Middleware Guidelines for Future Sensor Networks

Matthew Wolenetz, Rajnish Kumar, Junsuk Shin, Umakishore Ramachandran
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332–0280
Email: {wolenetz, rajnish, espress, rama}@cc.gatech.edu

***Abstract*— In the near future, we envision sensor networks to transport high bandwidth, low latency streaming data from a variety of sources, such as cameras and microphones. Sensor networks will be called upon to perform sophisticated in-network processing such as image fusion and object tracking. It is not too difficult to imagine that computational capabilities of network nodes will scale up relative to the fairly limited resources of current motes. However, it is likely that energy will continue to remain a constrained resource in such futuristic sensor networks.**

Recently, there have been proposals for middleware that provide capabilities for higher-level in-network processing while minimizing energy drain on the network. In this work, we analyze the interplay between resource requirements for compute- and communication-intensive in-network processing and resultant implications on figures of merit of interest to an application including latency, throughput, and lifetime. We use a surveillance application workload along with middleware capabilities for data fusion, role migration (simple relaying versus in-network processing), and prefetching. Through a simulation-based study, we shed light on the impact of device characteristics such as CPU speed and radio features on application figures of merit. We show, in the presence of prefetching, that increasing radio bandwidth may not improve latency nor throughput for compute-intensive workloads and may actually decrease productivity of the network. We show that cost function directed migration can significantly extend application lifetime in sensor networks with topologies two orders of magnitude larger than previous studies. We also show that a simple minded cost function may not be sufficient to guide migration decisions in the middleware.

I. Introduction

Due to their unique blend of distributed systems and networking issues, wireless sensor networks have become an active research area. Sensor networks also attract research due to the possibility they offer for supporting applications society cares about such as habitat monitoring and predicting weather. Most current sensor networks assume a homogeneous and dedicated arrangement of nodes with limited capabilities (such as Berkeley motes). Such networks have been successfully deployed for low-bit rate applications such as monitoring grape plants in vineyards.

Given the pace of technology, it is conceivable to imagine sensor networks in the near future wherein each node has the computational capability of today's handhelds (such as an iPAQ), and communication capability equivalent to 802.11b or Bluetooth. Recent advances in low-power microcontrollers (such as Intel's xScale), and increased power-conscious radio technologies lend credence to this belief. Coupled with this trend, high-bandwidth sensors such as cameras are becoming ubiquitous, cheaper, and lighter.

Thus, we envision future sensor networks to consist of deployments of high bandwidth sensor/actuator sources coupled with powerful wireless ambient processing hardware. Such a network would enable a whole host of high-bit rate, computationally intensive applications such as distributed surveillance, emergency response, and homeland security. The main characteristic of such applications is a sense-process-actuate *control loop*. Latency from sensing to actuation, and throughput are the two obvious figures of merit for such applications. In addition, an important figure of merit for such applications is network *lifetime*. By definition, sensor networks work on battery power with minimal manual supervision. Therefore, a key design consideration is the ability of the sensor network to adapt to changing application dynamics and network characteristics to meet application latency and throughput requirements, while optimizing energy usage. Middleware infrastructure for supporting such applications would include efficient stream transport, data fusion capabilities, placement and dynamic migration of fusion points in the network to adapt to changing application needs and node conditions, dynamic adaptations of the CPU and network bandwidths to changing needs, and prefetching fused items optimistically to hide latency. Recently proposed sensor network middleware infrastructures such as DFuse [1] and SensorWare [2], provide specific techniques toward supporting such applications.

Unfortunately, it is not always clear how best to tune the sensor network both in terms of hardware capabilities available in the node and in terms of middleware capabilities. For example, consider decreasing

CPU clock frequency to save processing energy. While possibly increasing network lifetime, slower processing may increase latency beyond tolerable limits for a compute-intensive application. On the other hand, if an application is communication-intensive and performance is limited by available network bandwidth at a node, then slower processing may yield the most energy savings without compromising performance. There are similar non-intuitive optimizations that are possible to fine tune the behavior of a sensor network. Thus, there is a need for a study that looks at the interplay between node characteristics and middleware features in the context of applications for such futuristic sensor networks. This is the focus of our work.

We develop a simulation framework enabling evaluation of these parameters. The simulation framework incorporates an application model that mimics a high-bandwidth sense-process-actuate control loop; a model of DFuse middleware that includes capabilities for data fusion, dynamic and locally decided role migration, and prefetching; a power model for various clock frequencies of the CPU; and a power model for two different radios, each with two models for energy accounting in idle *listen* mode. Using this simulator, and using observed latency, instantaneous throughput, number of processed items, and network lifetime as figures of merit, we evaluate the tradeoffs among the above parameters.

The contributions of this work are several:

1) An event driven simulator enabling modeling of futuristic sensor networks with on the order of 1000 nodes, surpassing previous DFuse simulators capable of modeling the middleware on only tens of nodes with neither CPU nor memory power models.
2) Quantitative results showing tradeoffs among different tunable parameters of middleware and node architecture.
3) Analysis of results to reveal non-intuitive guidelines to help tune future sensor networks.

We quantify figures of merit for a baseline configuration of middleware with only fusion capability as a function of processor speed and radio characteristics. We then quantify figures of merit with prefetching and with both prefetching and role migration.

Analysis of results for prefetching confirms intuition and reveals several non-intuitive findings. For example, except for the highly inefficient ORiNoCo radio listen mode, varying radio power saving modes (such as *sleep* versus *listen*) does not result in a significant change in the network lifetime in the presence of prefetching. Similarly, radio bandwidth differences between ORiNoCo and Bluetooth do not significantly affect latency and throughput for applications employing compute-intensive fusion operations in the network. Prefetching results in increased throughput compared to the baseline. As a corollary to this positive result, network lifetime with prefetching is lower than the baseline since more work is being done per unit time.

