [3], but is likely to be too expensive for a sensor network environment.

For unstructured networks, P2P solutions differ in terms of the indexing support. Decentralized networks provide no indexing. Centralized networks provide a centralized index structure. Finally, hybrid centralized networks provide hierarchical indexing, where supernodes each keep track of the data present at the nodes managed by them.

While the tradeoffs between these approaches are well studied in the P2P community, it is not clear how they apply for sensor networks. Specifically, P2P networks exist on the internet where resources are not nearly as limited as those in a sensor network. Solutions requiring large data movement or expensive indexing are likely to be inefficient. It is also unclear what keys/attributes should be indexed to facilitate execution of common queries. Finally, solutions that cause expensive query floods will also be inefficient. In the remainder of this section, we study existing research into indexing and retrieval in sensor networks.

Table 2: Comparison of different canonical methods.

Method	Total	Hotspot
External Storage	$D_{total}\sqrt{n}$	D_{total}
Local Storage	$Q_n + D_q\sqrt{n}$	$Q + D_q$
Data-Centric (Summary)	$Q\sqrt{n} + D_{total}\sqrt{n} + Q\sqrt{n}$	$2Q$

6.2 Data Centric Storage – Geographic Hash Tables

In the context of peer-to-peer systems, Distributed Hash Tables (DHTs) are a decentralized structured P2P implementation. Typically these DHTs provide the following simple yet powerful interface: Put(data d , key k) operation, which stores the given data item d based on its key k . Get(key k) operation can be used to retrieve all the data items matching the given key k .

The Geographic Hash Table (GHT) system implements a structured P2P solution for data centric storage (DCS) in sensor networks [21]. Even though GHT provides an equivalent functionality to structured P2P systems, it needs to address several new challenges. Specifically, moving data is costly in sensor networks. Moreover, the authors target keeping related data close such that queries can be more focused and efficient.

In the next few sections we describe how GHT incorporates physical connectivity in data-centric operations in resource-constrained sensor networks. Before delving into the details of GHT based data centric storage, we overview three canonical data dissemination methods along with their approximate communication costs and demonstrate the usefulness of data-centric storage [21].

6.2.1 Canonical Methods

The primary objective of a data dissemination method is to extract relevant data from a sensor network in an efficient manner. The following are three fundamentally different approaches to achieve this objective. Let us assume that the sensor network has n nodes.

1. External Storage (ES): Upon detecting an event, the relevant data is sent to the base station. ES entails $O(\sqrt{n})$ cost for each event to get to the base station, with zero cost for queries generated at the base station (external) and $O(\sqrt{n})$ for queries generated within the sensor network (internal).
2. Local Storage (LS): In this case, a sensor node, upon detecting an event, stores the information locally. LS incurs $O(n)$ cost for query dissemination, since a query must be flooded and $O(\sqrt{n})$ cost to report the event.
3. Data-Centric Storage (DCS): DCS stores named data within the network. It requires $O(\sqrt{n})$ cost to store the event and both querying and event response require $O(\sqrt{n})$ cost.

Let us assume that a sensor network detects T event types, and denote the total number of events detected by D_{total} . Q denotes the number of event types queries and D_Q denotes the number of

events detected for each event queried. Further, assume that there are a total of Q queries (one per event type). Table 6.2.1 shows the approximate communication cost for the three canonical methods. Total cost accounts for the total number of packets sent in the network, whereas the hotspot usage denotes the maximum number of packets sent by any particular sensor node.

From this analysis, it is clear that no single method is preferable under all circumstances. If the events are accessed more frequently than they are generated (by external observers), external storage might be a good alternative. On the other hand, local storage is an attractive option when the events are generated more frequently and accessed infrequently. DCS lies in the middle of these two options and is preferable in cases where the network is large and a large number of events are detected but few are queried. The next sections provide the implementation details and a discussion of GHT.

