Multi-user Data Sharing in Radar Sensor Networks

M. Li, T. Yan, D. Ganesan, E. Lyons, P. Shenoy, A. Venkataramani, and M. Zink
Department of Computer Science,
University of Massachusetts Amherst, MA 01003.
{mingli,yan,dganesan,elyons,shenoy,arun,zink}@cs.umass.edu

ABSTRACT

In this paper, we focus on a network of rich sensors that are geographically distributed and argue that the design of such networks poses very different challenges from traditional "mote-class" sensor network design. We identify the need to handle the diverse requirements of multiple users to be a major design challenge, and propose a utility-driven approach to maximize data sharing across users while judiciously using limited network and computational resources. Our utility-driven architecture addresses three key challenges for such rich multi-user sensor networks: how to define utility functions for networks with data sharing among end-users, how to compress and prioritize data transmissions according to its importance to end-users, and how to gracefully degrade end-user utility in the presence of bandwidth fluctuations. We instantiate this architecture in the context of geographically distributed wireless radar sensor networks for weather, and present results from an implementation of our system on a multi-hop wireless mesh network that uses real radar data with real end-user applications. Our results demonstrate that our progressive compression and transmission approach achieves an order of magnitude improvement in application utility over existing utility-agnostic non-progressive approaches, while also scaling better with the number of nodes in the network.

1. INTRODUCTION

While much of the focus of the sensor network community has been on the design of miniature low-power "mote-class" wireless sensor networks, there is an equally important ongoing networking revolution for "rich" powerful higher-power sensors. This revolution has been driven by two technology trends. The first trend is the emergence of cheaper, more efficient and more compact designs of traditionally large and unwieldy sensors such as radars and cameras [7], enabling more mobile, solar-powered deployments in remote locations that lack sensing coverage. The second trend is the recent success in designing WiFi-based long-range, multi-hop mesh networks [1, 5], which facilitate ad-hoc remote deployments of these sensors in areas where a wired network infrastructure is unavailable. These technological developments have led to several efforts to deploy large-scale, dense, wirelessly connected networks of powerful sensors, including earthquake sensing ([8]), weather monitoring using wireless radars ([18]), and road traffic monitoring ([3]).

These emerging large-scale sensor systems (shown in Figure 1) have important differences from their existing resource-poor counterparts and raise a number of new research challenges. The first major difference between the two types of sensor networks is their design objective. Due to limited energy resources and the need for long lifetime, the design performance goal in mote-class sensor networks is to minimize energy consumption. Other resources such as bandwidth and computation are typically less of a concern since simple, low data-rate sensors such as those for temperature, humidity, or pressure are used. In contrast, rich sensors such as radars and cameras generate raw data at hundreds of kilobits or tens of megabits per second. However, the per-node bandwidth on a shared wireless mesh is limited. Consequently, the need to optimize network bandwidth usage is as important as minimizing energy consumption in such networks.

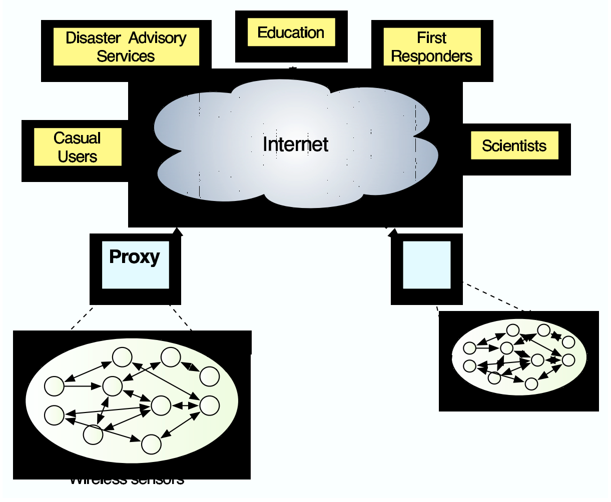


Figure 1: **Multi-user Sensor Networks**

A second key difference is the diversity of end-users that the two types of sensor networks are designed to support. Mote-class sensor networks typically have many tens of nodes deployed in a small geographic area, and are designed to perform one or few tasks efficiently, primarily periodic data collection. This is both due to the lack of available resources on the sensors to perform computationally intensive in-network data processing, and due to the limited geographic area that the sensors span. In contrast, many rich sensor systems span vast geographies, and are intended to serve a spectrum of users with different needs. To illustrate, users of a large-scale transportation sensor network using cameras will include traffic police, first responders who need notification about accidents, commuters who are interested in traffic congestion on their routes, and even insurance companies who might desire information about accidents to settle claims. These different types of users often impose different, and sometimes conflicting, demands on the network. In a radar sensor network, scientists may desire access to raw data to conduct research, while meteorological applications may require data that has undergone intermediate processing, and end-users may only need the "final processed result". Further, a tornado detection application will require timely notifications of important events, while other users are less sensitive to delay in sensor updates (e.g., end-users are tolerant to slight delays in weather updates).

Thus, a key challenge in rich sensor networks is to optimize diverse user needs in the presence of limited resources. One option is to handle the different user needs separately, but this model ignores one of the most important characteristics of multi-user sensor networks — all the users of a sensor network operate on the *same* data streams and the data relevant to one user can potentially be used to handle the needs of other users. Thus, rather than separately handling user needs, an approach that jointly considers user needs to maximize data sharing among users is better suited to make judicious use of the limited computational and network resources. Since the workload seen by such networks can dynamically vary over time as user needs and interests change—for instance, the workload imposed by users can increase significantly during an intense storm or a major traffic problem—such data sharing techniques must also adapt to dynamic load conditions.

1.1 Research Contributions

In this paper, we describe a novel utility-driven architecture that maximizes data sharing among diverse users in a sensor network. We believe that maximizing utility across diverse end-user queries using *multi-user data sharing* techniques (henceforth referred to as MUDS[1]) is a key challenge for designing more scalable sensor networks. Our architecture is designed for hierarchical sensor networks where sensors are streaming data over a multi-hop wireless network to a sensor proxy. These incoming data streams at the proxy are used to answer queries from different users. The proxy and the sensors interact continually to maximize data sharing across queries while simultaneously adapting to bandwidth variations, and changing query needs of users. We instantiate this architecture in the context of ad-hoc networks of wireless radar sensors for severe weather prediction and monitoring. Our work has three main contributions:

- **Multi-query Aggregation:** A key contribution of our work is multi-query aggregation, where radar data streams are shared between multiple and diverse end-user queries, thereby maximizing total end-user utility. We demonstrate that different end-user application needs, spatial areas of interest, deadlines, and priorities, can be combined into a single aggregated query, thereby enabling more optimized use of bandwidth resources.
- **Utility-driven Compression and Scheduling:** At the core of our system is a utility-driven progressive data compression and packet scheduling engine at each radar. The progressive compression engine enables radar data to be compressed and ordered such that information of most interest to queries is transmitted first. Such an encoding enables our system to adapt gracefully to bandwidth fluctuations.The utility-driven scheduler compares the utility of different progressively compressed streams that are intended for different sets of queries, and transmits packets such that utility across all concurrent queries at a radar is maximized.
- **Global Transmission Control:** In addition to local utility-driven techniques, our system supports global utility optimization mechanisms driven by the proxy. The proxy continually monitors the utility of incoming data from different radars and decides how to control streams to maximize total utility across the entire network. Such a global control mechanism enables the system to adapt to uneven query distribution across the network, and to deal with disparities in available bandwidth among different radars due to wireless contention. This is especially important when some nodes in the network are observing important events such as tornadoes, and need to obtain more bandwidth than other nodes that are transmitting data for less critical queries.

In our experiments, we measure, evaluate and demonstrate the performance of our architecture and algorithms for radar sensor networks for severe weather monitoring. We have implemented the system on a testbed of Linux machines that form an 802.11-based wireless mesh network. Using a combination of simulations and experiments with real and emulated radar traces, we show that our system provides more than an order of magnitude (11x) improvement in query accuracy and utility for a 12 node network, when compared to an existing utility-agnostic non-progressive approach. Our system also degrades gracefully with network size — when the network size increases from three nodes to twelve nodes, the average utility achieved by each radar in our system only decreases by 15%, whereas the average utility of the existing *XRad* approach decreases by 57%. Further, our system adapts better to bandwidth variations with only 15% reduction in utility when the bandwidth drops from 150kbps to 10kbps.