Analysis of migration results yields similarly interesting findings. Even directing migration with a simple cost function that is based only on available energy in a node enables extension of application lifetime, confirming viability of middleware migration in large sensor networks. In most cases we study, this cost function yields only slightly lower latency and throughput than without migration, while extending lifetime and increasing delivered items per lifetime. However, interplay between radio bandwidth and migration cost yields poor performance with this cost function when migrating large state using the slower Bluetooth interface. In this case, although application lifetime was extended by a factor of 6 times, average latency and throughput suffered badly, leading to a drop in the total number of delivered items per lifetime.

We show, once migration is enabled using this simple cost function, compute-intensive workloads on high bandwidth radios and high bandwidth CPUs may perform as well in terms of throughput, latency, and delivered items per lifetime as communication-intensive workloads on low bandwidth radios with cheap CPUs. This promising result confirms the viability of our vision of future sensor networks for supporting high bandwidth, compute-intensive in-network processing.

The rest of the paper is organized as follows. In Section II, we present a motivating application scenario along with existing middleware support. The scope of this study is summarized in Section III. The evaluation methodology that includes the simulation framework, the application and power models is presented in Section IV. Results of the study and lessons learned are discussed in Section V. Related work is presented in Section VI, and concluding remarks with directions for future research are presented in Section VII.

II. Workload and Middleware Characterization

A. Application Scenario

As a concrete motivating application, consider a campus-wide surveillance application to provide safety for people on campus. The deployed infrastructure consists of a variety of high bandwidth sensors such as cameras and microphones scattered throughout campus. Nodes of the wireless sensor network are similarly scattered across the campus to provide redundant connectivity and in-network processing resources. Actuator nodes may be PDAs carried by security officers. As data from sensors pass through the network, nodes perform application-specific *fusion functions* (such as face detection and image correlation). Such an application that

performs in-network hierarchical computation is a *fusion* application. This specific application is an instance of the general *control loop* described earlier. Energy will continue to be a primary limiting factor for such a deployment, so performing in-network fusion in an energy conscious manner is key to application longevity.

B. Middleware Requirements

Fusion application examples include streaming media, surveillance, image-based tracking and interactive vision. These applications share a common requirement of applying synthesis operations (fusion functions) upon multiple input streams in hierarchical manner. Fusion functions can be used for efficiency (e.g. compressing an input stream), or can be part of the application behavior (e.g. feature extraction from an image).

Fusion applications are typically described as a task graph, where nodes in the graph are of three types: data *source* (data producer node), *sink* (a node where a user presents requests), and *fusion* (a node which applies a fusion function). This graph is deployed as an overlay network using *relay* nodes to interconnect different nodes which are indirectly reachable. The relay nodes act as simple data forwarders. When bound to a network node, a task graph data fusion node becomes a fusion point.

To support the campus security application in a campus-wide sensor network, we need specific systems facilities: support for applying synthesis operations at fusion points, support for migration of fusion points from one dying or non-optimal network node to a more suitable node, and support to handle time-stamped data items produced from the data sources. Other middleware requirements include memory and buffer management, programming support, etc. Our work here focuses upon fusion point support and fusion point migration across network nodes because these two are the main sources of energy consumption.

Energy is *the* most critical resource in wireless sensor networks, and it is even more critical when we target high bit-rate fusion applications. Communication of one bit costs an order of magnitude higher than processing one instruction. However, with large amounts of processing occurring in-network, processing cost must be accounted for when managing energy. Similarly, large memory footprints may incur significant cost.

SensorWare [2], DFuse [1], and other middleware are available for supporting high-bit rate applications in sensor network environments. For example, DFuse supports *Fusion Channels* and their migration for supporting dynamic in-network processing of time-stamped data items. DFuse provides a framework for evaluating energy-aware cost functions locally at nodes at runtime to dynamically direct fusion point migration.

III. Scope of Study

The design space is vast: characteristics of the application including the size and shape of the deployment, and the nature of the processing involved in the network; characteristics of the middleware; and the characteristics of each node of the sensor network. We limit ourselves to studying a well-defined portion of the design space:

- Cameras are the only sensor devices. We use video streams in place of all possible kinds of streams.
- All nodes are tuned to the same characteristics for a given configuration of the sensor network.
- 64 sources, 1 sink, and 800 other nodes are deployed randomly in a square two-dimensional campus, with source nodes at the left and the sink at the right as shown in Figure 1. Node locations, though random, remain the same across our experiments. Non-uniform placements parallel real-world deployments where uniform placements and connectivity are hard to obtain. Each node begins with 1000mW*hours and disconnects when it has less than 100mW*hours. Campus size scales with radio range, keeping the starting topology constant across different experiments. The initial deployment is a connected graph.
- We use a model of DFuse as our middleware to perform fusion, migration, and prefetching of items needed for fusion. Up to 5 sets of input items can be prefetched by each fusion point. Each fusion point evaluates the *MEV* (minimize energy variance) cost function every minute to determine if and where to migrate among immediate neighbors. If a neighbor has 10mW*hours more energy than the current fusion point, then migration to the neighbor with the most energy is triggered. We assume a greedy sink that continually requests the highest level of inference and aggregation from the point in the task graph furthest from the sources, driving the application to maximize throughput.
- We use several fusion functions ranging from simple collage to complex face-detection algorithm. The former results in a *communication-intensive* workload while the latter results in a *computation-intensive* workload.

