Algorithm Design and Synthesis for Wireless Sensor Networks*

Amol Bakshi and Viktor K. Prasanna
Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089
{amol, prasanna}@usc.edu

Abstract

Most of current research in wireless networked embedded sensing approaches the problem of application design as one of manually customizing network protocols. The design complexity and required expertise make this unsuitable for increasingly complex sensor network systems. We address this problem from a parallel and distributed systems perspective and propose a methodology that enables domain experts to design, analyze, and synthesize sensor network applications without requiring a knowledge of implementation details. At the core of our methodology is a virtual architecture for a class of sensor networks that hides enough system details to relieve programmers of the burden of managing low-level control and coordination, and provides algorithm designers with a clean topology and cost model. We illustrate this methodology using a real-world topographic querying application as a case study.

1. Introduction

Wireless sensor networks (WSNs) are ad hoc networks of unattended smart sensors performing in-network collaborative computation and communication to monitor the environment for events of interest. Most of the current research on information processing in WSNs has focused on mostly manual optimization and application-specific customization of the network protocol stack. While this approach has been successful for relatively simple application scenarios, there is now an increasing realization of the need for new programming models and abstractions for algorithm designers and programmers that do not demand an expertise in wireless networking - in addition to a knowledge of the application domain [4, 12].

In this paper, we explore a methodology for algorithm design and synthesis for a class of sensor network applications, with the goal of reducing design complexity. Our methodology enables a domain expert to design and analyze algorithms, and synthesize programs for the virtual architecture, without requiring a knowledge of low level networking aspects of the deployment. While WSNs are approached from a wireless ad hoc networking perspective in most of state of the art, we model them as parallel and distributed systems. However, there are important differences between sensor networks and traditional distributed systems that should be considered while defining models of computation and algorithm design methodologies for WSNs.

Sensor networks are data driven in the sense that data is created in the network (by the sensing interfaces) and the values of sensor data and the application semantics determine the pattern of computation and communication at run time. Also, the computation has to be performed as close to the data as possible, in order to reduce communication energy consumption in the network. This means that the (re)distribution of data and computation to nodes of the network is subject to constraints that arise from application semantics and performance requirements. In traditional parallel and distributed processing, latency and throughput of execution have been the major performance metrics, and the overall objective is to manage the resources efficiently so as to minimize execution time of the computation. In embedded systems such as sensor networks, the application essentially executes in an infinite loop, and the concept of a round of execution is ill defined in many scenarios due to their data driven behavior. Hence, at the system level, minimizing energy consumption of the network as a whole is the dominant concern, sometimes even at the expense of increased latency of some path of execution.

The rest of this paper is organized as follows. Section 2 presents our overall design methodology. Sections 3 and 4 illustrate the process of defining a virtual architecture, algorithm design, and (manual) program synthesis for our case study. Section 5 describes a set of protocols to implement the modeling abstractions at run time. Related work is discussed in Section 6, and we conclude in Section 7.

* This work is supported by the National Science Foundation under award number IIS-0330445.

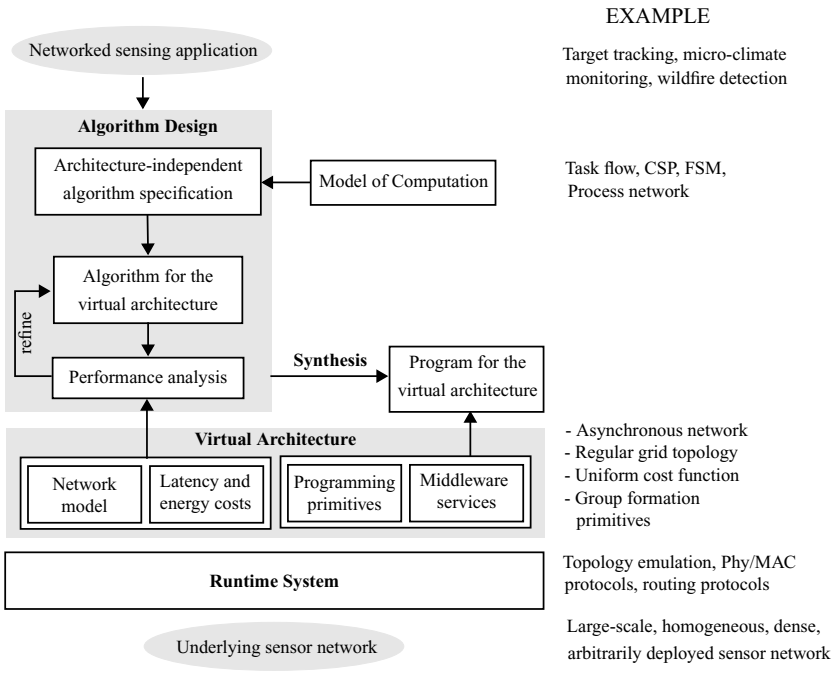

Figure 1: Our Proposed Design Methodology

2. Our Proposed Methodology

Our methodology is based on a virtual architecture for the sensor network that enables algorithm design and synthesis. A virtual architecture is an abstract machine model for algorithm design and synthesis and a set of primitives that are independent of low level protocols used to implement them in the underlying network. It is important for the end user that the modeling abstractions correspond to mental notions of application behavior on a sensor network system, that a formalism exists to specify such behavior, and that some mechanism exists to map the behavior onto the underlying network such that theoretical performance analysis corresponds to real performance measurements.

