Characterizing Middleware Mechanisms for Future Sensor Networks

Matthew Wolenetz
Proposal for Thesis
Advisor : Prof. Umakishore Ramachandran
College of Computing
Georgia Institute of Technology,
Atlanta, GA 30332, USA

1 Overview

Due to their unique blend of distributed systems and networking issues, wireless sensor networks (SN) have become an active research area. Most current SN use an arrangement of nodes with limited capabilities. Given SN device technology trends, we believe future SN nodes will have the computational capability of today's handhelds, and communication capabilities well beyond today's "motes", satisfying application demand for greater capabilities for performing computations in-network on higher bit-rate streaming data.

We focus on stream-based future SN applications, such as automated surveillance, that perform in-network streaming *data fusion* operations, such as face detection, in a hierarchical fashion to produce high-level inferences to guide actuation decisions, forming a *control loop*. Such an application that performs stream-based in-network hierarchical computation is a *fusion* application. Energy will continue to be a primary limiting factor for future SN, so performing in-network fusion in an energy-conscious manner is key to application longevity. There exists a need to study tradeoffs in terms of how much productivity an application can achieve during its lifetime, how application latency and throughput requirements affect both lifetime and productivity, and how various available middleware and device capabilities for performing low-power communication and processing impact these performance metrics. In the following we briefly introduce this problem and then outline the research we are carrying through.

1.1 Problem Statement

For future SN to successfully support stream-based fusion applications, they will need to be constructed to achieve application throughput and latency requirements while minimizing energy usage to increase application lifetime. We anticipate dynamic, bursty fusion application behavior due to their interface with dynamic pervasive computing environments. This thesis investigates some existing and new middleware mechanisms for improving application lifetime while achieving required latency and throughput, in the context of a variety of SN topologies and scales, models of potential fusion applications, and device radio, CPU, MAC, and routing capabilities. We expect tradeoffs exposed by this investigation to inform a model for how to construct a SN in terms of node capabilities and tuning parameters for the studied middleware mechanisms, given application characteristics and performance requirements, and given network topology and scale.

1.2 Research Outline

We evaluate and extend a set of mechanisms used by our recent novel middleware, *DFuse*, for application-directed energy management of future SN fusion applications. Our simulation-based evaluation enables modeling a variety of applications, network scales, network layers, and device capabilities to determine how each middleware mechanism impacts performance for a SN context. We extend the set of existing mechanisms (dynamic fusion point migration and optimistic data prefetching) to include local CPU scaling and predictive prefetching to better adapt to bursty workloads while employing an emerging device power management capability.

2 Design Space

2.1 Future Sensor Networks

Due to their unique blend of distributed systems and networking issues, wireless sensor networks (SN) have become an active research area. SN also attract research due to the possibility they offer for supporting applications society cares about such as habitat monitoring and weather prediction. Most current SN assume a homogeneous and dedicated arrangement of nodes with limited capabilities (such as Berkeley motes [34, 20, 17]). Such networks have been successfully deployed for many low bit-rate applications, for example seabird habitat monitoring [27] and grape plant monitoring in vineyards [8].

Given the pace of technology, it is conceivable to imagine SN in the near future wherein each node has the computational capability of today's handhelds (such as an iPAQ), and communication capabilities equivalent to Bluetooth, 802.11a/b/g, UWB, or even 802.15.3 (up to 55Mbps). Recent advances in low-power microcontrollers, and increased power-conscious radio technologies lend credence to this belief. For example, next generation iMote prototypes [20] and Telos motes [34] are available for research now. Although not as computationally powerful as a modern iPAQs, iMotes provide 12MHz 32-bit ARM7TDMI processors and 64KB RAM/512KB FLASH, a significant increase in capability above Berkeley mote MICA2 [17] predecessors that only had 8MHz 8-bit ATmega128L microcontrollers with 640KB FLASH. Furthermore, the wireless bandwidth available with iMotes is Bluetooth based (up to over 600Kbps application-level bandwidth), greatly exceeding Berkeley motes' 38.4Kbps data rate. Similarly, Telos motes, designed for long lifetime with very low duty cycles, energy-efficient idle modes and faster, energy-efficient microcontrollers and radios, provide increased computation and communication capabilities over previous generation motes. We believe this trend will continue as SN applications demand ever greater capabilities for performing computation on high bit-rate data within the network. It is conceivable that recent hardware capabilities enabling CPU frequency and voltage scaling for power saving, *e.g.*ARM xScale packages, will be integrated into future SN devices. Already, such technology is integrated into Stargate devices [18], providing higher capability backbones for mote-based SN. Coupled with this trend, high-bandwidth sensors such as cameras are becoming ubiquitous, cheaper, and lighter (in this case, possibly due to the large-scale demands of cell-phone manufacturers for these cameras, currently on the order of over 20 million annually for Nokia alone [43]).

Thus, we envision future SN 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* enabled by in-network processing of streaming data. 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, SN operate

on battery power with minimal supervision. Therefore, SN applications have a limited operational time before the network becomes partitioned due to energy consumption. There exist tradeoffs in terms of how much productivity an application can achieve during this lifetime, how application latency and throughput requirements affect both lifetime and productivity, and how various available device capabilities for performing low-power communication and processing impact these performance metrics.

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 still 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.

2.2 Application Domain

As a concrete motivating application, consider a campus-wide automated surveillance application to provide safety for people on campus. The deployed infrastructure consists of a variety of sensors such as cameras and microphones scattered throughout campus. Nodes of the wireless SN are similarly scattered across the campus to provide redundant connectivity and in-network processing resources. Actuator nodes may be PDAs carried by security officers, or other SN resources such as pan-tilt-zoom motors attached to cameras. As data from sensors pass through the network, nodes perform application-specific *fusion functions* (such as face detection, image correlation, and higher level inferencing). This specific application is an instance of the general *control loop* described earlier, where both automated and ”human-in-the-loop” actuation decisions result from in-network communication and computation. 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.

Other fusion application examples include streaming media, image-based tracking, interactive vision, and feature extraction for continuous queries used by applications such as EventWeb [29]. 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 indirectly reachable nodes. Relay nodes act as simple data forwarders. When bound to a network node, a task graph data fusion node becomes a fusion point.

Figure 1 shows a tiny example task graph of a surveillance application. The filter function selects images with some interesting properties (*e.g.* rapidly changing scene), and sends the compressed image data to the collage function. The collage function decompresses the images coming from possibly different locations, combines the images and sends the composite image to the root (sink) for further processing. We will return to both this tiny task graph and the hypothetical campus surveillance application in more detail later in this proposal.

To support fusion applications, 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

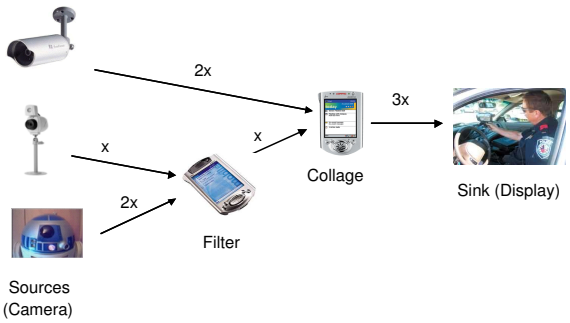


Figure 1. An example surveillance application that uses in-network distributed data fusion. Edge labels indicate relative (expected) transmission rates of data sources and fusion points.

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 so far in this design space has used the simplifying assumption of a constant, predictable amount of computation and communication to perform a particular fusion operation. We plan to relax this assumption to reflect more realistic SN application workloads that exhibit bursty and self-similar characteristics, in terms of demand for outputs at the task graph root (sink). Such behavior has been observed in network traffic [25, 31], and has been useful in understanding how to size IT infrastructure for supporting web workloads, by estimation of the *Hurst* exponent [28]. We will investigate reversal of this process (generation of a workload, given parameters including a Hurst exponent) as a potential way of generating bursty and self-similar workloads to assist our evaluations. An alternative method for generating an easily parameterized dynamic workload would be to employ a Poisson process model, though possibly not as faithful to real workloads.

2.2.1 Network Layers

Our work so far assumes that any SN node is initially reachable from any other node, and assumes a routing layer that exposes hop-count information between any two nodes in the network. As energy is drained on nodes due to computation, communication and idling overheads, nodes may "die", eventually causing network partition. Typically, these assumptions can be satisfied by a separate layer that supports a routing protocol for ad hoc networks, like Dynamic Source Routing (DSR) [19], and exposes an interface to query the routing information.