6.3 GHT: A Geographic Hash Table

GHT is a structured approach to sensor network storage that makes it possible to index data based on content without requiring query flooding. GHT also provides load-balancing of storage usage (assuming fairly uniform sensor deployment). GHT implements a Distributed Hash Table by hashing a key k into geographic coordinates. As mentioned above, GHT supports Put(data d , key k) and Get(key k) operations in the following way. In Put, data (events) are randomly hashed to a geographic location (x, y coordinates), whereas a Get operation on the same key k hashes to the same location. In the case of sensor networks, sensors can be deployed randomly and the geographic hashing function is oblivious to the topology. Therefore it might happen that a sensor node might not exist at the precise location given by the hash function. Also, it is crucial to ensure consistency between Get and Put operations, i.e., both of them map the same key to the same node. Put achieves that by storing the hashed event at the node *nearest* to the hashed location and Get operations can also retrieve an event from the node *nearest* to the hashed location. Of course, this policy ensures the desired consistency between Get and Put operations.

GHT uses Greedy Perimeter Stateless Routing (GPSR) [14]. GPSR is a geographic routing protocol that just uses location information to route packets to any connected destination. It assumes that every node knows its own location and also locations of its all one-hop neighbors. It has two flavors of routing algorithms: greedy forwarding and perimeter forwarding.

In the case of greedy forwarding, a node X upon receiving a packet for a destination D , forwards it to its neighbor that is closest to D among all its neighbors including X itself. Intuitively, a packet moves closer to the destination each time it gets forwarded and eventually it reaches the destination. Of course, greedy forwarding does not work in the case when X does not have any neighbor closer to D than itself. GPSR then switches to its perimeter mode.

In the perimeter mode, GPSR uses the following right-hand rule: upon arriving on an edge at node X , the packet is forwarded on the next edge counterclockwise about X from the ingress degree. GPSR first computes a planar subgraph of the network connectivity graph and then applies the right hand rule to this graph.

6.3.1 Interaction of GHT with GPSR

GPSR provides the capability to deliver packets to a destination node located at the specified coordinates, whereas GHT requires the ability to forward a packet to a node closest to the destination node. GHT defines a *home node* as the node that is closest to the hashed location; GHT requires an ability to store an event at the Home node.

Let us assume that hashing an event gives location (X_1, Y_1) as the coordinates of the node where the given event needs to be stored. Further let us assume that no node exists at location (X_1, Y_1) . Then GHT would store that event at its Home Node H . So node H , upon receiving a packet for destination (X_1, Y_1) , will switch to GPSR's perimeter routing mode (since it will have no neighbor closer to the destination than itself). By applying perimeter routing, the packet will eventually return to H upon traversing the entire perimeter. H can then conclude that it is the Home node for the given event and then will store the event. The interaction between GHT and GPSR described so far works well for static sensors, ideal radios and without considering failures of sensor nodes. However, when thousands of sensors are thrown in a hostile environment, one can hardly assume the presence of these ideal conditions.

GHT ensures robustness (resilience to node failure and topology changes) and scalability (by balancing load across the network) as follows. It uses a novel perimeter refresh protocol to achieve robustness in the face of node failures and topology changes, and it uses structured replication to do load balancing, thereby achieving scalability. We now briefly describe these two techniques.

6.3.2 Perimeter Refresh Protocol

GHT needs to address two challenges, namely home node failure and topology changes (e.g., addition of nodes). Periodically, a home node initiates a perimeter refresh message that traverses across the entire perimeter of the specified location. Every node on the perimeter, upon receiving this event, first stores it locally and then marks the association between the given event and the home node for a certain duration with the help of a timer. If the node receives any subsequent refresh messages from the same home node, it resets the timer and reassociates the home node with the event. However, if it fails to receive the refresh message, it assumes that the home node for the event has failed and initiates a refresh message on its own for electing a new home node. In this manner, GHT recovers from home node failures.

Due to topology changes, it might happen that some new node H_1 is now closer to the destination than the current home node H . When the current home node H initiates the refresh message, H_1 will receive it. Since H_1 is closer to the destination than H (by definition of the home node), it reinitiates the refresh message which then passes through all the nodes on the perimeter including H and eventually returning to H_1 . All the nodes on the perimeter update their associations and timers accordingly to point to H_1 . In this way, GHT addresses the issue of topological changes.

6.3.3 Structured Replication

If many events are hashed onto same node (location) then this node can become a *hot-spot*. Structured replication is GHT's way to balance the load and reduce the hot-spot effect. The basic idea

is to decompose the hierarchy geographically and replicate the home node rather than replicating the data itself. For example, instead of storing data at the root of the hierarchy (home node of the event), a node stores the data at the nearest mirror node (one of the replicas of the home node). For retrieving events, queries are still targeted towards the root node; however, the root node then forwards the query to the mirror node.