The rest of this paper is structured as follows. Section 2 provides an overview of radar sensor networks and the challenges in these networks. Section 3 provides an overview of our architecture, while Section 4 describes the design of the key components of our architecture. Sections 5 describes our implementation and evaluation. Finally, Sections 6 and 7 discuss related work and our conclusions.

[1]Actual system name is anonymized to aid double-blind reviewing. We use the term MUDS to refer to our system and XRad to refer to existing radar networks.

2. RADAR SENSOR NETWORKS

In this section, we provide an overview of the diverse end-user applications that use a radar sensor network, followed by the formulation of the problem addressed in this paper.

2.1 End User Applications

A network of weather sensing radar sensors can be used by diverse users such as automated weather monitoring applications, meteorologists, scientists, teachers and emergency personnel. Several different weather monitoring applications may be in use, each of which continuously requests and processes data sensed by various radars:

- *Hazardous weather detection:* Applications in this class are responsible for detecting hazardous weather such as storm cells, tornadoes, hail, and severe winds in real-time (e.g. [10]). This class of applications focuses on sharp changes in weather patterns; a tornado detection application, for instance, looks for sharp changes in wind speed and direction that are indicative of a tornado.
- *3D wind direction estimation:* This application constructs a 3D map by computing the direction of the wind at each point in 3D space. Since a single radar can only determine wind direction in a single dimension (radial axis), the application needs to merge data from two or more overlapping radars in order to estimate the 3D wind direction. Due to the need to merge data, only regions of overlap between adjacent radars are useful, and data from other areas need not be transmitted.
- *3D assimilation:* This application integrates data from multiple radars into a single 3D view to depict areas of high reflectivity (intense rain) that occur in the region.

We note that the first application is of interest to meteorologists for real-time weather forecasting, the second is useful to researchers, while the third is useful to emergency managers to visualize weather in their jurisdiction. In addition to these applications, end-users may pose other ad-hoc queries for data or instantiate continual queries that continuously request and process data to detect certain events or conditions.

2.2 System Model and Problem Formulation

Our MUDS radar sensing network comprises three tiers as shown in Figure 2 (i) applications and end-users who pose queries and request field data, (ii) sensor proxies that act as the gateway between the Internet and the radar sensor field, execute user queries, and manage the radar sensor network, and (iii) a wireless network of remote radar sensors that implement utility-driven services and stream their data to the proxy.

Each radar node comprises a mechanically steerable radar attached to an embedded PC controller; the embedded PC with dual-core Intel processor that runs Linux is equipped with 1GB RAM and a 802.11 wireless interface. A typical deployment will comprise many tens of radars distributed over a wide geographic area. The radars are "small" and are designed to be deployed in areas with no infrastructure using solar-powered rechargeable batteries; they can also be deployed on cellphone towers or on building rooftops where infrastructure such as A/C power is readily available. In either case, we assume that the radars connect to the proxy node using a multi-hop 802.11 wireless mesh network.

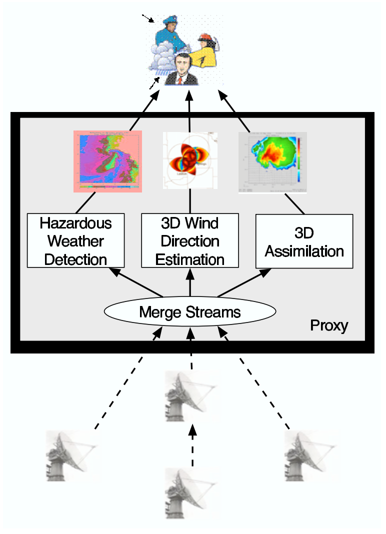


Figure 2: **Multi-hop Radar Sensor Networks**

Each mechanically steerable radar has two degrees of freedom (θ, ϕ) which enable control over the *orientation* and the *altitude* where the radar points and senses data. The radar scans the atmosphere by first positioning itself to point at an altitude ϕ and then conducts a *scan* by rotating θ degrees and scanning while rotating. The MUDS system operates in rounds, where each round is referred to as an *epoch*. For this paper, we assume an epoch of 30 seconds. Before each epoch begins, the proxy collects all queries for a particular radar. Each query represents a request for data from a weather monitoring application (e.g., the tornado detection) or from end-users (who may issue ad-hoc queries). A query can request *any subset* of the region covered by a radar scan—for instance, a tornado detection algorithm may only request data from regions where intense weather has been detected. Each query has a priority and a deadline associated with it, which is then used to to assign a *weight* to each region that the radar can scan. The weight represents the relative importance of scanning and transmitting data from the region in the next epoch.[2] Thus, region-specific weights represent collective needs of all queries that have requested data in that epoch.

Assuming the weights are computed before each epoch begins, the radar then scans all regions with non-zero weights during the epoch. Each scan is assumed to produce tens of Megabytes of raw data, which is typically much higher than bandwidth available to each radar in a multi-hop 802.11 mesh network. Thus, the primary constraint at a radar node is bandwidth, and the radar node must determine how to intelligently transmit the results of the scan back to the proxy.

[2]For instance, a region that is not requested by any query will receive a weight of zero and need not be scanned by the radar.

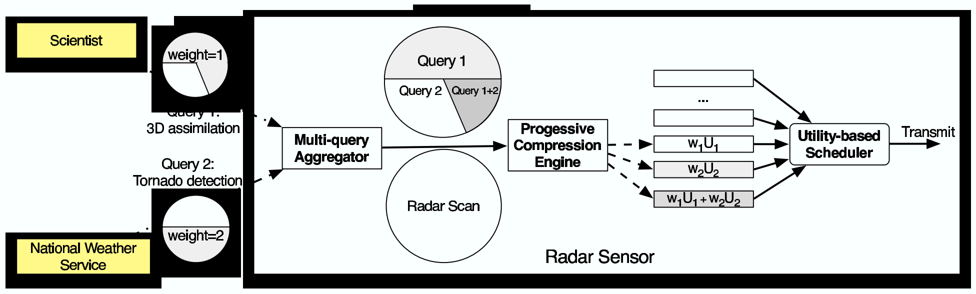


Figure 3: **Multiple incoming queries in an epoch are first aggregated by the multi-query aggregator at the radar. The merged query and the radar scan for the epoch are input to the progressive encoder which generates different compressed streams for different regions in the query. The streams are input to the utility-driven scheduler which schedules packets across all streams whose deadlines have not yet expired.**

Each proxy is assumed to be a server (or a server cluster) with significant processing and memory resources. The weather monitoring applications described in Section 2.1 are assumed to execute at the proxy, processing data streams from various radars in real-time. Each application is assumed to process data from an epoch and issues per-radar queries for data that it needs in the next epoch.

Assuming such a system, this paper addresses the following questions:

- How can the radar sensor system merge and jointly handle queries with diverse high-level needs such as tornado detection, 3D wind direction estimation and 3D assimilation?
- Since the raw data from a scan exceeds the available network bandwidth and this bandwidth can vary significantly over time, how should a radar node intelligently compress the raw data prior to transmission?
- How should the radar prioritize the transmission of this compressed scan result back to the proxy node so that application overall utility is maximized?
- Since the query load on different radars can be uneven and data from some radars may be more critical than others during intense storms, how should the proxy globally control transmissions across radars to ensure that important data gets priority?

The following section discusses techniques employed by the MUDS system to address these questions. For simplicity of exposition and because optimizing the radar scan strategy is not the goal of our work, we assume each radar points at a fixed altitude ϕ and performs a 360^o scan of the atmosphere resulting in a full 2D scan. It is straightforward to extend the discussion to three dimensional partial scans where both the altitude ϕ and the rotation θ are varied in a scan. Also, since our focus is on multi-user data sharing in a wireless environment, we do not focus on the design issues of long range wireless mesh networks, and assume that existing techniques such as [5, 17] can be used.

3. MUDS SYSTEM ARCHITECTURE

The proxy and sensor in the MUDS system interact continually to maximize utility under query and bandwidth dynamics. This interaction has four major parts: (a) a multi-query aggregation phase at the proxy and radar to compute a single unified query per epoch, (b) progressive compression of the radar scan at each radar by using the unified query as input, (c) a utility-driven scheduling phase at each radar where packets are prioritized by overall utility gain, and (d) a global transmission control phase driven by the proxy to optimize transmissions from different radars.