Application performance is quantified using the following figures of merit: latency, instantaneous throughput, number of processed items, and network lifetime. Given this design space, our study determines:

1) The performance of the two workloads for a baseline configuration of the sensor network without prefetching and migration. This configuration allows for globally tuning the device characteristics and carrying out fusion operations on the nodes.
2) The performance of the two workloads with

prefetching of fusion function inputs.

3) The performance of the two workloads in the presence of migration using the *MEV* cost function.

IV. Evaluation Methodology

To quantify application performance, we build an event driven simulator encompassing futuristic sensor network devices, middleware and applications. This section first presents the set of application workloads designed to be representative of general high-bandwidth, low latency, and energy constrained sensor network applications. Then, it presents sensor network node power models commensurate with recent technology. Finally, it discusses details of the design and implementation of our simulator that binds these models of application workloads and node resources to a simulated sensor network middleware, while recording energy usage by simulated network node components over time.

A. Campus Surveillance Application

We model the motivating surveillance application as a general fusion application that performs hierarchical in-network processing on streams produced initially by cameras. To arrive at a realistic model of video-based in-network processing and communication requirements, we use a representative set of fusion functions that an application can use as part of its deployed task graph. These functions are image *Collage*, which simply concatenates two input images together to produce its output; *EdgeDetect*; *Select*, whose output is the brightest of the two input images; *MotionDetect*, which is based on the calculation of a centroid of inter-frame differences and their extent; and a CPU-intensive face detection and recognition function (*FD/FR*).

Since active CPU energy consumption is related to how many cycles are required to complete one function, and memory and network energy consumption are related to the function's input and output data sizes, we report these numbers for each of the functions in Table I. For *FD/FR*, we use previously published time measurements [3] using 206MHz SA-1100 iPAQ H3600. We report measured results from our benchmarks for the remainder of the fusion functions. Our benchmarks are from a 206MHz iPAQ 3870 running Linux "familiar" distribution version 0.6.1. We believe the architecture difference between H3600 and 3870 is insignificant in this context. To verify measured time, we calculate instruction counts from assembly code generated by the gcc 2.95.2 ARM cross-compiler with "-c -g -Wa,-a,-ad" options. Because the code size of each function is small and the functions are iterative, SA-1110 with 16K-Icache and 8K-Dcache should obtain frequent cache hits and low CPI as shown in Table I.

With this set of fusion functions, we model a communication-intensive workload as an application which does not employ *FD/FR*, and we model a CPU-intensive workload with a task graph including this heavyweight image processing function.

Fusion Function	Instr Count	Cycles	Time (ms)	CPI	footprint (KB) (I/O/Runtime)
Collage	309K	803.4K	3.9	2.59	112/112/-
EdgeD	1844K	2616.2K	12.7	1.42	56/56/-
Select	327K	721K	3.5	2.20	112/56/-
MotionD	N/A	1009K	4.9	N/A	56/56/94
FD/FR	N/A	1959M	9510	N/A	30/30/3.5MB

TABLE I: Fusion Function Costs: Required number of cycles, measured time, and memory footprint.

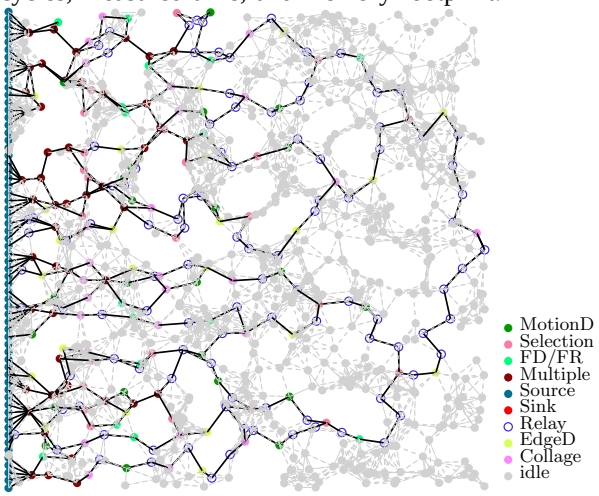

Fig. 1: Campus-Wide Surveillance Application Topology

Collage and *Select* fuse two inputs into one output, while *EdgeDetect*, *MotionDetect*, and *FD/FR* each transform a single input into one output. We compose these two classes of functions into subgraphs and connect the subgraphs to build the application task graph. Each subgraph consists of a two-into-one fusion function whose two inputs come from two one-into-one fusion functions' outputs. We randomly choose functions from the appropriate class to perform task graph construction.

We map the tree-like task graph's fusion functions onto nodes closest to an exact tree geography. Using lowest hop-count paths between fusion points adjacent in the task graph, we build relay node chains to connect the deployment. We disguise fusion point I/O mismatches by assigning the upstream function's output item size as the size of items transmitted through the relay chain to the downstream function.

Figure 1 depicts a sample topology from our experiments, prior to any fusion point migrations and node failures. 64 cameras are located along the left edge, and the sink is located in the middle of the right edge. Many nodes and links in the sensor network are idle. Darker lines indicate actively mapped links (relay chains). Some nodes host more than one fusion point simultaneously.

To show that this topology is general, imagine replicating it so multiple trees emanate in different directions

Mode	Power (mW) 802.11b @ 11Mbps	Power (mW) Bluetooth @ ∼721 Kbps
`Transmit`	1600	102
`Receive`	950	165
`Listen`	805	66
`Sleep`	60	30

TABLE II: Radio power model.

from the sink. Although there would be some overlap of links, the general hierarchical structure would remain intact. The geographical tree structure is generally representative of sensor network fusion applications, which are inherently location-based. Extension of this topology to one with multiple sinks is trivial by simply plumbing these extra sinks to the point in the hierarchy corresponding to the level of aggregation and inference required from the sources. Finally, multiple overlapping task graphs are a merger of basic structures like the topology presented here.