However, these assumptions ignore the overhead in terms of energy and time used for maintaining routing information. Similarly, our work so far assumes an ideal MAC layer, ignoring potentially significant energy and latency overheads caused by collision, non-ideal MAC scheduling, and noise [12]. As will become clear during our evaluation methodology and results presentations in subsequent sections, we plan to leverage existing models for a variety of available MAC and routing layers to investigate tradeoffs caused by their overheads relative to application requirements, device capabilities and topology, and middleware mechanisms.

2.3 Devices and Network Layers Considered

Where we have included device-level bandwidths and resource consumption in our exploration, we have used models based on ORiNOCO 802.11b and Bluetooth $\sim$721Kbps radio specifications. We do not anticipate greatly extending the set of radio models we consider, as the design space is already quite large. However, if there are models of other

radio devices coupled with the existing MAC and routing layer models we plan to leverage, we will potentially include those additional radio device models.

Similarly, we have limited the scope of our exploration of CPU capabilities to a linear model of CPU speed and consequent power consumption, based on published experiments of SA-1100 and SA-110 processor power consumption at various frequencies and voltages. We present our specific processor power model later in this proposal. Since evaluation of a CPU-scaling middleware mechanism is a primary goal of this work, we will incorporate appropriate CPU models based on more recent studies as work progresses.

To constrain the search space, we have used a simple power model for memory in a SN node. Specifics of these models are reviewed later in this proposal and are available in our published results [44]. Again, power models appropriate to our proposed CPU scaling scheme will be explored as part of this work. For example, Pouwelse *et al.* [35] 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.

2.4 Related Work

It is well-recognized that energy is critical in SN, driving a significant amount of recent research into mechanisms for SN energy optimization. Most current SN research focuses on contemporary devices and device models for low-bit rate communication and minimal in-network computation, rather than on mechanisms for supporting high-bit rate communication with significant in-network computation. Approaches for SN energy optimization range from hardware [34, 20], MAC [45, 40], routing [41, 6], cross-layer approaches [21], and application-specific optimizations such as energy-efficient target tracking [14]. Additionally, there have been middleware approaches to bridge the gap between application and lower layers [16, 24].

Recent research in power-aware routing for mobile ad hoc networks [41, 6] proposes power-aware metrics for determining routes in wireless ad hoc networks. We use similar metrics to formulate different cost functions for guiding our fusion point migration mechanism. While designing a power-aware routing protocol is not the focus of this thesis, routing protocol information may possibly be usable for defining more flexible cost functions or for informing our proposed predictive prefetcher and CPU scaling mechanisms.

Similarly, this thesis does not propose a cross-layer algorithm for SN energy optimization, although recent analytical work [21] in this area may assist with characterizing performance bounds. In this particular approach, the low-level scheduling and power control problem that optimizes energy usage for application QoS is shown to be NP-Complete, and the proposed algorithm is centralized, limiting its applicability in distributed SN environments. However, the observation of the intractability of optimal scheduling further motivates our proposed distributed heuristics.

Research into application-specific SN energy optimizations propose evaluation metrics suitable to the applications being studied. An example metric is *QoSv* [14], or “quality of surveillance”, determined by how far a target moves before the sensor network detects it. Our research focuses on mechanisms to support more general streaming fusion applications, so we choose application figures of merit applicable and important to these applications, including latency, throughput and lifetime.

Our approach focuses on middleware techniques for SN energy optimization, to bridge the gap between stream-

based application requirements and low-level device and network layer capabilities. MiLAN [16] has the most similar goals to our DFuse [24] work, providing a set of middleware mechanisms for adapting the SN to effect application supplied performance policy. Our example campus surveillance SN fusion application could be accommodated to some degree by MiLAN, however that middleware does not provide the combination of general streaming data abstractions for in-network computation along with approaches for optimizing the energy usage given application latency and throughput requirements.

Beyond our initial prototype implementation and evaluation, we have built a simulation-based evaluation framework for our middleware. Prowler [39], TOSSIM [26], and Em$*$ [13] simulators and emulator are specialized towards Berkeley mote sensors and communication channels. Our study focuses first on modeling energy usage and performance of a variety of middleware mechanisms 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. As we relax the ideal MAC and routing layer assumptions in our simulation-based evaluation of these middleware mechanisms, coupling our middleware simulator with an existing wireless network layer simulator will be an immediate objective. Of the available simulator options, we will likely proceed with GloMoSim [3] rather than ns2-wireless [7], as GloMoSim provides practical support for larger scale wireless deployments than ns2-wireless, critical to successful evaluation of our middleware model.

3 Approach

Our approach focuses upon evaluation of several adaptive middleware mechanisms for achieving application required performance while minimizing energy usage. In the following, we introduce our core *Fusion Channel* middleware abstraction, followed by two application-directed performance management mechanisms we have studied so far: fusion point migration and "optimistic" prefetching to hide latency. We then present our evaluation methodology, details of implementation and results of experiments conducted so far.

3.1 Fusion Channel Abstraction

The *Fusion Channel* middleware abstraction, introduced in our recently proposed DFuse middleware [24], aims to simplify the application of programmer-supplied transformations to correlated sets of input items from sequenced input streams, producing a (possibly shared) output stream of "fused items." It does this by providing a high-level API for creating, modifying, and manipulating fusion points that subsumes certain recurring concerns (failure, latency, buffer management, prefetching, mobility, sharing, concurrency, etc.) common to fusion environments such as SN. We have published a full description of the design, prototype implementation and API microbenchmark evaluation of this abstraction [24].

3.2 Fusion Point Migration

Of specific note in this proposal, DFuse uses a distributed role assignment algorithm for placing fusion points in the network. Role assignment is a mapping from a fusion point in an application task graph to a network node. Given an application task graph provided to a designated root SN node along with a parameterized cost function, distributed role assignment outputs an overlay network that optimizes the role to be performed by each node of the network. The "goodness" of the role assignment is with respect to the input cost function. The distributed algorithm executes periodically to reevaluate the mapping in a local fashion. If a locally "better" mapping of a fusion point is determined, then the fusion point is migrated to the new host node.

Fusion point migration can be used to optimize a variety of application figures of merit. Most importantly, we hypothesize that it can be used to dynamically minimize the energy used by the task graph's overlay network as the network conditions and application behavior changes, and consequently increase application lifetime. We have considered three cost functions for directing fusion point migration:

1. **MT2** "Minimize Tranmission Cost": This cost function aims to decrease the amount of data transmission required for running a fusion function. Input data needs to be transmitted from sources to the fusion point, and the output data needs to be propagated to the consumer nodes (possibly across hops). For a fusion function *f* with *m* input data sources (fan-in) and *n* output data consumers (fan-out), the transmission cost for placing *f* on node *k* is formulated as:

$$c_{MT2}(k,f) \;=\; (\; power(k) \;<\; threshold\;)\;?\;(\; INFINITY \;:\; \sum_{i=1}^{m} t(source_i) * hopCount(input_i, k) + \sum_{j=1}^{n} t(f) * hopCount(k, output_j))$$

   Here, *t(x)* represents the transmission rate of the data source *x*, and $hopCount(i,k)$ is the distance (in number of hops) between node *i* and *k*.

2. **MPV** "Minimize Power Variance": This cost function tries to keep the power of network nodes at similar levels. If $power(k)$ is the remaining power at node *k*, the cost of placing *any* fusion function on that node is:

$$c_{MPV}(k) = 1 \;/\; power(k)$$

3. **MTP** "Minimize the Ratio of Transmission cost to Power": This cost function aims to decrease both the transmission cost and lower the difference in the power levels of the nodes. The intuition here is that the cost reflects how long a node can run the fusion function. The cost of placing a fusion function *f* on node *k* can be formulated as:

$$c_{MTP}(k,f) = c_{MT2}(k,f) * c_{MPV}(k)$$

3.3 "Optimistic" Prefetching

Fusion Channels, as implemented in our prototype, each have an associated output buffer, containing fused output data not yet retrieved by all consumers. We accommodate up to 5 sets of input items that can be prefetched and fused before the output buffers become full. As implemented, Fusion Channels will greedily attempt to keep their output buffers full by requesting their next inputs when they are idle and observe free output space. With prefetching occurring at all task graph fusion points and the sink, the in-network processing should become pipelined, with latency approximating the slowest pipeline stage rather than a complete round trip through the pipeline. Although this prefetching should benefit latency, it will also increase the local memory footprints and the state communicated during fusion point migrations. Our prefetching mechanism is "application-directed" in the sense that an application can request it be enabled or disabled as part of the task graph specification presented during startup.