Multi-query aggregation: The first phase of our system operation is the multi-query aggregation phase where multiple user queries in an epoch are combined to generate a single unified query. This is done both by the proxy as well as the radars — the proxy uses the unified query for global transmission control, and the radar uses it for progressive compression and scheduling. Each user query is associated with a weight, a spatial region of interest, and a deadline. The weight of a query is dependent on the priority of the user (e.g. the National Weather Service is a high priority user), and the priority of the query to the user (e.g. a tornado detection query has higher priority during times of severe weather). Each query is also associated with a spatial area of interest, for instance, the wind direction estimation query is only meaningful for overlapping regions between radars. Queries are executed in batches — queries that arrive within a single epoch are merged to generate a joint spatial query map that captures the needs of all concurrent queries. An example of the spatial map that merges a tornado detection and a 3D assimilation query is shown in Figure 3. The merging of queries results in their weights being accumulated for shared regions of interest. The set of queries in an epoch is communicated by the proxy to the individual radar sensors whenever there is a change due to the arrival of new queries.

Progressive compression: Each radar scan produces tens of Megabytes of raw data that must then be transmitted back to the proxy node. Since the raw data rate is significantly higher than the bandwidth available per radar on the mesh, the data rate must somehow be reduced prior to transmission. The existing *XRad* system employs a simple averaging technique to down-sample data—neighboring readings are averaged and replaced by this mean; the larger the number of neighboring readings over which the mean is computed, the greater the reduction in data rate. Rather than using a naive averaging technique, our system relies on the query map to intelligently reduce the data rate using a *progressive compression* technique. The progressive compression engine uses the unified query map and compresses data in two steps. First, the weights of different regions in the map are used to split the radar scan into multiple smaller regions, such that each region has a fixed weight and a fixed set of associated queries. Thus, the radar scan in Figure 3 is split into three regions with weights 1, 2, and 3 respectively. Each of these regions is then progressively encoded using a wavelet-based progressive encoder. The encoder compresses and orders data in each region such that most important features in the data is transmitted first, and less important features are transmitted later. Finally, the progressively encoded streams corresponding to different regions are input to a utility-based scheduler at the radar.

Utility-driven packet scheduling: The utility-based scheduler schedules packets between different streams from different epochs, and makes a decision regarding which packet to send from among the streams. This decision is based on the weight associated with the stream and the utility of the packet to the queries that are interested in the stream. For example, stream 3 in Figure 3 is of interest to both queries; therefore transmitting a packet improves the utility for both the queries. In order to compute the utility of a packet, the radar uses *a priori* knowledge of how application utility relates to the mean square error (MSE) of the data. This provides a mechanism for the scheduler to observe *error in the compressed raw data* and determine how this error would translate to *application error*. As we describe later, the mean square error of the data influences utility in different ways for different applications. The scheduler computes the total benefit (computed as the product of marginal utility of the packet and weight assigned) that would result from transmitting the first packet from each stream, and picks the stream with greatest increase in benefit. Figure 4 provides an illustration of the scheduling decision. In the example, 66% of the first stream has been transmitted but only 33% of the second stream has been transmitted. Therefore, the difference in mean square error is likely to be higher by transmitting a packet from the second stream. However, there are two additional factors to consider. The first stream corresponds to a tornado detection query, which requires high resolution data in order to precisely pinpoint the location of the tornado, whereas the second stream corresponds to a 3D assimilation query and 3D wind direction estimation queries, each of which needs only less precise data. On the other hand, a packet from the second stream is useful to two concurrent queries, whereas a packet from the first stream is only useful for tornado detection. Thus, the decision of what packet to choose depends on the mean square error of the data, number of queries interested in the data, weights of the queries, and importantly, the utility function of the queries.

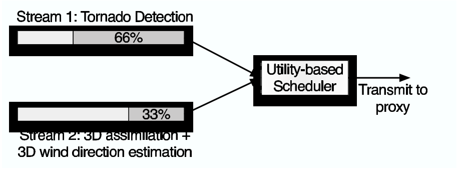

Stream 1: Tornado Detection
66%
Utility-based Scheduler
Transmit to proxy
33%
Stream 2: 3D assimilation + 3D wind direction estimation

Figure 4: **In this scenario, 66% of stream 1 and 33% of stream 2 have been transmitted. The scheduler determines the marginal utility of transmitting a packet from each of the streams for the applications interested in the streams and decides which packet to transmit next.**

Global Transmission Control: While the progressive encoding and utility-driven scheduling at each sensor optimize for multiple queries at a single radar, there is a need for global control of transmissions to maximize overall utility across the network. In particular, this is useful when queries are not evenly distributed across the network, and some nodes that are handling higher priority queries need more bandwidth than others. The proxy uses a simple global transmission control policy where it monitors the utility of incoming packets from different radars. If there is a great imbalance in the utility of streams from different radars, it notifies the radar with lower utility to stop its stream temporarily. This has the effect of reducing contention in the network, especially at nodes close to the proxy, thereby potentially enabling a radar with more important data to obtain more bandwidth to the proxy.

4. MUDS SYSTEM DESIGN

We describe each component of the MUDS architecture in greater detail in this section.

4.1 Multi-Query Aggregator

The multi-query aggregator is central to the data sharing goals of our system. Aggregating multiple user queries into a single aggregated query has two benefits. First, it minimizes the number of scans performed by the radar (which is time and energy-intensive) since each radar scan is used to answer a batch of queries. Second, it allows the data in a single scan to be transmitted once but shared to answer multiple queries, thereby maximizing query utility in limited bandwidth settings. In contrast, a system that scans and transmits data separately for each query would be extremely inefficient both due to increased scanning overhead, as well as the duplication of data transmitted.

The proxy batches all queries that are posed in each epoch, and at the beginning of the next epoch, it sends to each radar a list of queries that require data from that radar. An alternative model could have been for the proxy to merge the queries and transmit only the merged query to the radar sensor, but we eschewed this option since it would consume more bandwidth than just sending the queries to the radar. Each query is specified by a 4-tuple <*QueryType, ROI, Priority, Deadline*> that shows the type, region of interest, priority, and the deadline of the query. In our system, the region of interest is represented by a sector or a rectangle for simplicity, although our approach can be easily extended to handle more arbitrary regions of interest. The priority can be either specified by the query or implicitly specified by the proxy — for instance, if the user is a high priority user like the National Weather Service — or can be determined as a combination of the two.

The multi-query aggregator then combines multiple user queries into a single aggregated query plan. The query plan that is generated is a spatial map in which the spatial area corresponding to the region covered by the radar is pixelated. For each pixel in the scan data, the corresponding pixel in the query plan is a list of 3-tuples <*QueryType, Weight, Deadline*>, that show the type, weight, and the deadline of queries interested in data sensed at that pixel.

The weight value of a pixel for each query represents the "importance" of transmitting data sensed from that pixel to that query. We use a heuristic for determining pixel weights in order to maximize application utility. Let p_i, I_i , and d_i represent the priority, the region of interest, and the deadline of query i. Priority p_i is represented as a scalar value; region of interest, I_i, is represented as a 2D map where $I_i(u, v)$ is 1 if the pixel (u, v) is within the region of interest of i, and 0 otherwise; and deadline, d_i is in seconds.

Let $w_i(u, v)$ represent the weight of pixel (u, v) for query i. We would like the following three criteria to be satisfied: i) the weight for the pixel should be greater if the query has higher priority than other queries, ii) the weight for the pixel should be greater if the query's deadline is shorter than other queries since higher weight will result in the data being transmitted first, and iii) the weight for the pixel should be zero if the pixel is not in the region of interest of query i. Thus, the weight $w_i(u, v)$ is defined as:

$$w_i(u, v) = p_i I_i(u, v) \frac{1}{d_i} \tag{1}$$

4.2 Progressive Compression Engine

Data compression is an integral component of rich sensor networks where the data rates can be considerably higher than available bandwidth. In our system, we use progressive encoding to compress raw data. Progressive compression of data yields two benefits: (a) it enables the system to use all available wireless bandwidth to transmit data, thereby adapting to bandwidth fluctuations, and (b) it enables us to order data packets based on utility of data to queries, thereby maximizing overall utility.

Progressive encoding (also known as embedded encoding) compresses data into a bit stream with increasing accuracy. This means that as more bits are added to the stream, the decoded data will contain more detail. In our system, we use a wavelet-based progressive encoding algorithm called *set partitioning in hierarchical trees (SPIHT)* [20]. The choice of a wavelet encoder is well-suited for radar data processing applications since meteorological tornado detection algorithms use wavelet-based processing in order to detect discontinuities in reflectivity and velocity signals [6, 14]. Moreover, SPIHT orders the bits in the steam such that the most important data is transmitted first. Thus, the decoded data can achieve high fidelity even with few packets transmitted.