B. Power Models

1) Processor: Voltage scaling is a popular technique for saving energy in today's CMOS microprocessors. Energy consumption in CMOS circuits can be accurately represented as a simple equation [4] that says clock frequency reduction linearly decreases energy consumption, and voltage reduction results in a quadratic decrease in energy consumption.

From SA-1100 specification, we find that the processor consumes at most 230 mW at 133 MHz, and at most 330 mW at 206 MHz at 1.5 Volts [5]. Power measurement experiments on SA-1100 microprocessor indicate that power requirement increases monotonically with increase in clock frequency [6], [7]. Earlier research on SA-110 confirms the linear relationship too [8]. We use a linear model for energy consumption based on these two data points for determining energy usage at clock speeds from 59-206MHz.

2) Memory: Memory is also a major source of energy consumption, especially for memory-intensive workloads [6]. But, its impact on overall energy consumption is difficult to predict because a change in clock frequency changes the available memory bandwidth in a non-linear fashion, and it also affects the energy consumption for memory access [7].

For our evaluation purpose, we use a simplified model for memory access energy breakdown. We assume that memory works in three modes similar to the operation of Direct Rambus DRAM (RDRAM): *active*, *idle*, and *sleep*. Power consumption in these modes is as cited by Fan *et al.* [9] (300 mW *active*, 20 mW *idle*, 3 mW *sleep*). We assume that while the CPU is executing a fusion function, the whole memory is being accessed actively. In a realistic scenario, CPU execution and memory activity will be interleaved, and memory will keep switching between active and standby modes during CPU execution. Our assumption accounts for the worst case energy consumption by memory and it also simplifies simulation efforts.

Pouwelse *et al.* [7] report that EDO-DRAM energy consumption per MB of data read decreases monotonically with increase in clock frequency. In other words, clock frequency scaling has opposite effects on CPU and memory energy consumption. Any potential dynamic CPU scaling decision needs to address this relationship. However, in our studies presented here, we do not perform this local dynamic CPU scaling, as we are interested in first discovering fundamental tradeoffs to help direct future work with local scaling.

3) Communication: Radio is the communication medium in our target sensor network domain, and it is the most power hungry among CPU, memory, and radio. Hence, saving communication energy is critical to increasing application lifetimes.

For our simulations, power consumption for different radio modes is shown in Table II. We use numbers corresponding to two different bandwidths: one with an OriNoCo network card [10], and another for a Bluetooth radio card [11]. Though the same OriNoCo card can operate at multiple data rates, corresponding power results are not available in their specifications. We use only one transmission rate for each of the two radios. Also, Bluetooth numbers are valid only for shorter transmission range (∼66 ft for Class 2 devices) compared to the range of 802.11b (∼500 ft in open and ∼125ft in closed space). We scale campus size with respect to radio range to have the same initial topology across our experiments.

From our early experiments, we observe that energy drain by idle nodes waiting in *listen* mode for long periods of time dominates overall energy use by the network. One way of reducing this cost is to impose a duty cycle on the network nodes, enabling enables them to incur lower *sleep* radio costs for much of the time they would have been in *listen* mode otherwise. We therefore include a variant of the radio power model that assumes an optimal sleep duty cycle such that a radio never uses *listen* mode, but uses *sleep* mode instead. Having such a duty cycle incurs overhead (scheduling). Rather than imposing an arbitrary overhead onto our general sensor network model, we choose to explore the lower bound of radio cost in *listen* mode by including this optimal sleep mode as an optional radio power model. Previous research shows that such a lower bound assumption is reasonable by using an efficient radio to wake the main communication radio when necessary [12].

C. Simulator

We present here the event-driven simulator we have built to evaluate future sensor network de-

ployments under varying architectural, middleware, and workload characteristics. It consists of approximately 5700 lines of C++ code, and is available for download at http://www.cc.gatech.edu/∼wolenetz/files/basenets_04_simdfuse.tar.gz.

The simulator includes a rich set of configuration options and is extensible to support additional simulated middleware features. Currently, our simulator models a sensor network as a collection of nodes and communication links, much as in Figure 1. It supports simulation of in-network data fusion on application generated items using application specified fusion functions. It also supports fusion point migration across nodes driven by an application specified cost function such as *MEV*. The current implementation supports upwards of 1000 simulated sensor network nodes. The limiting factor is recalculation complexity of routing tables using the O(n^3) Floyd/Warshall All Pairs Shortest Path algorithm, which happens every time a node dies due to low energy.

The simulator models shared scheduling of CPU and radio resources by multiple concurrent resource requests. For example, if a node hosts two fusion points that simultaneously begin fusion function execution, the simulator serializes their access to the CPU in simulated time.

The bulk of the simulator is concerned with accurately modeling the middleware with events ranging from message delivery to migration completion. For example, if a node on one of a fusion point's input relay chains shuts down, the simulator needs to correctly destroy and rebuild that input relay chain, rebuilding the routing tables during the process. Items in-transit on the relay chain need to be accounted for, and the state of both the producer and consumer ends of the relay chain needs to be updated to account for the change. Migration uses this basic relay chain rebuild mechanism to implement the remapping of a fusion point to a neighbor node. However, to prevent the need for the old fusion point host to forward later communications to the new host, migration is delayed until there are no items in-transit along any of the migrating point's input and output relay chains. Prefetching is implemented by giving each fusion point a buffer to store fused results into, and by attaching a sink directly to every fusion point. These special sinks incur no energy or delay costs, but they drive the fusion points to request and fuse as fast as possible while they have room in their local output buffers.