4 Evaluation

Our evaluation goal is to investigate these middleware features for improving application lifetime while achieving required latency and throughput, in the context of a variety of SN topologies and scales, models of potential fusion applications, and device radio, CPU, MAC, and routing capabilities.

4.1 Initial DFuse Implementation

Due to the complexity of interactions between middleware mechanisms, application workloads and device capabilities, a purely analytical approach to evaluating our middleware mechanisms is not feasible. These mechanisms employ local heuristics that operate without global knowledge, motivating experimentation and simulation to determine their effectiveness for various SN applications. There are two reasons we limit the mechanisms to be local heuristics. Primarily, gathering global context for performing dynamic adaptations incurs communication costs, potentially reducing the performance of the SN. Second, even if we used a global heuristic, determining an optimal mapping of a fusion application to the SN for comparative evaluation rapidly becomes infeasible as the scales of application and SN increase (this problem equates to the NP-hard general Steiner tree problem).

For confirming the utility of our core middleware fusion point migration feature, we have implemented the fusion channel abstraction along with a simulator of the role assignment mechanism for evaluation on a small iPAQ farm with a simple application by [24]:

1. Implementing a multi-threaded architecture for the fusion module that supports the basic Fusion Channel API calls and the prefetching mechanism. This implementation employs a programming system called Stampede [36, 1] to meet the fusion module's infrastructural requirements for timestamping data produced from different sensors, and a reliable transport layer for moving data through the network. Additional porting of the Stampede system to the target ARM-Linux architecture was done, including re-tuning a proprietary reliable UDP cluster messaging layer to perform better on wireless platforms.

2. Implementing the placement module that supports the role assignment tasks for cost-function directed dynamic fusion point migration. For ease of evaluation, we have decoupled the fusion and placement module implementations, interfacing them with a built-in communication channel and a protocol that facilitates dynamic task graph instantiation and adaptation using the DFuse API. Transmission rates exhibited by the application are collected by this interface and communicated to the placement module for use as cost function inputs.

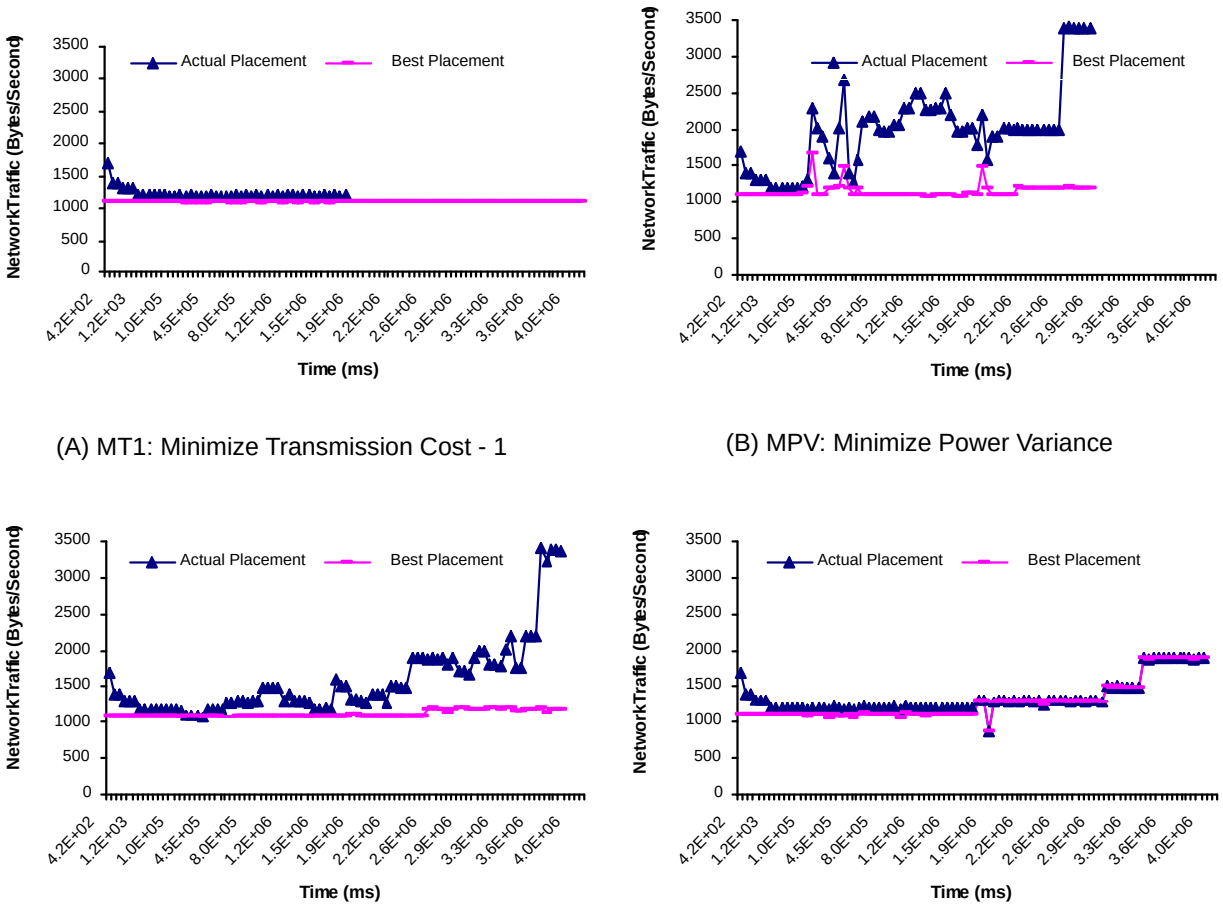
4.2 DFuse Implementation Results

Using our initial DFuse middleware implementation on a 12-node iPAQ wireless "farm", with a simple tracking fusion application containing two dynamically migrated application fusion points (similar to Figure 1), we have discovered [24]:

1. Fusion point migration, directed by application cost function, for a small application on a small SN deployment, can definitely increase application lifetime while maintaining constant (low) throughput, when the energy model used to determine lifetime is driven purely by application-level communication amounts. Figure 2A shows the energy/time performance when migration is disabled after an initial optimization period (using a cost function very similar to **MT2** for this optimization). All of **MPV**, **MTP** and **MT2** (Figures 2B-D) realize greater application lifetime, due to cost-function directed fusion point migration.

2. A cost function aimed at minimizing transmission costs (**MT2**) achieves close to an optimal minimum transmission cost once past a brief initial mapping stabilization (Figure 2A).

3. A cost function aimed at minimizing battery variance across SN nodes (**MPV**) reduces variance by a factor of 4 (Figure 3A), at the cost of many more fusion point migrations (role transfers) and lower lifetime than **MT2** (Figure 3B).



(A) MT1: Minimize Transmission Cost - 1

(B) MPV: Minimize Power Variance

(C) MTP: Ratio of Transmission Cost to Available Power

(D) MT2: Minimize Transmission Cost - 2

Figure 2. The network traffic timeline for different cost functions. X axis shows the application runtime and Y axis shows the total amount of data transmission per unit time.

4. A cost function that attempts to minimize transmission cost while maximizing the time until the fusion point host dies (**MTP**) achieves comparable lifetime to **MT2** (Figure 3B) and comparable variance to **MPV** (Figure 3A).

5. Microbenchmarks of our fusion abstraction's API (Figure 4) reveal as much as 74.5% latency overhead for streaming item fetching (*getFCItem*) beyond that predicted by analysis of application-level messages communicated along with measured maximum performance of the network layer. This latency overhead includes computation, synchronization and network layer latencies incurred during message handling, and varies considerably across multiple trials. These observations are explained by the potentially high latency cost of the wireless channel, motivating both a prefetching mechanism and accounting for energy usage for retransmissions to obtain more accurate energy/performance tradeoff results.