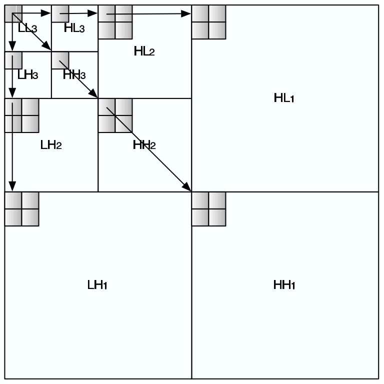


Figure 5: **An example subband pyramid generated by wavelet transform. The arrows represent a spatial orientation tree that is constructed on the pyramid. Each arrow is oriented from the parent node to its descendants.**

We provide a brief overview of the SPIHT algorithm next (refer [20] for a detailed discussion). The input data for the algorithm is assumed to be a two-dimension matrix $D(i, j)$. Before SPIHT encoding, the data D is first transformed into a subband pyramid $P(i, j)$ using the wavelet transform. In the transform, the data D is split into subbands recursively. After each iteration, four new subbands labeled as *LL, LH, HL, HH* are generated as shown in Figure 5. The first letter *L/H* in the label refers to the low/high frequency component in the horizontal dimension of the matrix, while the second *L/H* means low/high frequency component in the vertical dimension. The next step is generating a hierarchical tree from the subband pyramid as shown by arrows in Figure 5. Each arrow is oriented from the parent node to its four descendants that cover the parent node's corresponding area in higher frequency, except that each root node only has three descendants. The data encoding iterates through the hierarchical tree starting from the root node. In each iteration, the most significant bit of each point is output into a stream and is removed from that point. In the generated stream, the most important data is at the head of the stream because most natural images like photos or radar scans have energy concentrated in the low frequency components so the significance of a point decreases as we move from the highest to the lowest levels of the tree.

Besides generating the progressive stream, the SPIHT encoder also generates an incremental trace of the encoded stream that shows what the mean square error of the decoded data would be after sending each byte of the stream. As described in the next section, this feature is essential to perform utility-driven scheduling of packets.

We made a few modifications to the standard SPIHT encoder to adapt it to our needs. The progressive encoding engine in our system first splits each scan into multiple regions such that all pixels in a region share the same list of three tuples, <*QueryType, Weight, Deadline*> in the aggregated query map. Although this may result in an exponential number of regions with respect to the number of queries in the worst case, in practice we find the number of regions to be small for radar queries. Each of these regions is encoded to generate a progressively compressed stream per region. One practical problem is that the standard wavelet transform that expects a square matrix, but each region can be of arbitrary shape. To deal with this, we use a shape adaptive wavelet transform that can handle arbitrary shapes to encode each region. The generated streams are buffered and fed into the local transmission scheduler.

4.3 Local Transmission Scheduler

At any given time, a radar may have multiple streams that are buffered and being transmitted by the local transmission scheduler. The goal of this scheduler is to optimize the transmission order of the data in the streams in order to maximize overall application utility despite fluctuating bandwidth conditions. We describe this in detail next.

Each stream buffered by the scheduler comprises packets of the same length (1KB in our implementation). The local transmission scheduler optimizes the transmission order of the packets based on their marginal utility to the set of queries corresponding to the stream. The marginal utility of a packet is the increase in utility resulting from the transmission of that packet. Informally, the utility of a prefix of a stream is determined by the application error that results from decoding and processing that prefix.

Formally, let p denote some prefix of a stream and let i denote a query corresponding to that stream. The utility $U_i(p)$ of p to query i is given by

$$U_i(p) = \begin{cases} w_i & \text{if } \mathit{err}_i(p) < \mathit{req\_err}_i(p) \\ w_i \frac{\mathit{max\_err}_i(p) - \mathit{err}_i(p)}{\mathit{max\_err}_i(p) - \mathit{req\_err}_i(p)} & \text{if } \mathit{err}_i(p) \geq \mathit{req\_err}_i(p) \end{cases} \tag{2}$$

where w_i is the weight of the query i; $\mathit{err}_i(p)$ is the application error that results from decoding and processing p; $\mathit{max\_err}_i(p)$ is the maximum value of the application error (computed as the error corresponding to a 1KB prefix of the stream); and $\mathit{req\_err}_i(p)$ is the error value below which the user is satisfied with the result. Thus, the utility decreases linearly with the application error and stops decreasing when the user-specified limit is reached. The marginal utility of a packet to a query is the difference in utility to the query just before and after sending the packet.

How does the scheduler compute the application error $\mathit{err}_i(p)$? It is impractical for the scheduler to measure $\mathit{err}_i(p)$ by running the application on each prefix of the stream because of the huge computation overhead of decompressing data and executing the application. Thus, we need a simple and accurate method to determine $\mathit{err}_i(p)$ given just the compressed stream. One possibility is to use a data-agnostic metric such as the compression ratio as an indicator of application error. However, since the progressive encoder could be encoding different scans with very different features, this metric is only weakly correlated with application error.

Fortunately, our empirical evaluation confirms that a data-centric metric, the mean square error of the data stream, is highly correlated to the application error. We leverage this observation to estimate application error as follows. We seed the scheduler with a function $\mathit{seed\_err}_i(\mathit{mse})$ that maps mean square error of the decoded data to application error. Such a function is generated a priori for each application using training data from past radar scans. In the training procedure, scans are compressed into a progressive stream using the SPIHT compression algorithm. The stream is cut off at different prefix lengths, giving us decoded data of varying fidelity. For each such prefix, the application is run on the decoded data, and the error of the decoded data as well as the application error are measured. Based on this measured data, we build a function $\mathit{seed\_err}_i(\mathit{mse})$ for each application and seed each radar with this function.

Finally, during regular operation, the scheduler needs to compute *mse* corresponding to the decoded prefix just after sending the packet. The *mse* can be obtained from the error trace generated by the progressive compressor as described in Section 4.2. The scheduler estimates $\mathit{err}_i(p)$ as $\mathit{seed\_err}_i(\mathit{mse})$ by simply performing a lookup table. The weight of the query w_i is incorporated in Equation 2 so that more urgent queries have higher utility. Note that by construction, all pixels in a region have the same weight.

The total marginal utility of a packet x is its marginal utility across all queries corresponding to the stream. To understand this, suppose there are m queries corresponding to a stream. Let $U_i(p)$ be the utility of prefix p to query i just before sending packet x, and $U_i(p+x)$ just after. Then, the overall marginal utility of packet is given by

$$\Delta U(p) = \sum_{i=1\cdots m} (U_i(p+x) - U_i(p)), \tag{3}$$

where the operator '$+$' denotes extending the prefix to include the next packet. Based on Equation 3 the scheduler can calculate the marginal utility of the packet at the head of each stream. Given the utility, the scheduler picks in each round the packet with maximum marginal utility across all packets at the heads of existing streams, and transmits that packet. Such a scheduling algorithm can be implemented efficiently in practice. First, we note that packets within a stream are already present in order of decreasing marginal utility, so only the packet at the head of each stream needs to be examined for a scheduling decision. The marginal utility of the packet at the head of each stream can be computed efficiently with a small number of table lookups — one lookup to identify the MSE difference resulting from transmitting the packet, and one lookup per query to identify the marginal utility for the query from decoding the packet. Finally, the packet with the highest marginal utility across all streams needs to be chosen. Since the number of streams is small, our implementation simply uses a linear insert and search procedure; it is straightforward to use a heap instead.

Theorem 1. *The above packet scheduling algorithm achieves the maximum total utility across all the concurrent streams at each point in time if $U(p)$ is concave, i.e., the marginal utility is strictly decreasing.*

The proof follows from a straightforward reduction to the knapsack problem and a standard result from convex optimization (omitted for space). Our empirical evaluation confirms that the marginal utility decreases with the length of the progressively encoded stream.

Example: We exemplify our methodology for computing the *seed_err*() function for the three applications. We first consider tornado detection. This application uses a clustering-based technique to detect tornadoes, and generates the centroids and intensities of each tornado. In order to determine the error in tornado detection, we run the application on scans that were decoded after compressing them to different compression ratios. Let the data MSE for a decoded scan be mse_1. There are three cases to consider to determine $seed\_err(mse_1)$. First, if the result on the decompressed scan detects a tornado, t, within 300m of the result on the raw scan, then this is a positive result. The choice of 300m as the threshold for positive detection was made based on discussions with meteorologists. In this case, tornado detection error $tornado\_err(t)$ is computed as follows: $tornado\_err(t) = |(RI(t) - DI(t))| \cdot \frac{d(t)}{300}$ where $RI(t)$ is the intensity of the tornado as determined from processing the raw data, $DI(t)$ is the intensity from processing the decoded data, and $d(t)$ is the distance between the actual centroid from the raw data, and the computed centroid from the decoded data. Second, if a tornado, t, is detected in the decoded scan but no tornado is detected in the raw scan within 300m, then it is considered a false positive. In this case, $tornado\_err(t) = DI(t)$. Finally, if a tornado, t, is detected in the raw scan but no tornado is detected within 300m of its centroid in the decoded scan, then this is considered a false negative, and $tornado\_err(t) = RI(t)$.