We use an ideal MAC layer that incurs neither energy nor latency overhead due to packet loss. The simulator serializes, in simulated time, all access to radio channels between nodes on a pairwise basis, modeling a very simple lossless and collision-free MAC layer. Future work will relax this ideal assumption, but we are looking for basic guidelines in this current work. We assume that the routing layer provides notification of pending node battery failure piggybacked on top of regular traffic, enabling route maintenance. We currently impose no modeled overhead for local calculation of the cost function, as these are relatively infrequent and only incur minimal communication with immediate neighbor nodes (we do account for migration costs, though). We do not model the cost of initial application deployment currently, as this is highly dependent on many potential factors, primarily sensor node OS and bootstrapping characteristics.

V. Results

To answer the questions presented in the earlier scope section, we present the results of a series of experiments using our simulator with varying workload, architecture and middleware characteristics.

A. Baseline Results

First, we discuss resulting application figures of merit from simulations without using any middleware prefetching or migration features. Figure 2 presents round-trip latency, throughput, lifetime, and total number of delivered items for different CPU speeds, each of 2 radio types (ORiNoCo vs. Bluetooth), each of 2 sleep duty cycle configurations, and each of 2 application workloads. This combination of input variables leads to 8 different setups we experiment with at varying CPU speeds.

The top left graph shows round-trip latency at different CPU speeds for each of the 8 setups. An optimal sleep duty cycle makes no difference in achieved latency, so there are only 4 distinct curves visible. Sleep duty cycle makes no difference in achieved latency when all other variables are held constant because a more costly duty cycle will simply shorten application lifetime by draining nodes' energy faster, but it does not impact communication behavior. CPU-intensive application workload latency decreases as CPU speed increases. Additionally, the slower Bluetooth interface (750Kbps vs 11Mbps ORiNoCo) causes larger latency for both of the workloads. The ORiNoCo communication-intensive workload achieves the best latency.

The top right graph shows throughput at different CPU speeds. For the CPU-intensive workload, throughput increases monotonically with CPU speed increase. However, as in the latency graph, CPU speed does not affect the communication-intensive workload's throughput. Similarly, ORiNoCo achieves better throughput than Bluetooth for this baseline.

The bottom left graph depicts application lifetime, *i.e.*, the amount of time before the application's task graph becomes partitioned due to node failure. The communication-intensive workload has a longer lifetime. The effect of not using an ideal sleep duty cycle is apparent here, where expensive ORiNoCo radio listen

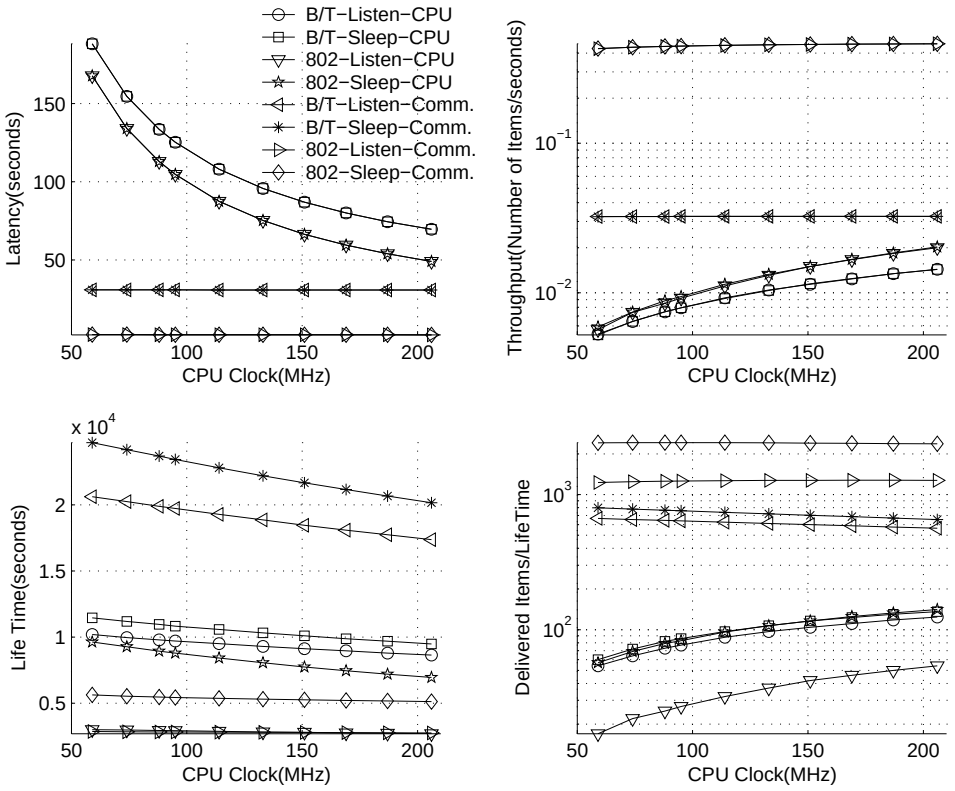

Fig. 2: Baseline results: Migration and Prefetching Disabled

energy in combination with sleep duty cycle leads to the lowest application lifetimes. Also, it is interesting to note that the communication-intensive workload has shorter lifetime than the CPU-intensive workload when using ORiNoCo and an optimal sleep duty cycle. This implies that communication is more costly than computation for the communication-intensive workload using the ORiNoCo radio. Bluetooth setups reverse this relation, giving longer lifetimes for communication-intensive workloads, and longer lifetimes overall against ORiNoCo configurations.