4.3 Simulator Framework

To support further evaluation in the context of larger network and application scales and a variety of device capabilities, we have built a simulator of our DFuse middleware [44]. Our previous DFuse evaluations modeled cost solely

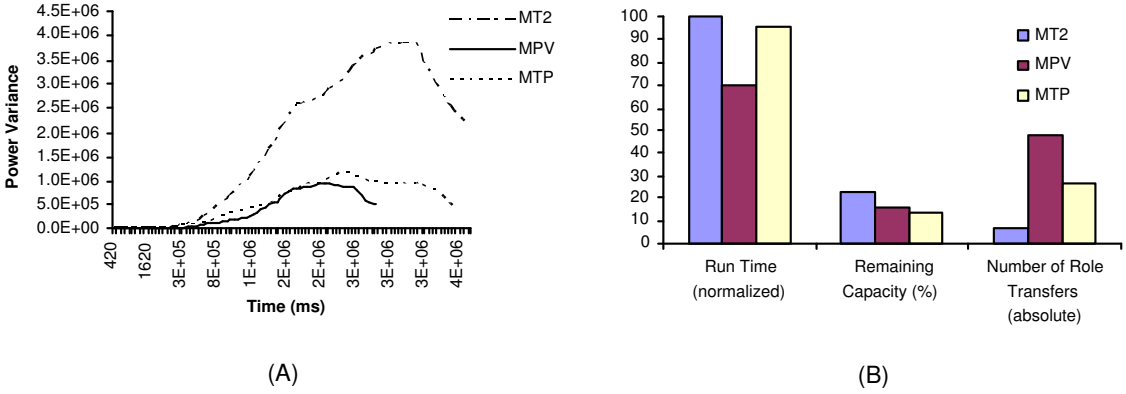


Figure 3. Comparison of different cost functions. Application runtime is normalized to the best case (*MT2*), and total remaining power is presented as the percentage of the initial power.

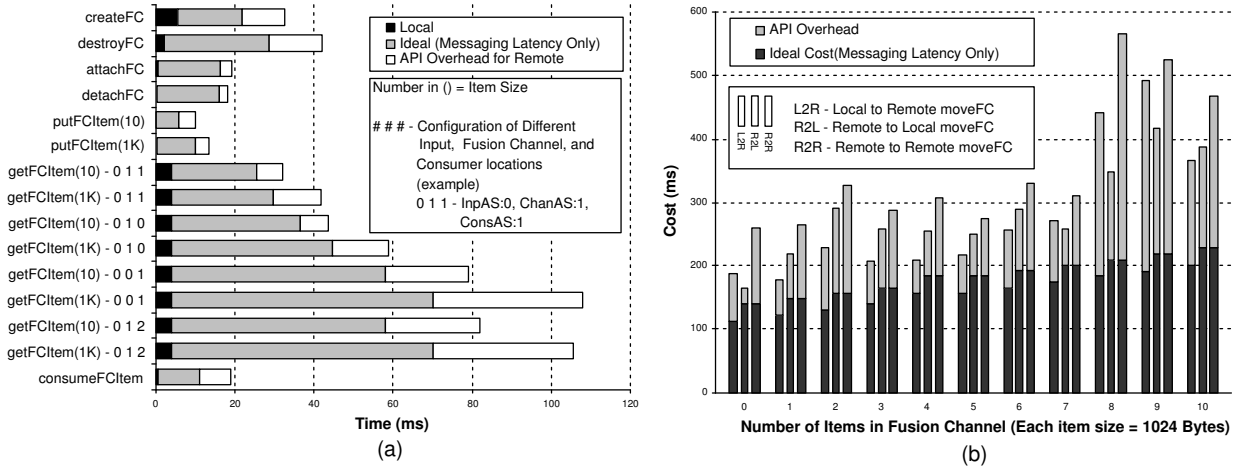


Figure 4. (a) Fusion Channel APIs' cost (b) Fusion channel migration (moveFC) cost

as a function of application-level bytes transmitted and node battery level. A primary design goal of our simulator is to incorporate more representative sources of energy usage in SN devices beyond the radio, such as the CPU and memory hardware. While still using simple models of application workload and device power models, we can begin to understand performance trends exposed by our middleware mechanisms and application contexts. Existing simulators for SN (TOSSIM [26], Em$*$ [13], Prowler [39] and GloMoSim [3]) or general wireless communication (ns2-wireless [7]) do not incorporate the new middleware mechanisms we focus our evaluation on, motivating the need for their extension or the construction of a new middleware simulator for our study. We have begun with building a middleware-only simulator to evaluate the fusion point migration and optimistic prefetching mechanisms under ideal MAC and routing layer assumptions for a variety of device and application models and scales.

Campus Surveillance Application Model

For generating a tunable application workload for our simulator, 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

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

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 1. For *FD/FR*, we use previously published time measurements [23] 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 1.

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. For these initial experiments, we assume that the demand at the sink is continuous, and that the various fusion functions perform statically as shown in Table 1, except when time-sharing a CPU or communication link with another fusion point. Future experiments will employ more dynamic and bursty workloads.

Collage and *Select* each 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. Figure 5a gives an expanded view of a composed task graph, showing how these two classes of functions are built into subgraphs that join to form a task graph. Our model assumes the cameras are on one side of campus, and fusion processing occurs in-network as data travels to the sink on the other side of the campus. The number of fusion operations is a function of the number of cameras in this model, enabling rapid construction of generic surveillance applications of different scales.

To construct the initial overlay network, we map the tree-like task graph's fusion points 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 overlay network. We disguise fusion point I/O mismatches by assigning the upstream

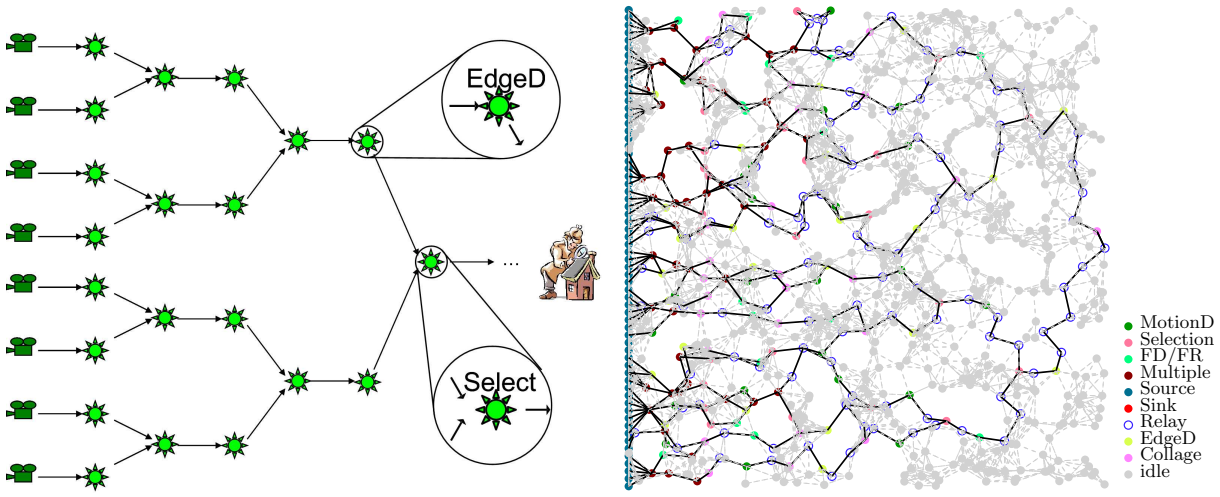


Figure 5. Campus-Wide Surveillance Application Simulation:
a) Task Graph Portion b) Sample Topology

function's output item size as the size of items transmitted through the relay chain to the downstream function.

Figure 5b depicts a sample overlay network 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 (common redundancy in SN). In addition to the cameras and sink, 800 SN nodes are randomly placed within the campus and ensured to be fully connected, initially. Darker lines indicate actively mapped links (relay chains). Some nodes host more than one fusion point simultaneously.

Power Models

Processor Power Model
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 [5] 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 [9]. Power measurement experiments on SA-1100 microprocessor indicate that power requirement increases monotonically with increase in clock frequency [42, 35]. Earlier research on SA-110 confirms the linear relationship too [30]. We use a linear model for energy consumption based on these two data points for determining energy usage at clock speeds from 59-206MHz.

Memory Power Model
Memory is also a major source of energy consumption, especially for memory-intensive workloads [42]. 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 [35].

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*.

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

Table 2. Radio power model.

Power consumption in these modes is as cited by Fan *et al.* [11] (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.

Communication Power Model
Radio is the communication medium in SN we consider, 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 2. We use numbers corresponding to two different bandwidths: one with an ORiNOCO network card [10], and another for a Bluetooth radio card [38]. 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. This is a common practice among today's motes, designed for sleeping over 99% of the time. 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 [2].