The total error, $seed\_err(mse_1)$ is the sum over of the above errors over all tornadoes detected in the raw scan and the compressed scan. Determining the error function for the 3D wind direction estimation and 3D assimilation applications is more straightforward. Here, the applications are run on the raw radar scan and the decompressed scan, and the mean square error of the difference between these results is used as the error for the application.

4.4 Global Transmission Control

While the local transmission scheduler uses the weight map to optimize what order to transmit packets from each radar, the global transmission controller performs a decision across all the concurrent streams on all the radars. Radars compete with each other for wireless bandwidth in a number of ways: (i) radars within the same wireless contention domain contend with each other when transmitting, (ii) in multi-hop communication, all the nodes in the same routing branch share the bandwidth of a forwarding node, and (iii) the proxy's incoming bandwidth is shared among all the radars in the network. As a result, maximizing local utility at each radar may not optimize global utility across all radars in the network. A radar with higher utility data might have much lower available bandwidth than a radar with lower utility due to a number of factors.

This necessitates global control of transmissions from radars, in addition to local utility optimization. Global transmission control in wireless networks has been the subject of significant work (e.g. [12]). Most of these approaches use the idea of a conflict graph that captures the interference patterns between nodes in the network. Such a conflict graph can be used as the foundation for scheduling transmissions from nodes such that spatial reuse is maximized, in addition to throughput.

While the use of conflict graphs is the subject of our future research in the area, we use a simple but effective heuristic in this work. In our approach, the proxy monitors the incoming streams from the radars, and stops the transmission of streams that will not improve overall utility much. Specifically, the proxy stops a stream when its utility reaches 95% of its maximal utility. The proxy knows the maximum utility since it has a locally generated version of the aggregated query plan. Since utility is a concave function of the length of the transmitted data stream, the utility of a stream grows very slowly after having achieved 95% of its maximal value. Therefore stopping the stream does not affect overall utility significantly. However, stopping a stream can benefit other streams since there will be less channel contention, and less forwarded data to the proxy. We experimentally demonstrate the effectiveness of such a threshold-based global transmission control in Section 5.

5. EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of our system using a radar trace-driven prototype implementation as well as trace-driven simulations. We use two data traces in our experiments. The first is the *Oklahoma dataset* collected from a 4-radar testbed deployed in Oklahoma (obtained from meteorologists [4]). Each radar in the testbed generates 107MB Doppler readings per 360-degree scan every 30 seconds. We collected 30 minutes of trace data from each radar. To obtain a larger scale dataset for scalability experiments, we also obtained an emulated radar data set generated by the Advanced Regional Prediction System (ARPS) emulator. The ARPS emulator is a comprehensive regional-to-stormscale atmospheric modeling system designed by the Center for Analysis and Prediction of Storms, which can simulate weather phenomena like storms and tornadoes, and generate data at the same rate as the real radars in the Oklahoma testbed. We emulated 12 radars in the emulator and collected 30 minutes of trace data from each of them. We refer to this trace as the *ARPS dataset*. The ARPS emulator takes days to generate a 30 minute trace, hence larger traces were prohibitively time consuming. Note that the actual raw data from radars can be up to an order of

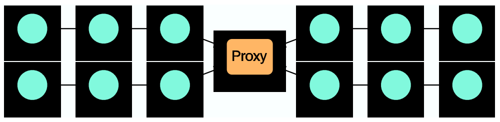


Figure 6: **The routing topology of a 13-node wireless testbed with one proxy and twelve emulated radars.**

magnitude larger than the two datasets that we used. We were limited to collecting smaller datasets by the bandwidth and storage capacity in the Oklahoma network, and the speed of the ARPS emulator.

Our radar network prototype comprises 13 radar nodes, each emulated by a Apple Mac Mini computer with an 802.11 b/g wireless card. We manually configure the nodes into a 3-hop wireless topology (shown in Figure 6) by setting their routing tables appropriately. The proxy is a server running a proxy process that collects data from radars and processes user queries. The other twelve nodes run radar processes that encode and transmit radar data. To simplify our protocol design, we use TCP as our transmission protocol, since the progressively stream needs to be received reliably and in-order for decoding. Two TCP/IP connections are built between each radar and the proxy—one for transmitting data from the radar to the proxy, the other for sending control information from the proxy to the radar. The progressive compression engine was adapted from the open-source QccPack library [9] that provides an implementation of SPIHT for images.

To evaluate performance of individual components of our system under controlled conditions, we augment prototype experiments with simulations using real traces. In order to evaluate the query processing performance of our system, we implement a query generator. Each generated query is a 4-tuple $< Type, ROI, Deadline, Priority >$. The *Type* field is the application type which can be tornado detection, wind direction estimation or 3D assimilation. The *ROI* field shows the query's region of interest which is represented by a sector of the radar's circular sensing range. The *Deadline* field represents the query's reply deadline in seconds. The *Priority* field represents the query's priority, which is determined by the user's preference to this query. We implemented two query arrival models: (i) a *Poisson arrival model* in which queries arrive at each radar as a Poisson process with configurable average arrival rate, (ii) a *deterministic model* in which queries arrive at each radar in fixed order at fixed rate. For the tornado detection query, we designed a additional model, in collaboration with meteorologists, that models query patterns during a tornado. In this model, the priority of the tornado query, and the nodes on which it is posed depends on where the tornado is predicted to be localized.

5.1 Determining the Utility Function

At the core of our system is a utility function that captures application-perceived utility as a function of the mean square error of data being transmitted by the radars. To evaluate the utility functions for the three applications, we ran the applications on lossily compressed versions of the Oklahoma dataset. We lossily compress the data traces to $\frac{1}{2^i}$ of original size with i ranging from 1 to 13. For each of these compression ratios, we measure the mean square error of the resulting data after decompression, as well as the application error after executing it on the decompressed data. Given the application error, the utilities for the applications are generated using Equation 2. Here we use fixed user requirement E_{user} in the experiments so that the utility functions only need to computed once. We fit piece-wise linear functions to utility functions, and use these functions as the utility functions in the rest of our experiments. The graph on the top of Figure 7 shows the piece-wise utility functions of the three applications obtained from our empirical evaluation. The bottom graph shows an example of how this utility function would translate to actual number of packets when a scan is compressed.

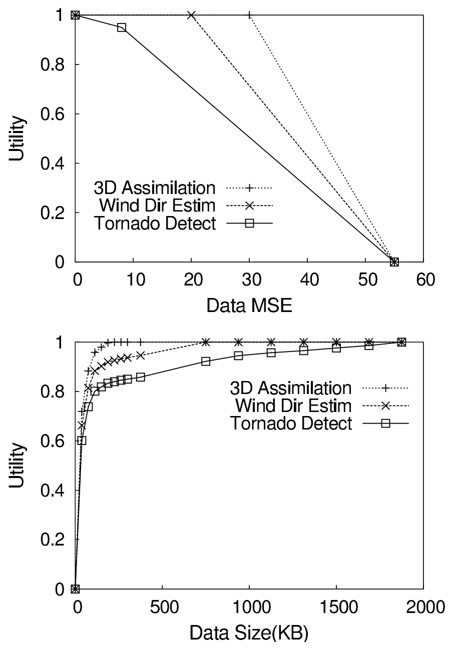


Figure 7: **Utility functions for the three applications are derived by compressing and evaluating application performance on traces from the Oklahoma dataset.**

5.2 Performance of Progressive Compression

In this section, we evaluate two main benefits of the SPIHT progressive compression algorithm: (i) higher compression rate, and (ii) adaptation to bandwidth fluctuation.