6.4 Graph EMbedding for Sensor Networks (GEM)

GHT uses GPSR, a geographic routing protocol that routes packets just based on location information. GEM (Graph EMbedding for sensor networks) is an infrastructure for node-to-node routing and data-centric storage and information processing in sensor networks without using geographic information [18]. GEM embeds a ringed tree into the network topology and labels nodes in a way to construct a VPCS (Virtual Polar Coordinate Space). VPCR is a routing algorithm that runs on top of VPCS and requires no geographic information. It provides the ability to create consistent associations without relying on geographic information.

6.5 Distributed Index for Features in Sensor Networks (DIFS)

GHT targets retrieval of high-level, precisely defined events. The original GHT implementation is limited to report whether a specific high-level event occurred. However, it is not able to efficiently locate data in response to more complex queries. The Distributed Index for Features in Sensor Networks (DIFS) attempts to extend the original architecture to efficiently support range queries. Range queries are the queries where only events within a certain range are desired. In DIFS, the authors propose a distributed index that provides low average search and storage communication requirements while balancing the load across the participating nodes [8]. We now describe the notion of high-level events and discuss a few sample range queries that can be posed by the end user.

6.5.1 High-level Event

A High-level event can be defined in terms of composite measurements of sensor values themselves. For example, a user might query average animal speed, peak temperature in hot regions, etc. Often the user can add timing constraints to these values: for example, a user may query the average animal speed within the last hour. Alternatively, the user might incorporate a spatial dimension as well: for example, a user may look for an area with average temperature greater than a certain threshold. Some end users might be interested in querying relations among various events as well. An example of a query in the temporal domain: Is an increase in temperature above a threshold followed by the detection of an elephant? Of course, all the above types can be used in combination, and more complex queries can be posed. DIFS assumes that all sensors store raw sensor readings, while only a subset of sensors serve as an index node to facilitate the search. Note that DIFS runs on top of GHT to support the above mentioned range queries in an efficient way. Since the reader is already familiar with GHT, we first describe a naive quad tree based approach for indexing followed by the DIFS architecture.

6.5.2 Simple Quad Tree Approach

This approach works by constructing a spatially distributed quad tree of histograms that summarize activity within the area they represent. A root node maintains four histograms describing the distribution of data in each of four equi-sized quadrants (its children). A drill down-query approach can be used on top of this tree; the summaries at children nodes are more detailed. However, in that case, since the root node has to process all the queries (queries move in top-down fashion over the tree), it might prove to be a bottleneck. Also, propagating data up in the hierarchy to update histograms every time a new event gets generated might pose problems from an energy point of view. DIFS extends this naive approach to address these problems.

6.5.3 DIFS architecture

Similar to the quad tree implementation, DIFS constructs an index hierarchy using histograms. Unlike the quad tree approach, every child has b_{fact} number of parents instead of a single parent, where $b_{fact} = 2^i, i \geq 1$. Also, the range of values a child maintains in its histograms is b_{fact} times the range of values maintained by its parent. The crucial point is that, to ensure energy-storage load balancing, a range of values an index node knows about is inversely proportional to the spatial extent it covers. Therefore, rather than having only one query entry point, as in the case of naive quad tree approach, a search may begin at any node in the tree. Query entry points are selected according to both the spatial extent, as well as the range of values mentioned in the query.

Another important feature is *index node selection*. Index node selection is done with the help of a geographically bound hash function. A given sensor field is divided into rectangular quadrants recursively proportional to the number of levels in the index hierarchy. Given a source location, a string to hash, and a bounding box, the output of this hash function is a pair of coordinates within the given bounding box. Note that, in the case of GHT, the hash function can result to any location within the sensor field and not to any location within the bounding box. Intuitively, a node forwards an event to the first local index node with the narrowest spatial coverage but widest value range. This node then forwards it to its parent with wider spatial coverage but narrower value range. This forwarding process can be applied recursively.

7 Concluding Remarks

In this chapter, we considered the problem of storage management for sensor networks where data are stored in the network. We discussed two classes of applications where such storage is needed: (1) Off-line scientific monitoring: data is collected off-line and periodically gathered by an observer for later playback and analysis; and (2) Augmented reality applications: data is stored in the network and used to answer dynamically generated queries from multiple observers. In such storage-bound networks, the sensors are limited not only in terms of available energy, but also in terms of storage. Furthermore, efficient data-centric indexing and retrieval of the data is desirable, especially for the second application type.