The final bottom right graph in Figure 2 gives the number of delivered items (per application lifetime) for different setups and CPU speeds. The communication-intensive workload, with ORiNoCo, does best because ORiNoCo provides the largest bandwidth, enabling much higher throughput while having midrange application lifetime. The CPU-intensive workload with any radio configuration results in a much lower number here, due to the high latency (and incurred CPU energy cost plus radio sleep cost) per item delivered.

B. Middleware Prefetching Results

Next, we present effects of enabling middleware prefetching upon application figures of merit for the same setups as above. Figure 3 shows the results. Results indicate that prefetching has a very good effect upon latency, cutting it in approximately half vs the baseline results, and improves throughput similarly. Prefetching at all levels in the task graph enables pipelining of items. Prefetching disguises radio differences under CPU-intensive workloads in terms of latency and throughput: maximum latency when prefetching is enabled is upper bounded by the slowest step in the pipeline, which is CPU-bound in CPU-intensive workloads.

Throughput improves with prefetching, so energy consumption per unit time in the overall network also increases for all cases, leading to a drop in lifetime. However, prefetching increases the number of delivered items per application lifetime in general for both workloads, wasting less energy overall listening or sleeping. For the same reason, when prefetching is enabled, the number of delivered items for the communication-intensive workload with an expensive listen mode rises to become close to the amount for the optimal sleep duty cycle radios.

C. Middleware Migration Results

Here, we present the effects of migration on application figures of merit while keeping prefetching enabled. Figure 4 shows the results.

The latency, although not as ”smooth” as earlier experiments, is fairly similar in trend: it improves with increasing CPU speed. Furthermore, the CPU-intensive workload continues to be the worst performing and most sensitive to CPU speed. Latency spikes seen in the Bluetooth CPU-intensive configurations result from large