5.2.1 Compression Efficiency

The extreme data generation rates of radar sensors makes compression an essential component of radar sensor system design. In this section, we compare the compression efficiency of the SPIHT algorithm that we employ against an *averaging* compression algorithm that is currently used in the *Xrad* radar system. Each radar scan is represented as a matrix

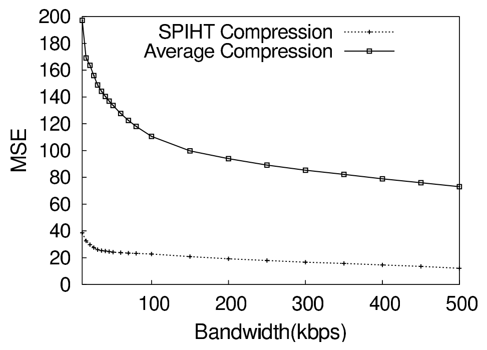


Figure 8: **Comparison of SPIHT progressive compression against averaging compression. Each algorithm compresses data to the size that can be transmitted in one epoch for a given bandwidth.**

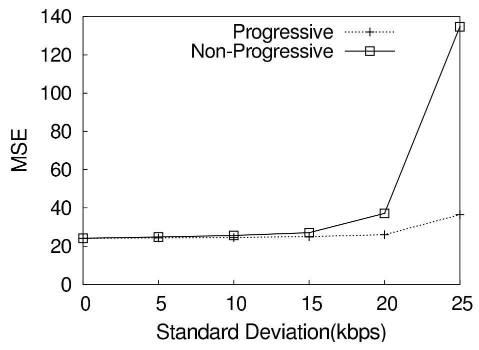


Figure 9: **Comparison of progressive compression against non-progressive compression for different levels of bandwidth fluctuation. Bandwidth fluctuation follows a normal distribution with mean 40kbps; standard deviation is varied from 0kbps to 25kbps.**

of gates x azimuths, where the radial axis is divided into gates, and the angular dimension is divided into azimuths. The averaging compression algorithm compresses data simply by averaging along the azimuth dimension. In order to compress data n times, the averaging compression algorithm averages values from n adjacent azimuths in the same gate position. The compressed data has n times fewer azimuths than the original data.

We compare the two compression algorithms using trace-driven simulations with the Oklahoma dataset. Each scan in the trace is compressed to the size s that can be sent in one epoch (30 seconds) under a fixed bandwidth B, i.e., $s = 30{\cdot}B$. The MSE of the compressed data is measured for different bandwidth settings ranging from 10kbps to 500kbps. Figure 8 shows MSE as a function of bandwidth. With increasing bandwidth, the MSE of the SPIHT algorithm decreases much more quickly than the averaging algorithm since SPIHT captures the key features of the radar scan using very few packets. Even at extremely low bandwidths such as 20kbps, the MSE of the SPIHT compressed stream is 20, whereas the MSE of the same stream with averaging compression is an order of magnitude higher at 200. This shows that SPIHT is an extremely efficient compression scheme for radar data.

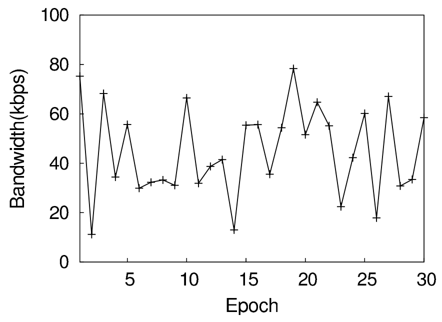
5.2.2 Bandwidth Adaptation

Next, we evaluate the ability of SPIHT to adapt to bandwidth fluctuations. SPIHT adapts to fluctuations naturally because of its progressive feature, i.e., data can be decoded progressively without receiving the entire compressed data stream. We compare it against a non-progressive compression algorithm under different levels of bandwidth fluctuation. The non-progressive algorithm is implemented by simply removing the progressive feature from SPIHT. In other words, the non-progressive SPIHT encoder first estimates how much bandwidth is highly likely to be available until the deadline of the stream, and would compresses data to that size before transmission. The data can be decoded by the proxy only after the entire compressed stream is received since no partial decoding is possible.

Unlike progressive compression where the receiver can decode even a partially transmitted stream, a non-progressive compression-based scheme has to rely on a conservative estimate of the available bandwidth to ensure the compressed data can be fully transmitted and received before the query deadline. We use a moving window estimation algorithm in our implementation. The non-progressive encoder considers a window of bandwidth values in last w epochs. The values are sorted in descending order and the 95th percentile value is taken as the estimated bandwidth. We use a window size of 20 in the experiments.

We perform a trace-driven simulation using the Oklahoma dataset where the available bandwidth in each epoch is chosen from a normal distribution with mean 40kbps. The standard deviation of the distribution is varied from 0kbps to 25kbps in steps of 5, and the resulting MSE from the two schemes is measured. Figure 9 shows MSE of the decoded data as a function of the standard deviation of the distribution. At a standard deviation of zero, the two compression algorithms achieve the same accuracy since they utilize the same amount of bandwidth. As the standard deviation increases, the bandwidth utilized by the non-progressive algorithm drops quickly, because it estimates available bandwidth conservatively. Therefore, the accuracy of the non-progressive algorithm degrades much more quickly than the progressive algorithm. For the highest standard deviation, the MSE of the non-progressive algorithm is six times more than that of the progressive algorithm.

Figure 10 gives us a time-series view of how bandwidth fluctuation impacts the two schemes. While the non-progressive scheme has high MSE due to its conservative estimate, the MSE for the progressive compression scheme follows the fluctuations in bandwidth since it is able to exploit the entire available bandwidth. The R-value for the bandwidth and MSE time-series for the progressive algorithm is -0.79, indicating robust anti-correlation: *i.e.* bandwidth is inversely correlated to the MSE.



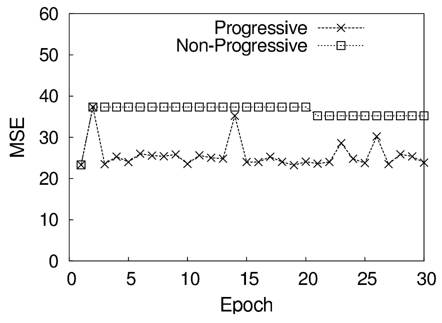


Figure 10: **Time series of bandwidth and MSE of decoded data. Bandwidth fluctuation follows a normal distribution with mean value at 40kbps and standard deviation of 25kbps.**

5.3 Performance of Data Sharing

In multi-user sensor networks with diverse end-user needs, sharing data among queries greatly improves utility of the system. We evaluate the ability of our system to handle two types of data sharing: i) queries with identical regions of interest but with different deadlines, and ii) queries with identical deadlines but with overlapping regions of interest.

5.3.1 Temporally Overlapping Queries

We first consider the case where queries have the same region of interest but have different deadlines. In this case, the progressive compression engine generates a single progressively compressed stream for both queries. The query processor decodes the compressed stream as it is received, and processes the two queries when their deadlines arise. In contrast, a system using non-progressive compression cannot easily share data between queries. We compare our approach against a non-progressive compression scheme in which data is compressed and transmitted separately for each query individually.

We evaluate the two schemes using trace-driven simulations with the Oklahoma dataset. Two queries—tornado detection and 3D assimilation—arrive at a radar every two epochs. They have different deadlines but both ask for all the data from a 360-degree scan. The tornado detection query has a deadline of one epoch, and the 3D assimilation query has a deadline of two epochs. Figure 11 shows the utility of the two schemes as bandwidth is varied from 10kbps to 100kbps. At bandwidth of 10kbps, our system achieves five times the utility of the non-progressive scheme. As the bandwidth increases, both schemes can get significant data through to the proxy, therefore the relative utility gains from our system reduces.

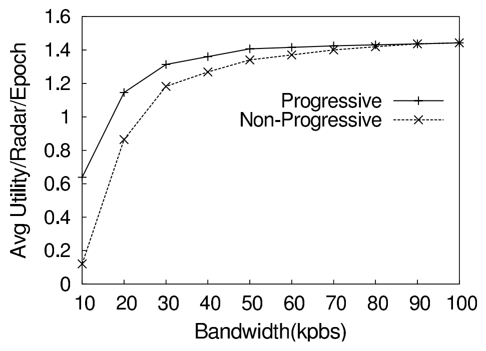


Figure 11: **Performance for temporally overlapping queries. Two queries with different deadlines but same region of interest arrive at the radar every two epochs.**

5.3.2 Spatially Overlapping Queries

We next evaluate our system's ability to handle queries with the same deadline and overlapping regions of interest. Regions that are of interest to multiple queries are weighted higher than regions of interest to just one query, and are therefore transmitted earlier and with higher fidelity than regions that are only of interest to one of the queries. We compare our scheme to a scheme without data-sharing. For the non-data-sharing scheme, data for different queries are sent separately even when there is overlap between the queries. We consider two queries, tornado detection and 3D assimilation, each of which requires data from a 180-degree sector. The degree of overlap between the regions of interest for the two queries is varied from 0 degrees to 180 degrees in steps of 30 degrees.