We have identified the goals, challenges and design considerations present in storage-bound sensor networks. We organized the challenges into three areas, each incorporating a unique de-

sign/tradeoff space with respect to the identified goals. Moreover, each of the areas has preliminary existing research and protocols. These areas are:

1. System support for storage: storage for sensors is different from traditional non-volatile storage both in terms of the hardware, applications and resource limitations. As such, the hardware and required file system support differ significantly from traditional systems.
2. Collaborative storage: storage can be minimized by exploiting spatial correlation between nearby sensors and by reducing redundancy/adjusting sampling periods. Such techniques require data exchange among nearby sensors (and the associated energy costs). An intricate tradeoff in terms of energy exists: local buffering may save energy during the data collection phase but not during indexing and data retrieval. Furthermore, storage and energy may need to be balanced depending on the resource that is scarcer.
3. Indexing and retrieval: the final problem is similar to the indexing and retrieval problem in P2P networks. However, it differs in the type of data and the high costs for communication (requiring optimized queries and limiting of unnecessary data motion). We discussed existing solutions in this area.

Most existing work addresses these different aspects of the problem independent of the others. We believe that there remains large areas of the design space in each area that are unexplored. However, issues at the intersection of these areas appear most challenging and have not been addressed to our knowledge.

References

- [1] S. Androutsellis-Theotokis. A survey of peer-to-peer file sharing technologies, 2002. Available on the Web: http://www.eltrun.aueb.gr/whitepapers/p2p_2002.pdf.
- [2] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. *Lecture Notes in Computer Science*, 1987, 2001.
- [3] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. 2002. in The ACM SIGCOMM'02 Conference, August 2002.
- [4] 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.
- [5] D. Ganesan, S. Ratnasamy, H. Wang, and D. Estrin. Coping with irregular spatio-temporal sampling in sensor networks. *ACM Computer Communication Review (CCR)*, 2004. (forthcoming).
- [6] D. Gay. Design of matchbox, the simple filing system for motes, 2003. Available on the Web: <http://www.tinyos.net/tinyos-1.x/doc/matchbox.pdf>.

- [7] D. Gay. Matchbox: A simple filing system for motes, 2003. Available on the Web: www.tinyos.net/tinyos-1.x/doc/matchbox-design.pdf.
- [8] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker. Difs: A distributed index for features in sensor networks. In *First IEEE International Workshop on Sensor Network Protocols and Applications (SNPA 2003)*, 2003.
- [9] W. Heinzelman. *Application-Specific Protocol Architectures for Wireless Networks*. PhD thesis, Massachusetts Institute of Technology, 2000.
- [10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.
- [11] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. Technical Report TR-01-750, Univ. of Southern California, Nov. 2001.
- [12] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. 6th ACM International Conference on Mobile Computing and Networking (Mobicom'00)*, Aug. 2000.
- [13] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In *Proceedings of the 10th international conference on architectural support for programming languages and operating systems*, pages 96–107. ACM Press, 2002.
- [14] B. Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254. ACM Press, 2000.
- [15] S. Madden. *The Design and Evaluation of a Query Processing Architecture for Sensor Networks*. PhD thesis, U.C. Berkeley, 2003.
- [16] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *OSDI*, 2002.
- [17] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for UNIX. *Computer Systems*, 2(3):181–197, 1984.
- [18] 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 ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, 2003.
- [19] Network Simulator. <http://isi.edu/nsnam/ns>.

- [20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, 2001.
- [21] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-centric storage in sensornets with ght, a geographic hash table. *Mob. Netw. Appl.*, 8(4):427–442, 2003.
- [22] A remote ecological micro-sensor network, 2000. (Available on the web at: <http://www.botany.hawaii.edu/pods/overview.htm>).
- [23] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 6(1).
- [24] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, 1992.
- [25] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.
- [26] S. Tilak, N. Abu-Ghazaleh, and W. Heinzelman. Storage management issues for sensor networks. Available on the Web: www.cs.binghamton.edu/~sameer/CS-TR-04-NA01.
- [27] Crossbow Technology Inc. <http://www.xbow.com>.