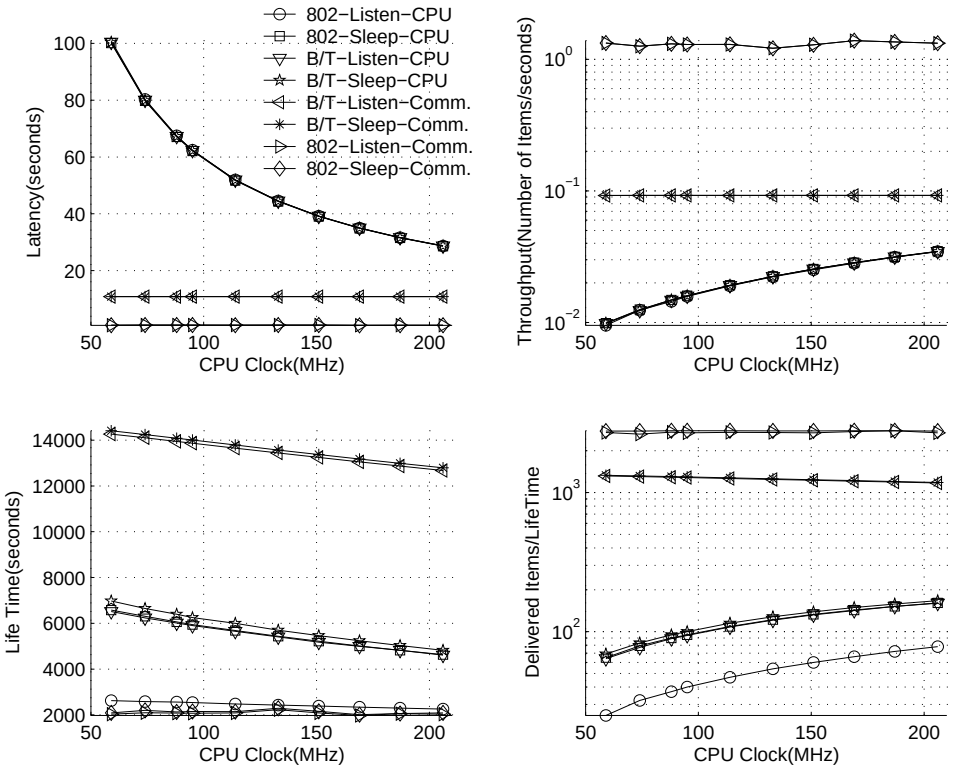

Fig. 3: Results with Prefetching Enabled

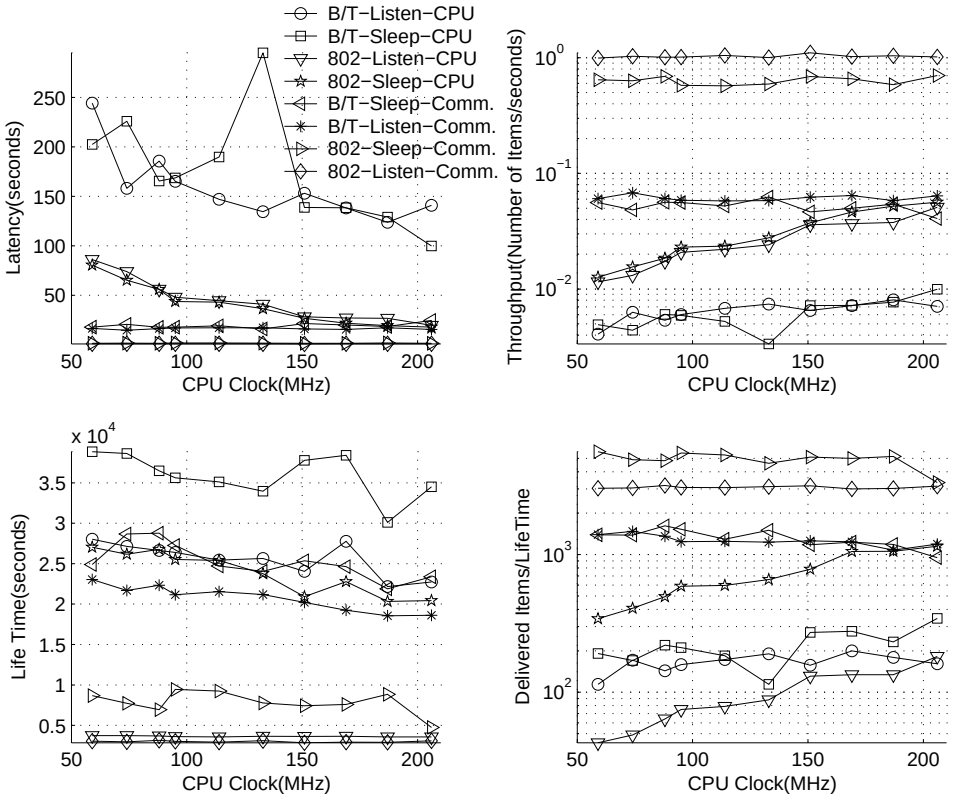

Fig. 4: Results with Prefetching and Migration Enabled

latency incurred during migration of fusion function state (which, for the CPU-intensive workload's *FD/FR*, is over 3MB). Limited Bluetooth bandwidth in combination with frequent migrations triggered by the *MEV* cost function leads to these spikes. Throughput suffers similarly under Bluetooth CPU-intensive configurations with migration.

Minor differences appear in throughput results for other configurations that are artifacts of differing migration paths chosen by MEV. For example, there is a perceivable difference now between ORiNoCo CPU-intensive workloads' throughput with and without an optimized sleep duty cycle.

Lifetime results with prefetching and migration are quite interesting. As expected, many of the configurations experienced greatly increased lifetimes. For example, the Bluetooth, CPU-intensive, optimized sleep duty cycle configuration increased its lifetime from approximately 5000-7000 seconds without migration to 30000-40000 seconds with MEV migration. This makes sense for this configuration because its radio energy drain is cheapest, and it spends much of its time performing high-latency migrations, during which a migrating channel performs no actual fusion, also limiting the usage of processors.

Delivered items per lifetime increases for all configurations when migration is enabled. This is because the *MEV* cost function enables a task graph to remap itself locally when a node hosting a fusion function has a failing battery. Migration enables an application to take advantage of redundant energy resources in the network.

Over all experiments, the CPU-intensive workload achieves increased delivered items per lifetime in general as CPU speed increases, whereas the communication-intensive workloads do not experience this trend. CPU speed increase incurs a smaller increase in energy consumption. Hence, throughput increases as CPU speed increases, with a smaller increase in energy consumption. At low CPU speeds, communication-intensive workloads are already near their maximum throughputs.

D. Lessons learned

Our results confirm intuition that latency and throughput are very closely dependent upon the architectural configuration options. From this, we can see benefits of multiple clock frequency and variable bandwidth support. A typical sensor network environment will get bursty load, and such bursts can be handled by temporarily increasing the CPU clock frequency and network bandwidth. Also, migration has to be done judiciously with a carefully chosen cost function. Indiscriminate use of migration may result in wasting energy and greatly reducing throughput.

VI. Related Work

Our work to characterize future sensor applications is motivated from recent middleware initiatives such as IrisNet [13] and DFuse [1] to support smart sensors, and current technology trends such as voltage and frequency scaling [7], [9], [14], [15]. These studies reinforce our own vision for the future evolution of sensor networks assumed in this study.

IrisNet builds on the ubiquity of cameras and their role in serving as a source for abundant information. IrisNet uses a database centric approach to publish generated data, and uses XML to query the network. Sensors send their data to a resource-rich proxy that processes the data to find interesting features and update the sensor database distributed across the network. As the processing nodes are always powered, IrisNet does not care about optimizing energy consumption.

DFuse middleware supports migration of processing across nodes to save energy or to balance load. This is done using application-specified cost functions similar to the one used in our work to test the effectiveness of migration. However, the results presented in [1] account only for transmission energy and not for energy consumed by processing or memory at a node. Also, the DFuse study, while based on an actual sensor network deployment, is limited to 12 nodes and is bound to their hardware characteristics. Our simulation-based study supports evaluation of varying middleware and architectural characteristics for sensor networks on the order of 1000 nodes.

Other simulation frameworks exist for evaluating sensor network infrastructures such as Prowler [16] and Em* [17]. Both of these wireless network simulation frameworks are specialized towards Berkeley mote sensors and communication channels. Our study focuses on modeling energy usage and performance of DFuse-like middleware for a whole range of futuristic sensor node architectures, requiring a fairly detailed implementation of the middleware inside the simulator and a decoupling from a specific target device. Relaxing the ideal MAC layer assumption is a clear avenue of future work that will benefit from existing work such as Prowler's probabilistic wireless model and Em*'s MAC simulator.

In pursuit of quality of service support and saving energy, researchers have developed techniques for adapting the application [18], middleware [19], OS [20], network protocols [21], and hardware [22]. There is a need to look into adaptation in a coordinated manner and across layers and our study is an effort in that direction.

VII. Conclusion

We envision future sensor networks to be comprised of nodes that have the computational power of today's handhelds and wireless communication capabilities of

today's 802.11b and Bluetooth. This vision opens up the possibility for supporting a variety of interesting applications that have a sense-process-actuate control loop involving high bandwidth streams and complex processing of such streams. Middleware for supporting such applications would include capabilities such as fusion, migration of fusion points, prefetching items needed for fusion, and the ability to dynamically change the device characteristics. Using a simulation-based study, we quantify the interplay between the device characteristics and middleware features for supporting an instance of a control loop application, namely, distributed surveillance. There are quite a few non-intuitive results from the study. In the presence of prefetching, the capacity of the radio beyond a threshold does not lead to improved latency for compute-intensive workloads. Indiscriminate use of migration can result in poor latency compared to a baseline configuration with no migration. With migration and a simple cost-function, CPU-intensive workloads on high bandwidth radios and high bandwidth CPUs may achieve the same throughput, latency, and productivity as communication-intensive workloads on low-bandwidth radios with cheap CPUs.