Defining a Virtual Architecture: The virtual architecture should: (i) facilitate rapid first-order performance estimation of algorithms, (ii) provide suitable primitives to enable translation of the selected algorithm into a program for the underlying network, and (iii) export appropriate middleware services that allow the end user to think in terms of abstract logical entities such as events of a specific type. Key components of our virtual architecture are:

Network model: The network model specifies the topology of the deployment that can be assumed at design time. This (virtual) topology can be emulated on the real network deployment in a variety of ways that could be hidden from the algorithm designer. The choice of virtual topology will be influenced by the expected nature of the deployment, and also by the nature of collaborative computation and collaboration in the target application. Depending on the type of network, the model could support synchronous algorithms (e.g., TDMA), purely asynchronous message-passing paradigms, or a combination of the two [8].

Programming primitives: The virtual architecture specifies the computation and communication primitives available to the programmer. These primitives could be for the individual node or for a set of nodes (collective). Communication primitives could range from the simple send() and receive() message passing primitives to more sophisticated ones for group communication. Computation primitives could include summing, sorting, or ranking a set of data values from a set of sensor nodes [5]. The implementation of these primitives could be transparent to the end user who is aware only of their functionality and associated costs.

Middleware services: A middleware service implements a high-level functionality that is commonly needed for a certain class of applications [20]. For example, collaborative computation through the formation of logical groups is a very useful concept for a large number of WSN applications, allowing the end user to reason in terms of a follower-leader relationship between nodes in a group. Middleware services are essential in decoupling high-level functional abstractions from their implementation details, and hence are part of our virtual architecture.

Cost functions and performance metrics: Computation and communication costs in terms of time and energy are spec-

ified for each primitive, as well as for the operations supported by the middleware services. These cost functions, combined with the application representation should provide sufficient information to decide an efficient mapping of application tasks onto sensor nodes and for communication synthesis. The performance metric to be used for evaluating an algorithm will be based on these cost functions, but will depend on the application. For example, total energy, energy balance, total latency of a set of operations, system lifetime, etc., are various performance metrics that can be calculated from the cost model, but which of these to use will depend on the algorithm designer's objective.

Design Flow: In Figure 1, the graphical depiction of the major components of our methodology and the design flow is annotated with concrete examples for some of the abstractions. First, the network model and cost model for the target class of sensor networks is defined using a **bottom-up** approach that involves analysis of the expected nature of deployment, characteristics of the node hardware, etc. The end user analyzes alternate algorithms for the desired functionality and selects one with the best performance for that network and cost model. For example, the end user could decide if a divide and conquer approach is better than a centralized approach if, say, total latency of one round of the application is to be minimized. After the algorithm is chosen, a **top-down** approach is required to convert (synthesize) it into the program that executes on each node of the network. In the top-down approach, the algorithm is specified using an architecture-independent application model such as an annotated task graph. The application graph is used as an input to a mapping tool (manual or automatic) that will explore alternate design time and run time role assignments in the network and determine an efficient one. When the roles are assigned (i.e., the tasks are mapped), the actual software has to be synthesized for each node. The structure of the task graph and explicit annotations by the application developer are used to determine which of the available middleware services (if any) are useful. For instance, in a task graph structured as a k-ary tree, the interaction between every parent node and its k children can be implemented using a middleware API for group communication if one is available. Further, if logical naming service is supported, the group membership can even be determined at run time. Finally, the program for each node is synthesized, including the appropriate calls to middleware.

3. Case Study: Identification and Labeling of Homogeneous Regions

The purpose of this case study is to demonstrate the definition and use of a virtual architecture for an example application scenario. For the problem described in the next subsection, we choose a divide-and-conquer algorithm for in-network merging of boundary information. This algorithm is represented as a high-level task graph. Using the communication primitives and cost models offered by the virtual architecture, the performance of this algorithm can be analytically estimated. Then we illustrate how this high-level specification is (manually) synthesized into an algorithm for the virtual architecture. This synthesis includes mapping logical entities in the high-level representation to nodes in the (virtual) topology, while satisfying some constraints on the mapping process. We then demonstrate how the architecture-specific algorithm is expressed as a program that executes on the individual node of the (virtual) topology. Finally, in Section 5, we present some protocols that could be used to to implement the modeling abstractions, with a view to preserving the correspondence between the theoretical performance analysis and the actual performance on the underlying network. These protocols are representative of the approaches that can be adopted to implement a virtual architecture.

3.1. The Application

Topographic querying is the process of extracting data from a sensor network for understanding the graphical delineation of features of interest in the environment. The end user might be interested in visualizing gradients of sensor readings across the region or other queries such as enumeration of regions with sensor readings in a specific range. Application areas where this is useful range from contaminant monitoring to HVAC (heating, ventilation, and air conditioning) applications. Boundary estimation and counting regions of interest can also be used in acoustic monitoring and tracking applications. Querying the properties of sensor node such as residual energy levels is useful for resource management, dynamic retasking, preventive maintenance of sensor fields, etc.