Simulator

We present here the event-driven simulator we have built to evaluate future sensor network deployments 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 SN as a collection of nodes and communication links, much as in Figure 5. 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

(we use **MPV** in these experiments). The current implementation supports upwards of 1000 simulated sensor network nodes, far beyond our capability to actually deploy for real-world experiments. 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. We use an ideal MAC layer that incurs neither energy nor latency overhead due to packet loss for these initial simulator experiments. 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.

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 is dying, 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, we employ "weak" migration, blocking and delaying until there are no items in-transit along any of the migrating point's input and output relay chains, and then transferring buffer function state associated with the fusion point to the target node. 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 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. We also assume a simplified fusion channel API, wherein fusion points only request the immediate next set of input items, performing "optimistic" prefetching by trying to keep their output buffer full.

4.4 Middleware Simulation Results

We have built this simulator for DFuse middleware with the ability to model over 1000 SN nodes, including power models for SN radio, CPU and memory, and the ability to model large applications (hundreds of fusion points), and including some large complexity for handling the synchronization and messaging necessary to perform migrations even under simplifying API, MAC and routing layer assumptions (similar to those employed in our initial DFuse implementation). We have performed preliminary experiments to shed light on the impact on application figures of merit of using combinations of middleware fusion point migration and optimistic prefetching features with varying device CPU speeds and radio characteristics: Bluetooth (*B/T*) vs ORiNOCO (*802*), and normal *Listen* cost vs ideal *Sleep* listen cost, for both our compute-intensive (*CPU*) and communication-intensive (*Comm*) application models [44]:

1. In the presence of optimistic prefetching, increasing the radio bandwidth may not improve latency nor throughput for compute-intensive workloads (Figure 7, *B/T-*-CPU* vs *802-*-CPU*). Also, network productivity may actually decrease if the change induces extra cost for idle nodes. For example, delivered items per lifetime de-

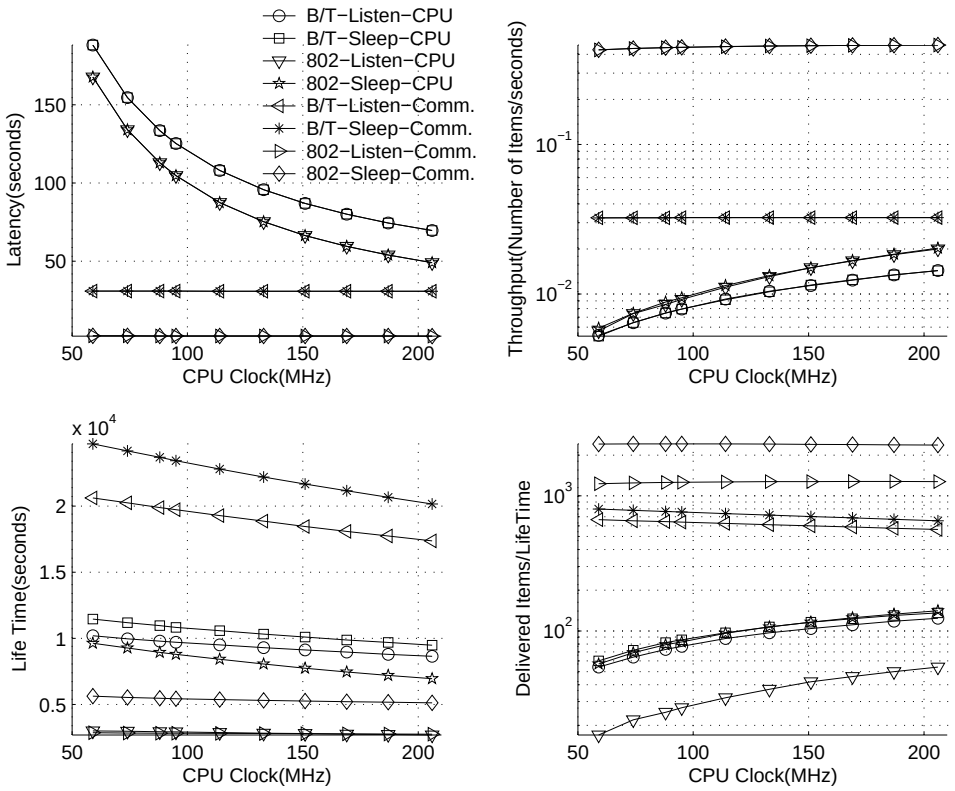


Figure 6. Baseline results: Migration and Prefetching Disabled

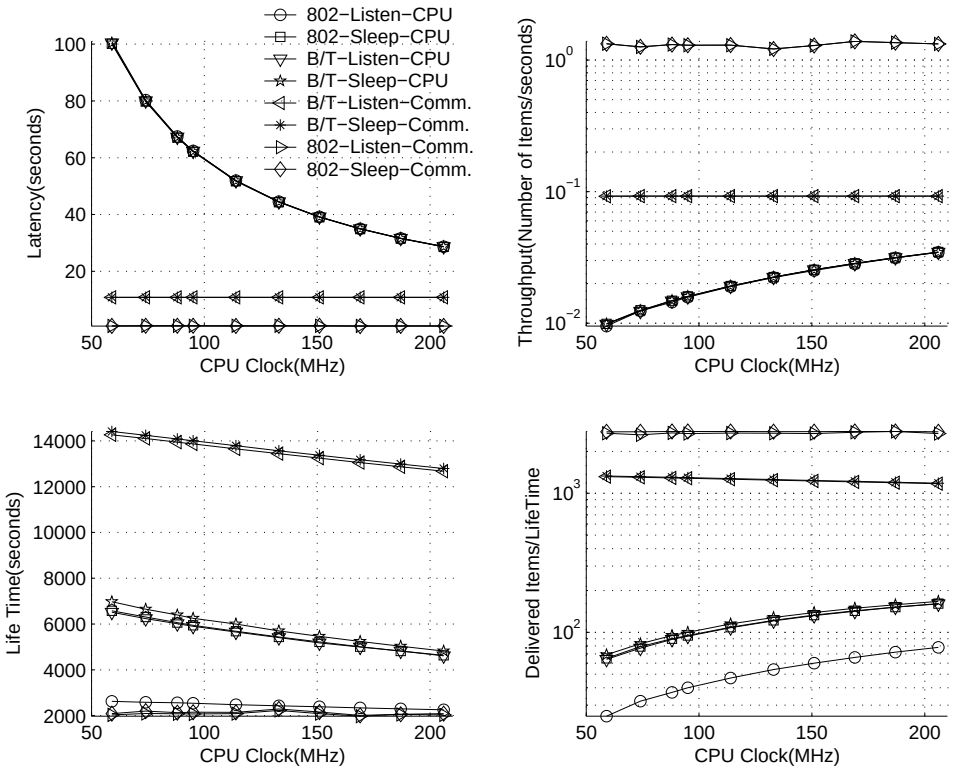


Figure 7. Results with Prefetching Enabled

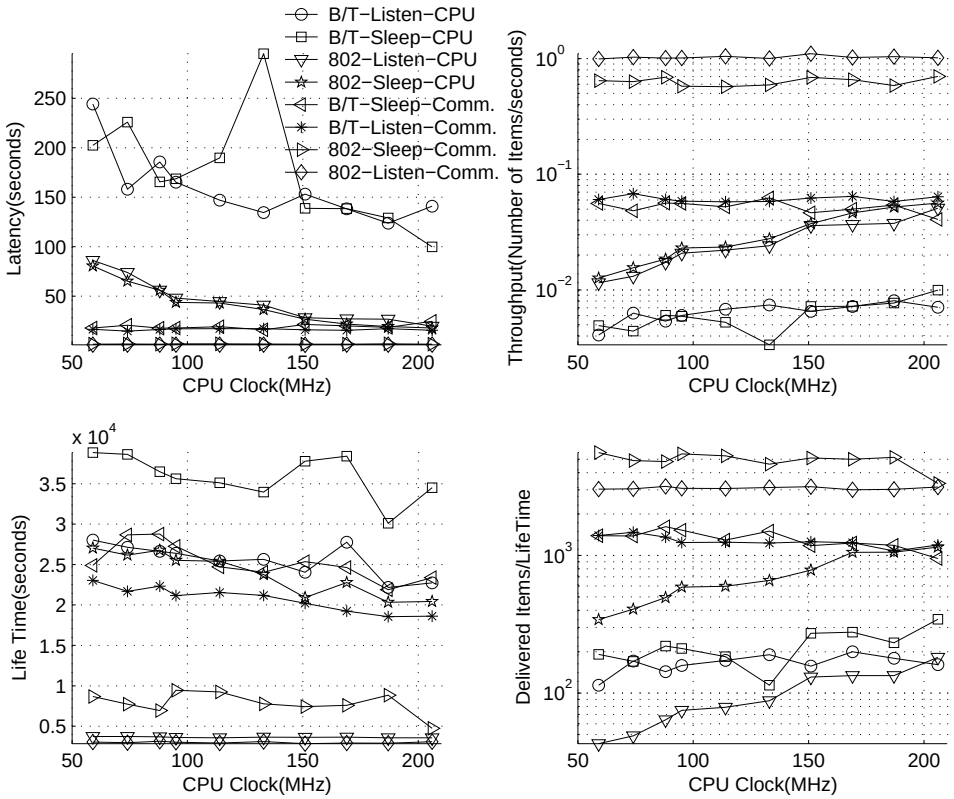


Figure 8. Results with Prefetching and Migration Enabled

creases when not using the ideal *Sleep* model and changing to a more expensive radio model in terms of listen cost (Figure 7, *B/T-Listen-CPU* vs *802-Listen-CPU*).