The design space covered in this study is only part of what is possible. Future work includes investigating cost functions that are more aware of the dynamic state of the application and the device characteristics and not just the instantaneous energy level. As wireless is collision- and error-prone, incorporating a more realistic MAC layer model for dense sensor networks is future work. Also, it would be interesting to study the effects of dynamically scaling devices' characteristics at each network node.

ACKNOWLEDGMENTS

The work has been funded in part by an NSF ITR grant CCR-01-21638, NSF grant CCR-99-72216, HP/Compaq Cambridge Research Lab, the Yamacraw project of the State of Georgia, and the Georgia Tech Broadband Institute. The equipment used in the experimental studies is funded in part by an NSF Research Infrastructure award EIA-99-72872, and Intel Corp.

REFERENCES

[1] R. Kumar, M. Wolenetz, B. Agarwalla, J. Shin, P. Hutto, A. Paul, and U. Ramachandran, "DFuse: a framework for distributed data fusion," in *Proceedings of the first international conference on Embedded networked sensor systems*. ACM Press, 2003, pp. 114–125.

[2] A. Boulis, C. C. Han, and M. B. Srivastava, "Design and implementation of a framework for programmable and efficient sensor networks," in *The First International Conference on Mobile Systems, Applications, and Services(MobiSys)*, San Francisco,CA, 2003.

[3] U. Kremer, J. Hicks, and J. Rehg, "A compilation framework for power and energy management on mobile computers," 2001. [Online]. Available: citeseer.ist.psu.edu/kremer01compilation.html

[4] T. Burd and R. Brodersen, "Processor design for portable systems," 1996. [Online]. Available: citeseer.ist.psu.edu/burd96processor.html

[5] "Intel StrongARM SA-1100 Developer's Manual," *Document no. 278088-04, Intel*, 1999.

[6] M. A. Viredaz and D. A. Wallach, "Power evaluation of a handheld computer," *IEEE Micro*, 2003.

[7] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic voltage scaling on a low-power microprocessor," in *7th ACM Int. Conf. on Mobile Computing and Networking (Mobicom)*, Rome, Italy, July 2001, pp. 251–259.

[8] J. Montanaro, R. T. Witek, K. Anne, A. J. Black, E. M. Cooper, D. W. Dobberpuhl, P. M. Donahue, J. Eno, G. W. Hoeppner, D. Kruckemyer, T. H. Lee, P. C. M. Lin, L. Madden, D. Murray, M. H. Pearce, S. Santhanam, K. J. Snyder, R. Stephany, and S. C. Thierauf, "A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor," *Digital Tech. J.*, vol. 9, no. 1, pp. 49–62, 1997.

[9] X. Fan, C. Ellis, and A. Lebeck, "Memory controller policies for dram power management," in *Proceedings of the 2001 international symposium on Low power electronics and design*. ACM Press, 2001, pp. 129–134.

[10] "ORiNOCO PC Card Specification: *http://www.hyperlink tech.com/web/orinoco/ orinoco_pc_card_spec.html*," 2003.

[11] "OKI Semiconductor: ML7050LA Specification *http://www.oki.com/semi/english/t-blue.htm*," June 2001.

[12] Y. Agarwal and R. K. Gupta, "On Demand Paging Using Bluetooth Radios on 802.11 Based Networks," Center for Embedded Computer Systems, UC Irvine, UC San Diego, Tech. Rep. 03-22, July 2003.

[13] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, "Irisnet: An architecture for a worldwide sensor web," vol. 2, no. 4, 2003.

[14] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *ACM Symposium on Operating Systems Principles*, 2001, pp. 89–102. [Online]. Available: citeseer.ist.psu.edu/pillai01realtime.html

[15] G. Semeraro, D. H. Albonesi, S. G. Dropsho, G. Magklis, S. Dwarkadas, and M. L. Scott, "Dynamic frequency and voltage control for a multiple clock domain microarchitecture," in *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*. IEEE Computer Society Press, 2002, pp. 356–367.

[16] G. Simon, P. Volgyesi, M. Maroti, and A. Ledeczi, "Simulation-based optimization of communication protocols for large-scale wireless sensor networks," in *Proceedings of IEEE Aerospace Conference*, March 2003.

[17] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin, "Em*: a Software Environment for Developing and Deploying Wireless Sensor Networks," in *Proceedings of USENIX 04*, 2004.

[18] E. de Lara, D. Wallach, and W. Zwaenepoel, "Puppeteer: component-based adaptation for mobile computing (poster session)," *SIGOPS Oper. Syst. Rev.*, vol. 34, no. 2, p. 40, 2000.

[19] C. Poellabauer, H. Abbasi, and K. Schwan, "Cooperative run-time management of adaptive applications and distributed resources," in *Proceedings of the tenth ACM international conference on Multimedia*. ACM Press, 2002, pp. 402–411.

[20] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile application-aware adaptation for mobility," in *Proceedings of the sixteenth ACM symposium on Operating systems principles*. ACM Press, 1997, pp. 276–287.

[21] W. Ye, J. Heidemann, and D. Estrin, "An Energy-Efficient MAC protocol for Wireless Sensor Networks," in *Proceedings of INFOCOM 2002*, New York, New York, June 2002.

[22] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*. IEEE Computer Society, 1995, p. 374.