Figure 12 shows the end-user utility achieved by our scheme and the non-data-sharing scheme as the angle of overlap of the two queries is varied. As the angle of overlap increases, the utility gain from our scheme increases. For an overlap of 180 degrees (maximum overlap), our scheme achieves 21% higher utility than the non-data-sharing scheme.

5.4 Performance of Local Scheduler

We now evaluate the benefit of the local transmission scheduler, which always transmits the packet with the highest utility gain first. We compare this approach against an approach that uses a random transmission scheduler, which picks packets randomly from heads of the data streams. In the experiments, we simulate one radar and one server and control the available bandwidth. The tornado detection, wind direction estimation, and 3D assimilation queries arrive in round robin order at the radar at the beginning of each epoch. All queries have the same priority and the same deadline of three epochs. We run the two systems at bandwidth ranging from 10kbps to 150kbps, and seeded with the Oklahoma dataset.

Figure 13 shows the average utility per epoch as a function

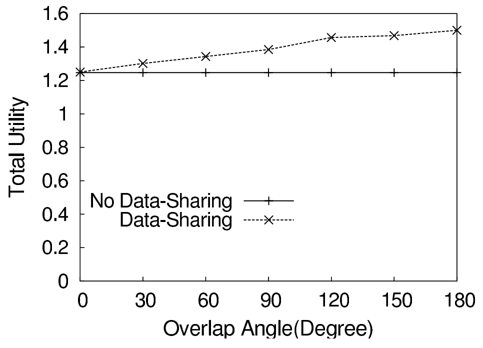


Figure 12: **Evaluation of the impact of data sharing on utility. Two applications, tornado detection and 3D assimilation, with overlapping sectors are considered.**

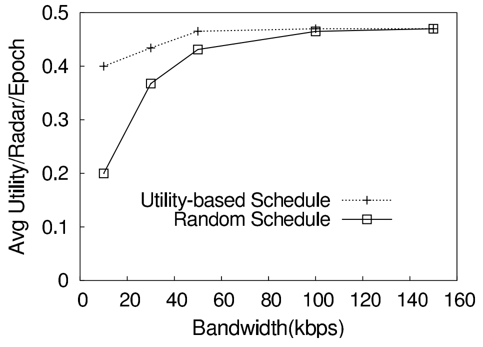


Figure 13: **Comparison of utility-driven scheduling against random scheduling.**

of bandwidth. For bandwidth lower than 150kbps, the utility-driven scheduler always achieves higher utility than the random scheduler, with as much as 100% increase in utility at low bandwidth. As bandwidth increases, the utilities of the two systems become closer. Our system performs better under low bandwidth conditions because the most important data are always sent in the first packets. When bandwidth is high enough to send all data in high fidelity, e.g., at 150kbps, there is negligible benefit from utility-driven scheduling.

5.5 Performance of Global Control

Having evaluated the performance of local transmission control, we next consider global transmission control by the proxy. Such an optimization is beneficial when there is an imbalance in query load across different regions in the network. We designed an uneven query pattern as follows - a tornado detection query with priority=3 arrives at each radar which is i-hops (i varies from 1 to 3) away from the server in each epoch, while each of the other radars has a wind direction estimation or 3D assimilation query with priority=1 in each epoch. We use the testbed consisting of one server and twelve radars as shown in Figure 6. Each radar in the testbed is seeded with a radar trace from the ARPS dataset.

Figure 14 shows the average utility per epoch with increas-

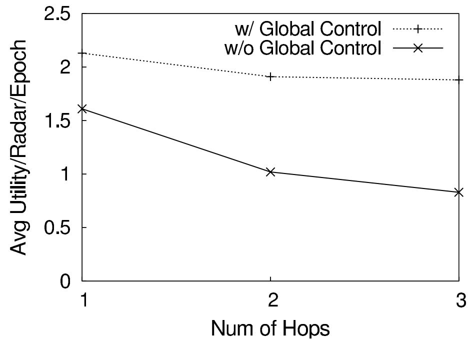


Figure 14: **Performance of global transmission control. Utility is shown for differing numbers of hops from the proxy to nodes having high-priority queries.**

ing number of hops from the proxy. The utility decreases for both of the approaches as queries with high priority arrive at nodes farther from the proxy. This is because nodes on the edge of the routing topology usually have less available bandwidth than nodes closer to the proxy, as packet loss probability increases as packets travel more hops. Thus, a query arriving at an edge node cannot achieve high utility because of the limited bandwidth, therefore, the contribution of the tornado detection query to the overall utility is reduced. However, the global control-based approach degrades much slower than the approach without global control. For instance, when the tornado query is posed three hops from the proxy, the global control-based approach achieves twice the utility of the approach without such control. This shows that global transmission control provides a simple but effective approach to deal with imbalanced query loads.

5.6 System Scalability

Until now, we have characterized the performance of individual components of our system. We now turn to full system measurement and evaluation on our testbed. Our goals are two-fold: i) to demonstrate that our system as a whole scales well with network size and number of queries per epoch, and, ii) to provide a breakdown of the utility gains provided by the different components of our system.

5.6.1 Impact of Network Size

Our first set of scalability experiments test our system at different network scales. In the experiments we use different number of nodes in the testbed shown in Figure 6 — the one and four node experiments are for a one hop topology, the eight node experiments are for a two hop topology, and the twelve node experiments are for a three hop topology. Each radar is seeded with data traces from the ARPS dataset.

The query distribution for our experiments was designed, in collaboration with meteorologists, to realistically model query patterns during a tornado. The three queries — tornado detection, wind direction estimation, and 3D assimilation — arrives at each radar as a Poisson process with average arrival rate of one query per three epochs and standard deviation of one query per epoch. The wind direction estimation queries and 3D assimilation queries are assigned weights of one or two randomly.

The priority of the tornado query, and the nodes on which it is posed depends on where the tornado is predicted to be. Meteorologists use tracking algorithms such as Extended Kalman Filters to track tornado trajectories, thereby predicting its likely location. Therefore, in our query model, we assume that the priority of tornado detection queries is three on radars where the tornado is predicted to be observed by the tracker, and is one otherwise. To generate this query pattern, we use a visual estimate from the ARPS emulator data to determine the likely centroid of the tornado.

We compare four schemes in this experiment. The existing *XRad* system with averaging compression and conservative bandwidth estimation (described in Section 5.2.2) provides us a baseline for comparison. Then, we consider three variants of our system: first, we turn on progressive compression only, then we turn on progressive compression as well as local transmission scheduling, and finally, we include global control as well. Figure 15 shows the average utilities per epoch of *XRad* and the three variations of our system.

For small networks (1 or 4 nodes), our gains over the *XRad* system are primarily due to progressive compression. For instance, when there is only one radar in the network, just the addition of progressive compression gives us 3x as much utility as the XRad scheme. Both local scheduling and global control have limited impact for the one and four node network settings, because there is limited contention and considerable available bandwidth from each node to the proxy. Thus, at a network size of one, the addition of local scheduling achieves only 4% more utility than just having progressive compression. Global control has no impact at network size 1, and limited impact at network size 4.

As system size increases, contention between nodes also increases. There is less available bandwidth per radar and more bandwidth fluctuation due to increased contention and collisions, and consequent variations in TCP window size. As a result, both local scheduling as well as global control give more gains. The benefit from these schemes increases with growing network size. For instance, the addition of local scheduling to progressive compression increases utility from 15% at network size four to 38% at network size 12. The inclusion of global control improves utility by only 4% at network size 4, but provides a 30% improvement at network of size 12.

Another point to note is the increasing difference in performance between the *XRad* scheme and our full system. With all three techniques enabled, our system achieves more than an order of magnitude improvement in utility over the XRad system for network size at 12. As network size increases from one to twelve, the utility of our system only decreases by 25%, whereas the utility of XRad decreases by 80%; this comparison demonstrates the scalability of our system.