2. Cost function directed migration can significantly extend application lifetime in sensor networks with topologies and task graphs two orders of magnitude larger than previous studies: comparing Figure 8 to both Figures 6 and 7, lifetime is generally increased in all cases studied.

3. Compared to experiments with only prefetching enabled, turning on dynamic fusion point migration yields only slightly lower latency and throughput in most cases we study, while extending lifetime and increasing delivered items per lifetime (Figures 8 and 7). The exception, *B/T-*-CPU*, is encountered when frequently migrating larger state across a lower bandwidth connection. Although application lifetime is still extended, average latency and throughput may suffer, potentially leading to a drop in the total number of delivered items per lifetime. Suggested potential solutions to this specific problem would incorporate the latency cost of migration within the cost function being evaluated or in the determination of cost function evaluation frequency.

4. Although an optimal radio *Sleep* duty cycle is expected to improve application lifetime by not wasting energy in listen mode for idle nodes, it does not result in a significant change in lifetime in the presence of optimistic prefetching, except when using expensive ORiNOCO listen (Figure 7, **-Sleep-** vs **-Listen-**).

5. More intuitively, prefetching results in increased throughput compared to the baseline, while network lifetime with prefetching is lower than the baseline since more work is being done per unit time. (Compare Figure 7 to Figure 6.)

6. With prefetching enabled and migration directed by a battery variance minimizing cost function (**MPV**), we find that compute-intensive workloads on high bandwidth radios and high bandwidth CPUs may perform as well in

terms of throughput, latency, lifetime and delivered items per lifetime as communication-intensive workloads on low bandwidth radios with no significant dependence on CPU speed (Figure 8, *802-Sleep-CPU* at 206 MHz CPU Clock vs *B/T-*-Comm*). This result, while confirming the viability of our vision of future SN for supporting high bandwidth compute-intensive in-network processing, also indicates the potential for further studying the tradeoff between device capabilities, middleware features and application workload to help characterize the device and middleware features necessary for a particular level of application performance.

4.5 Middleware Scalability Results

We are currently performing experiments to determine how well each DFuse cost function for directing fusion point migration scales with respect to network topology and application size. Scalability is key to utility in real, large scale SN deployments. We are using our simulator, extended to include optimal cost "oracles" for each cost function, to analyze this scalability under simplifying API, MAC and routing layer assumptions and ignoring CPU and memory energy and delay costs as in our original DFuse evaluation. Both the migration and optimistic prefetching features are enabled in these experiments. We have discovered the following results so far:

1. As the network is scaled up to 1024 nodes for a single fusion point application, all three cost functions behave similarly with respect to each other in terms of transmission cost relative to the current optimal transmission cost: **MT2** performs close to optimal, followed by **MTP** and **MPV** performs worst.

2. For large topologies and small applications studied so far, we find that the energy of the neighbors of the fusion application's powered sources and sinks typically determines the lifetime of the application. In this case, there are so many redundant in-network nodes that the lifetime is limited by the fixed location of application endpoints (sources and sinks are assumed to not migrate).

3. A better evaluation of cost function performance is how it performs relative to the oracle for that cost function, not always the transmission cost oracle. For example, initial results indicate that **MPV** also performs very close to optimal in terms of distance from a mapping that would achieve minimum variance. Even this result is misleading, as it is for the single fusion point on a 1024 node SN, whose lifetime is already limited by the application endpoint's neighbor nodes: for the life of the application, there is always a completely unused node 1 hop away from the current mapping.

4. For **MTP** however, initial results indicate that the heuristic results in a mapping that is around 5 hops away from an optimal **MTP** mapping, rather consistently for this setup. This is likely due to the limitation of single-hop migration, along with a cost function that is simultaneously attempting to optimize for transmission cost as well as for hosting on a node with maximum power remaining. However, it remains to be seen whether using **MTP** will indeed limit the lifetime of the application, relative to **MT2**, and if it will not achieve as minimal battery variance as **MPV** for large scale applications not limited by application endpoints' neighbors.

5 Proposed Research

5.1 Completion of Current Middleware Scalability Evaluation

As we complete this current scalability study, for publication in a journal, we plan to test our hypothesis that for large topologies and *large* applications, we expect simulations to show similar rankings in terms of lifetime and battery variance as we find from our small scale DFuse implementation study. If this hypothesis does not hold true, there will need to be further analysis of these scalability results. One possible route would be to implement approximating

Steiner tree oracles to get a sense of how well the cost function heuristics work for larger applications relative to an approximate of optimum cost.

5.2 Accounting for Workload Dynamism and Non-Ideal Communication Channels

Once our basic fusion point migration and optimistic prefetching optimization mechanisms are evaluated under ideal assumptions, we plan to extend our middleware simulator to model a predictive prefetcher alternative to the current "optimistic" prefetcher, along with a dynamic, local CPU scaling mechanism.

Modeling Application Dynamism

Evaluation of these new mechanisms requires building appropriate application models that incorporate periodic and bursty behavior typical of distributed streaming applications. Possible approaches include fractal-based and Poisson process based workloads, outlined in Section 2.2, reusing our campus surveillance model to enable comparative analysis to existing results.

Leveraging an Existing Network Layer Simulator

We will then couple our middleware simulator to an existing wireless MAC and routing layer simulator, and use their combination to provide more realistic models of radio, MAC and routing layer overheads impacting the performance of our predictive prefetching and CPU scaling mechanisms. We plan to use GloMoSim [3] as the preexisting MAC and routing layer simulator for our work, as it affords larger scale network topologies than ns2-wireless [7], and it incorporates more general MAC and routing layer models than mote-specific models used for TOSSIM [26]/Em$*$ [13]. We anticipate some complexity in mapping our simulator's current messaging onto a network layer simulator. Specifically, some assumptions made in the current middleware simulator will need addressing:

1. The messaging needed to evaluate cost functions among one hop neighbors is currently assumed to be free by our simulator because it occurs infrequently relative to application workloads studied. The simulator currently immediately evaluates the cost function, leveraging a subset of its global knowledge of network state to keep from doing any messaging. However, such messaging will need to be implemented to incur the proper associated costs when collision or noise cause packet loss.

2. Similarly, the teardown and reconstruction of fusion point input and output relay chains is currently done instantaneously, once the simulated middleware is sure no application level messages are in flight on the chains. A protocol for remapping these chains will need to cooperate with the routing layer provided by the network layer simulator.

3. The middleware currently ignores the possibility of a node dying due to any reason other than being below an energy threshold. However, a faithful network layer simulator may cause premature node failure due to link characteristics. Effort may need to be expended to ensure that the middleware simulator can continue its current assumption.

Modeling and Evaluating Predictive Prefetching

Current DFuse mechanisms and evaluations do not account for the dynamic nature of application streaming, nor the lossiness of wireless communication channels in SN. SN fusion applications exhibit both bursty and periodic demands on network devices and may desire to "skip" over stream items to achieve greater currency. For example, to save energy and increase lifetime, a campus surveillance SN fusion application may perform minimal, infrequent anomaly

"detection" operations. Once a situation needing attention is detected, information gathering and processing activities will increase to achieve improved latency until the situation is resolved. Some portions of the distributed surveillance application may only require the most recently available inputs (*GetLatest*), enabling dropping of intermediately produced inputs, while other portions may require every input item in sequence (*GetNext*). An example of the former is a latency-critical in-network display showing remote video as close to realtime as possible, and an example of the latter is a stream decompressor that requires every input in sequence. Both of these application characteristics (burstiness and differing input semantics for a fusion operation) motivate the need for a predictive prefetcher to dynamically adapt which data items are requested for a particular computation, while hiding latency. Furthermore, even in non-mobile SN deployments we consider, wireless communication is lossy, and prefetching behavior needs to adapt accordingly to reduce misprediction latency and energy overheads.

There are many related approaches in distributed systems research for performing energy-adaptive communication management that will inform our predictive prefetcher design, highlights including: queuing data for future delivery in an application-driven manner for saving energy in mobile communication [22], and integration of wireless card sleep scheduling with CPU and network packet scheduling for energy savings [33]. Also, integration with machine learning approaches may yield performance benefits for some workloads. Finally, previous work in the domain of distributed stream processing for application task graphs, where the computations running at each task graph node inform a distributed algorithm for identifying which intermediately processed items are *dead*, requiring no further propagation nor computation [15], may be leveraged in our SN context for providing a lower bound for timestamps we prefetch. We feel there is significant opportunity for exploration of energy savings via middleware managed fusion application communication.

Leveraging Dynamic CPU Scaling

Our preliminary evaluations of DFuse include the assumptions that the SN is homogeneous, and the device capabilities remain constant for an application lifetime. As SN devices become more computationally capable and SN applications perform greater amounts of computation to process high bit-rate data, there emerges a significant increase in energy usage for computation relative to communication. For example, expensive computations such as face detection and recognition can now be done on sensor nodes. For such computations, our iPAQ-based microbenchmarks and power models [44] indicate that about 100ms of iPAQ processing is necessary on a data size of 56KB. Single hop communication to fetch inputs would cost roughly 106 mJ using ORiNOCO 11Mbps, while computation would cost roughly 31 mJ on a 206MHz SA-1100 package. Although communication costs in this case still exceed processing costs, there is a significant opportunity for reducing energy consumption by reducing processing costs. Also, if the data streams are compressed, then the proportional amount of energy used for processing increases. A benefit of a correct prefetch prediction is the knowledge *a priori* of the time at which the result of computation will be demanded by the application. Since power consumption of modern processors decreases as processor frequency and voltage decrease (see our simulator's power model), a middleware for supporting fusion applications would also include the ability to dynamically *scale* the CPU speed of individual SN nodes to reduce predicted application computation energy usage. We anticipate current technology trends enabling voltage and frequency scaling [35, 11, 32, 37] to be available in future SN devices.

We therefore propose to design, implement in our simulator framework, and evaluate a dynamic, local CPU scaling mechanism for fusion points that cooperates with a predictive prefetcher, informed by application semantic (*GetNext* vs *GetLatest*) and behavior to reduce the costs associated with fetching and fusing data never used and to reduce the costs associated with fusing data at a processor power level higher than necessary. This local CPU scaling mechanism will need to interact with potentially multiple local fusion points to collectively optimize node energy usage. Our

hypothesis is that this combination of additional mechanisms will indeed yield significantly greater energy savings and application lifetime, while still meeting application throughput and latency requirements. We will test this hypothesis through further simulation based evaluation.

6 Summary of Expected Contribution from this Thesis

By performing the remaining proposed work, we hope to arrive at the following insights:

1. We hope to understand how scalable our fusion point migration mechanism is in terms of both topology and application (task graph) size. An approximating Steiner tree oracle may be necessary to complete the current scalability evaluation for non-trivial task graphs.

2. We hope to clarify if, and by how much, predictive prefetching along with CPU scaling impact SN lifetime for bursty fusion application workloads using lossy wireless communication channels, and how this impact is changed when cost-function directed fusion point migration is enabled.

3. We hope to determine trends in terms of how common wireless network layers available in GloMoSim (link, MAC and routing) impact SN lifetime for these fusion application workloads and middleware mechanisms.

4. We hope to use these studies to be able to generate a model for how to provision a future SN in terms of node radios, MAC, routing layer, initial battery energy and CPU (memory has not proven to be critical to our applications' performance so far, and we do not anticipate doing finer grained models as a result) for a particular class of application workload, parameterized by application scale, "burstiness", input semantic (*GetLatest* vs *GetNext*), required lifetime, required throughput, and required latency. In further scalability studies, if we find that **MT2** does *not* achieve maximum lifetime or minimum latency, we will further need to adjust this model to include which cost function the middleware should use for a particular workload characteristic. This model may be partially incomplete. For example, we do not expect it to output the number of nodes, nor their topology. These will be assumed as inputs to the model. It will also be limited to the devices, layers and mechanisms we study in our simulator framework.

7 Broader Application

Our work is focused on future SN. However, it may be possible to adapt our mechanisms to target lower-bandwidth, lightweight computation capabilities of today's motes. Also, our research may well be applicable outside of SN. Contemporary laptops and handhelds are immediate sibling platforms for applications and supporting middleware mechanisms we study. General application-directed migration of computation may apply in grid computing and distributed media processing, to better achieve latency and throughput requirements, regardless of energy consumption. Furthermore, focused contributions, such as our expected combination of a predictive prefetcher with a dynamic CPU scaling mechanism, may well apply to more general distributed streaming contexts outside of SN.

8 Open Questions

As the design space for future SN devices, applications, and middleware for optimizing energy (lifetime) while meeting application latency and throughput requirements is vast, we are aware of several open research questions outside the scope of our proposed work:

1. We are not concerned with mechanisms for dynamically adapting the bandwidth, range and signal strength of SN radios, although this route of research may provide additional benefits to applications in terms of latency,

throughput and lifetime. It should be possible for later work to reuse our middleware simulator to characterize the potential benefits of such mechanisms, coupled with appropriate models of radio, MAC and routing layers. There is currently much conflicting research on whether multi-hop communication saves energy vs. "shouting louder", and varying application domains may have different trends here.

2. We constrain our study to supporting a single fusion application with a static task graph (in terms of data flow dependencies). In this work, we do not consider relaxations of this assumption including providing support for multiple applications and for applications whose task graphs are dynamic. While our mechanisms rely on virtualization of local device resources to manage timesharing required when multiple task graph fusion points are mapped to the same device, virtualization support for multiple applications is not our focus.

3. We do not propose new routing layers for power-aware, or more correctly, application-performance aware placement of relay nodes used to connect our overlay network. We will leverage available models for wireless ad hoc routing in SN in our evaluations. In addition, we will leverage routing layer support where available (*e.g.* energy characteristics of relay nodes) for fusion point placement decisions.

4. There is a need for coordinated control in SN. Our application models and middleware implementations and models do not focus on control. Rather, they are concerned with keeping up with demand by downstream consumers. For stream based fusion applications we consider, coordinated data streaming from multiple sources is a needed contribution.

5. The design space greatly expands when mobility of sources and sinks, and general mobility of SN nodes is introduced. There are opportunities for leveraging such mobility for energy savings through radio power scaling and message ferrying, recharging batteries, and for increasing application throughput and latency by dynamically positioning resources more optimally. We do not consider mobility-based approaches for optimization in this work.

6. This proposal does not include plans to implement these middleware mechanisms in libraries usable on real SN devices. Although our initial DFuse prototype was evaluated on iPAQs, the proposed extensions and evaluations are based on a flexible set of device models and large scales. We do not have the resources to support an actual deployment of the existing and proposed mechanisms for large scale applications. Our focus is instead on the evaluation of performance tradeoffs for a promising set of future SN devices, applications, and middleware mechanisms.

7. We do not consider device failures other than for reasons of lack of energy. One potential incremental approach for addressing this is to create redundant fusion points in the network, creating an energy vs availability tradeoff. Other approaches in SN domain [4] have considered a similar tradeoff: energy vs accuracy. Our simulated middleware assumes that delayed transmission is due to lossy wireless channels, rather than a prematurely dead node. Another approach would be to implement *partial fusion*, the ability of a fusion operation to commence with partial inputs after some timeout or exception.

References

[1] Sameer Adhikari, Arnab Paul, and Umakishore Ramachandran. D-Stampede: distributed programming system for ubiquitous computing. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, Vienna, July 2002.

[2] Yuvraj Agarwal and Rajesh K. Gupta. On Demand Paging Using Bluetooth Radios on 802.11 Based Networks. Technical Report 03-22, Center for Embedded Computer Systems, UC Irvine, UC San Diego, July 2003.