5.6.2 *Impact of Query Load*

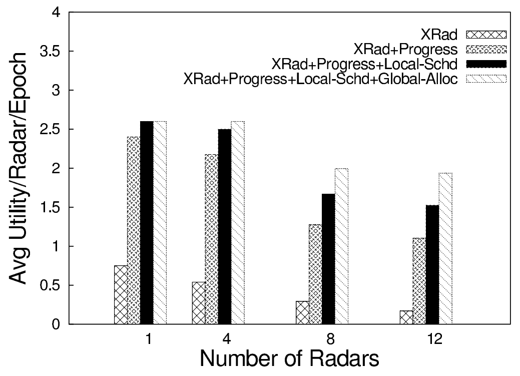


Figure 15: **Scalability to network size. Breakdown of contribution of each component of our system to the overall utility.**

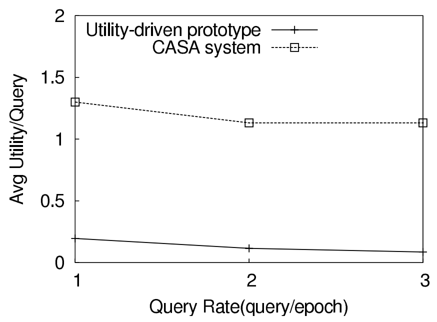


Figure 16: **System scalability to the query load.**

Our second scalability experiment stresses the query handling ability of our system. We compare our system against the *XRad* system under different query loads. Since the query processor aggregates the same type of queries into a single query in each epoch, there are at most three queries posted on each radar per epoch. We run the experiments on the wireless testbed at network size 12. Each radar node is seeded with a data trace from the ARPS emulator. We use constant query arrival rate in the query distribution for our experiments. In each epoch at most three queries of different types arrive at each radar. The priorities of the wind direction estimation queries and 3D assimilation queries are assigned one or two randomly. The priority of tornado detection queries is three on radars which the tornado is predicted to be observed by the tracker, and is one otherwise. We evaluate the two systems under different query rate ranging from one to three queries per epoch.

Figure 16 shows the average utility per query as a function of query rate. In our system, as the query rate increases, each query still gets data with sufficient accuracy to achieve high utility. Thus, the utility of XRad system decreases by 57% when the query rate increases from one to three, whereas the utility of our system only decreases by 15%. This demonstrates the scalability of our system to high query load.

6. RELATED WORK

We discuss related work not covered in previous sections.

Multi-query Optimization: A few approaches have addressed multi-query optimization in sensor networks [16, 21]. For instance, [16] considers a limited form of multi-user sharing where different users request data at different rates from different sensors, and [21] considers a multi-query optimization for simple queries such as min, max, sum, count and average. In contrast, we consider data sharing for considerably more complex applications involving spatial and temporal data sharing, and propose a general solution that can work across a variety of queries.

Utility-based Design: There is a growing body of research on utility-based approaches to address different problems in sensor networks including resource allocation in SORA [15], and sensor placement [2]. Much of this work is only peripherally related to our work. For instance, SORA employs a reinforcement learning and an economic approach for energy optimization in sensor networks [15]. The work is not designed for multi-user scenarios.

Data compression: Many techniques have used data compression to reduce communication energy overhead in sensor networks. For instance, Sadler et al. [19] consider data compression algorithms such as LZW for networks of energy-constrained devices. However, the use of compression together with multi-query optimization is a novel approach that has not been studied in the past.

Utility in Internet-based Systems: For Internet-like networks, Kelly [13] pioneered a utility-theoretic framework for rate control and, in particular, for deconstructing TCP like protocols. Such approaches have also been used for jointly optimizing routing and rate control [13, 11]. These schemes attempt to allocate resources such as bandwidth across users without consideration to data sharing between the users. Multicast rate control schemes exploit data sharing across users; but they apply to a one-to-many environment unlike MUDS that is designed for many-to-one or many-to-many environments.

7. CONCLUDING REMARKS

In this paper, we focused on a network of rich sensors that are geographically distributed and argued that the design of such networks poses very different challenges from traditional "mote-class" sensor network design. We identified the need to handle the diverse requirements of multiple users to be a major design challenge, and proposed a utility-driven approach to maximize data sharing across users while judiciously using limited network and computational resources. Our utility-driven architecture addresses three key challenges: how to define utility functions for networks with data sharing among end-users, how to compress and prioritize data transmissions according to its importance to end-users, and how to gracefully degrade end-user utility in the presence of bandwidth fluctuations. We instantiated this architecture in the context of geographically distributed wireless radar sensor networks for weather, and presented results from an implementation of our system on a multi-hop wireless mesh network that uses real radar data with real end-user applications. Our results demonstrated that our progressive compression and transmission approach achieves an order or magnitude improvement in application utility over existing utility-agnostic non-progressive approaches, while also scaling better with the number of nodes in the network.

Overall, these results demonstrate the significant benefits of multi-user data sharing in rich sensor networks. While we have considered only bandwidth optimization in this work, we are exploring joint radar sensing and bandwidth optimization in our future research. We also believe that the benefits of data sharing can apply to a wider range of applications and end-users than we have explored in this work. We plan to extend our work to camera sensor networks as well as resource-poor mote-class sensor networks in our future research.

8. REFERENCES

[1] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level Measurements from an 802.11b Mesh Network. In In *Proc. SIGCOMM*, 2004.

[2] F. Bian, D. Kempe, and R. Govindan. Utility based sensor selection. In In *Proc. IPSN*, 2006.

[3] V. Bychkovsky, K. Chen, M. Goraczko, A. Miu, E. Shih, Y. Zhang, H. Balakrishnan, and S. Madden. Cartel: A distributed mobile sensor computing system. In *Proc. SenSys*, 2006.

[4] `http://www.caps.ou.edu/`. CAPS: Center for Analysis and Prediction of Storms.

[5] K. Chebrolu, B. Raman, and S. Sen. Long-distance 802.11b links: Performance measurements and experience. In *Proc. MOBICOM*, 2006.

[6] P. R. Desrochers and S. Y. Yee. Wavelet-based Algorithm for MesoCyclone Detection. In *Proceedings of SPIE: Wavelet Applications in Signal and Image Processing*, 1997.

[7] B. Donovan, D. J. McLaughlin, J. Kurose, and V. Chandrasekar. Principles and design considerations for short-range energy balanced radar networks. In *Proc. IGARSS 2005*, 2005.

[8] `http://www.earthscope.org`.

[9] J. E. Fowler. QccPack: An Open-Source Software Library for Quantization, Compression and Coding. In *Proceedings of SPIE: Applications of Digital Image Processing*, 2000.

[10] R. Fritchie, K. K. Droegemeier, M. Xue, M. Tong, and E. S. Godfrey. Detection of Hazardous Weather Phenomena Using Data Assimilation Techniques. In *32nd Conference on Radar Meteorology*, 2005.

[11] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley. Overlay TCP for Multi-Path Routing and Congestion Control. In *Proc. of IMA Workshop on Measurements and Modeling of the Internet*, Jan 2004.

[12] K. Jain, J. Padhye, V. N. Padmanabhan, and L. Qiu. Impact of interference on multi-hop wireless network performance. *Wireless Networks*, 2005.

[13] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. In *Journal of the Operational Research Society*, volume 49, 1998.

[14] S. Liu, M. Xue, and Q. Xu. Using Wavelet Analysis to Detect Tornadoes from Doppler Radar Radial-Velocity Observations. In *Journal of Atmospheric Ocean Technology*, 2006.

[15] G. Mainland, D. Parkes, and M. Welsh. Decentralized, adaptive resource allocation for sensor networks. In *Proc. NSDI*, May 2005.

[16] R. Muller, G. Alonso, and D. Kossman. Efficient sharing of sensor networks. In *Proc. MASS*, 2006.

[17] R. Patra, S. Nedevschi, S. Surana, A. Sheth, L. Subramanian, and E. Brewer. WiLDNet: Design and Implementation of High Performance WiFi-based Long Distance Networks. In *Proc. NSDI*, 2007.

[18] B. Philips, D. Pepyne, D. Westbrook, E. Bass, J. Brotzge, W. Diaz, K. Kloesel, J. Kurose, D. McLaughlin, H. Rodriguez, and M. Zink. Integrating End User Needs Into System Design and Operation: The Center for Collaborative Adaptive Sensing of the Atmosphere (CASA). In *Proceedings of the 87th AMS Annual Meeting, San Antonio, TX, USA*, Jan. 2007.

[19] C. Sadler and M. Martonosi. Data compression algorithms for energy-constrained devices in delay tolerant networks. In *Proc. SenSys*, 2006.

[20] A. Said and W. A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6:243–250, 1996.

[21] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Multi-query optimization for sensor networks. In *Proc. DCOSS*, 2005.