[3] Lokesh Bajaj, Mineo Takai, Rajat Ahuja, Ken Tang, Rajive Bagrodia, and Mario Gerla. GloMoSim: a scalable network simulation environment. UCLA Computer Science Department Technical Report 990027, May 1999.

[4] Athanassios Boulis, Chih-Chieh Han, and Mani B. Srivastava. Design and implementation of a framework for programmable and efficient sensor networks. In *Proceedings of the First International Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Francisco, CA, May 2003.

[5] Thomas D. Burd and Robert W. Brodersen. Processor design for portable systems. *Journal of VLSI Signal Processing*, 13(2-3):203–222, August 1996.

[6] Jae-Hwan Chang and Leandros Tassiulas. Energy conserving routing in wireless ad-hoc networks. In *IEEE INFOCOM*, pages 22–31, 2000.

[7] CMU Monarch Project. Wireless and mobility extensions to ns-2. Available November 2004 at *http://www.monarch.cs.cmu.edu/cmu-ns.html*, 1999.

[8] Intel Corp. New computing frontiers - the wireless vineyard. Available November 2004 at *http://www.intel.com/labs/features/rs01031.htm*.

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

[10] Proxim Corp. ORiNOCO PC Card Specification. 2003 available at *http://www.hyperlinktech.com/web/orinoco/orinoco_pc_card_spec.html*, similar spec available November 2004 at *http://www.proxim.com/learn/library/datasheets/11bpccard.pdf*.

[11] Xiaobo Fan, Carla Ellis, and Alvin Lebeck. Memory controller policies for dram power management. In *Proceedings of the 2001 international symposium on Low power electronics and design*, pages 129–134, Huntington Beach, California, United States, 2001. ACM Press.

[12] Deepak Ganesan, Bhaskar Krishnamachari, Alec Woo, David Culler, Deborah Estrin, and Stephen Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. Technical Report CSD-TR 02-0013, UCLA, February 2002. Available November 2004 at *http://www.cs.umass.edu/~dganesan/PAPERS/empirical.pdf*.

[13] Lewis Girod, Jeremy Elson, Alberto Cerpa, Thanos Stathopoulos, Nithya Ramanathan, and Deborah Estrin. Em*: a software environment for developing and deploying wireless sensor networks. In *Proceedings of USENIX 04*, Los Angeles, California, USA, 2004.

[14] Chao Gui and Prasant Mohapatra. Sensor networks: Power conservation and quality of surveillance in target tracking sensor networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, September 2004.

[15] Nissim Harel, Hasnain A. Mandviwala, Kath Knobe, and Umakishore Ramachandran. Dead timestamp identification in stampede. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, pages 101–108, 2002.

[16] Wendi B. Heinzelman, Amy L. Murphy, Hervaldo S. Carvalho, and Mark A. Perillo. Middleware to support sensor network applications. *IEEE Network Mag.*, 18(1):6–14, 2004.

[17] Crossbow Technology Inc. MICA2 datasheet. Available November 2004 at *http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0042-06_A_MICA2.pdf*.

[18] Crossbow Technology Inc. Stargate gateway (SPB400) datasheet. Available November 2004 at *http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0049-01_C_Stargate.pdf*.

[19] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[20] Ralph M. Kling. Intel mote: An enhanced sensor network node. In *Proceedings of the International Workshop on Advanced Sensors, Structural Health Monitoring, and Smart Structures*, 2003.

[21] Ulas C. Kozat, Iordanis Koutsopoulos, and Leandros Tassiulas. A framework for cross-layer design of energy-efficient communication with qos provisioning in multi-hop wireless networks. In *Proceedings of IEEE/Infocom*, 2004.

[22] Robin Kravets and P. Krishnan. Application-driven power management for mobile communication. *Wireless Networks*, 6(4):263–277, 2000.

[23] Ulrich Kremer, Jamey Hicks, and James M. Rehg. A compilation framework for power and energy management on mobile computers. In *Proceedings of the International Workshop on Languages and Compilers for Parallel Computing (LCPC)*, August 2001.

[24] Rajnish Kumar, Matthew Wolenetz, Bikash Agarwalla, JunSuk Shin, Phillip Hutto, Arnab Paul, and Umakishore Ramachandran. DFuse: a framework for distributed data fusion. In *Proceedings of the first international conference on embedded networked sensor systems*, pages 114–125, Los Angeles, California, USA, 2003. ACM Press.

[25] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of ethernet traffic. In *Conference proceedings on communications architectures, protocols and applications*, pages 183–193, San Francisco, California, USA, September 1993.

[26] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[27] Alan Mainwaring, Joseph Polastre, Robsert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications*, 2002. Also Intel Research, IRB-TR-02-006, June 2002.

[28] Daniel A. Menasce, Bruno D. Abrah ao, Daniel Barbara, Virgilio A. F. Almeida, and Flavia P. Ribeiro. Fractal characterization of web workloads. In *Proceedings of the 11th International World Wide Web Conference (www2002)*, Honolulu, Hawaii, USA, 2002.

[29] Martin Modahl, Ilya Bagrak, Matthew Wolenetz, Ramesh Jain, and Umakishore Ramachandran. EventWeb: Distributed media correlation, analysis and distribution framework. In *Proceedings of 10th IEEE Workshop on the Future Trends of Distributed Computing Systems (FTDCS-04)*, Suzhou, China, May 2004.

[30] James Montanaro, Richard T. Witek, Krishna Anne, Andrew J. Black, Elizabeth M. Cooper, Daniel W. Dobberpuhl, Paul M. Donahue, Jim Eno, Gregory W. Hoeppner, David Kruckemyer, Thomas H. Lee, Peter C. M. Lin, Liam Madden, Daniel Murray, Mark H. Pearce, Sribalan Santhanam, Kathryn J. Snyder, Ray Stephany, and Stephen C. Thierauf. A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor. *Digital Tech. J.*, 9(1):49–62, 1997.

[31] Kihong Park. On the effect and control of self-similar network traffic: a simulation perspective. In *Proceedings of the 29th conference on Winter simulation*, Atlanta, Georgia, USA, 1997.

[32] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM Symposium on Operating Systems Principles*, pages 89–102, 2001.

[33] Christian Poellabauer and Karsten Schwan. Energy-aware traffic shaping for wireless real-time applications. In *Proceedings of the 10th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, May 2004.

[34] Joseph Polastre, Robert Szewczyk, Cory Sharp, and David Culler. The mote revolution: Low power wireless sensor network devices. In *Proceedings of Hot Chips 16: A Symposium on High Performance Chips*, 2004. Presentation available November 2004 at *http://webs.cs.berkeley.edu/papers/hotchips-2004-motes.ppt*.

[35] 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)*, pages 251–259, Rome, Italy, July 2001.

[36] Umakishore Ramachandran, Rishiyur S. Nikhil, Nissim Harel, James M. Rehg, and Kathleen Knobe. Space-time memory: A parallel programming abstraction for interactive multimedia applications. In *Principles Practice of Parallel Programming*, pages 183–192, 1999.

[37] Greg Semeraro, David H. Albonesi, Steven G. Dropsho, Grigorios Magklis, Sandhya Dwarkadas, and Michael 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*, pages 356–367, Istanbul, Turkey, 2002. IEEE Computer Society Press.

[38] OKI Semiconductor. ML7050LA Specification. Available November 2004 at *http://www.oki.com/semi/english/t-blue.htm*, June 2001.

[39] Gyula Simon, Peter Volgyesi, Miklos Maroti, and Akos Ledeczi. Simulation-based optimization of communication protocols for large-scale wireless sensor networks. In *Proceedings of IEEE Aerospace Conference*, Nashville, Tennessee, USA, March 2003.

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

[41] Suresh Singh, Mike Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Mobile Computing and Networking*, pages 181–190, 1998.

[42] Marc A. Viredaz and Deborah A. Wallach. Power evaluation of a handheld computer. *IEEE Micro*, 2003.

[43] Max Wang. Nokia sees strong demand for smartphones and camera phones in 2005. Available November 2004 at *http://www.digitimes.com/news/a20041104A6035.html*.

[44] Matthew Wolenetz, Rajnish Kumar, Junsuk Shin, and Umakishore Ramachandran. Middleware Guidelines for Future Sensor Networks. In *Proceedings of the First Workshop on Broadband Advanced Sensor Networks*, San Jose, California, USA, October 2004.

[45] Wei Ye, John Heidemann, and Deborah Estrin. An Energy-Efficient MAC protocol for Wireless Sensor Networks. In *Proceedings of INFOCOM 2002*, New York, New York, June 2002.