A basic operation that supports a large class of topographic queries is the identification and labeling of *homogeneous regions*. A homogeneous region (or *feature region*) is one where all sensors have the same reading of a phenomenon. Sensor nodes whose readings are of interest to a particular query are called *feature nodes*. For simplicity we assume that a sensor node has a binary status (feature node or not a feature node) for the query. Once this information is gathered and stored in the network, other queries can be answered. For example, a query to count the number of regions of interest can obtain and sum the local counts of each of the distributed storage nodes. Processing and responding to queries could be in most cases decoupled from the actual data gathering and boundary estimation process, which can occur independently.

3.2. Our Virtual Architecture

We define the following virtual architecture for algorithm design and synthesis on large scale, homogeneous sensor networks that are arbitrarily and densely deployed on a terrain for purposes of environment monitoring applications such as topographic querying described above.

The network: In our application scenario, the end user is interested in monitoring the temperature over the entire terrain with a certain granularity. In other words, the set of locations - or points of coverage (PoCs) - are uniformly distributed over the entire terrain. For a rectangular terrain, the PoCs form a grid with a certain 'cell size' corresponding to each PoC. Note that the locations of the PoCs (in this case, in a grid) is not necessarily related to the pattern of deployment. As long as there is at least one sensor node in each geographic cell, the topology emulation algorithm (Sec. 5) is responsible for overlaying the virtual grid topology over the possibly irregular topology of the underlying network. Other topology creation and maintenance algorithms such as the one proposed in [17] can also be employed.

A grid will be an appropriate choice of virtual topology for uniform node deployment over the terrain. For non-uniform deployments, other virtual topologies such as a tree could be more appropriate. For the purposes of this case study, our virtual architecture in this case study abstracts the underlying network topology as an *oriented, two-dimensional grid*.

Middleware services: The concept of a group is central to networked sensing applications. Most in-network collaborative computation is accomplished by temporarily or permanently organizing sensor nodes in terms of groups. The membership in a group can be determined based on different factors such as geographic location, current reading of a sensor, the functionality of the program running on a node, etc. Geographic groups are ones where all nodes that are deployed in a certain geographic region are members of the group. Our virtual architecture incorporates a group formation middleware service specifically tailored for our case study. In a general application scenario, this service can be implemented using a combination of geographically constrained groups and logical naming, but for simplicity of exposition, the service is defined as follows.

The concept of hierarchical groups is supported for the grid topology. At the lowest level of hierarchy (level 0), every node is both a group member and a group leader. At level 1, the grid is partitioned into blocks of 2×2 nodes. The node in the north-west corner is designated a level 1 leader, and remaining nodes of the block are level 1 followers, and so on. Since every node knows its own grid coordinates, it can also determine its role as leader and/or follower at each level of the hierarchy.

Communication primitives: The virtual architecture in this case study supports *send()* and *receive()* message passing primitives that a node can use to communicate with any other node in the network. A *group communication primitive* is also available that can be used by a node to directly address a level k leader as a logical entity.

Cost functions: We assume that each node has a short-range, omnidirectional antenna. For such antennas, the reception and transmission energy is of similar magnitude, and depends only on the radio electronics [13]. A *uniform cost function* for energy and time analysis of these systems can be defined – the energy cost for transmission, reception or computation of one unit of data is defined to be one unit of energy. One unit of latency is the time taken to complete p computations or transmit b units of data, where p and b are the processing speed and transmission bandwidth of the node respectively. This simple cost model has been used in most of the recent work related to algorithm design for sensor networks [5, 14, 18]. Whether these cost functions are realistic for a specific network is a decision for the end user. A different set of cost functions can be used if the characteristics of the deployment necessitate it.

4. Algorithm Design and Synthesis

4.1. Algorithm Specification

Our starting point is an algorithm for topographic querying that runs in $O(\sqrt{v})$ steps for a $\sqrt{v} \times \sqrt{v}$ grid, by using a divide and conquer strategy [3]. A step denotes a round of computation and is used for convenience of analysis. No assumptions are made about the degree of synchronization in the network. This algorithm can be represented as a data flow graph structured as a quad-tree (Figure 2). A leaf node corresponds to a task that is linked to the sensing interface, and interior nodes represent in-network processing on the sampled data. At each level of the tree, every node transmits its information to its parent at the next higher level. Processes at higher levels have greater oversight in terms of the extent of the regions they represent. In terms of groups, nodes at higher levels of the quad-tree are group leaders whose group members are its children in the tree.

A leaf node can compute its status as a feature node by comparing its current reading with a pre-specified threshold. At each level of hierarchy, a node receives data from its four children, containing a description of the boundaries of feature regions contained within the sender's geographic oversight. The boundary information also indicates whether the feature region(s) lie entirely within that extent, or information from neighboring extents is required to identify the true boundary of the feature region.

Again, the choice of application model will depend on the deployment scenario. In this case study, a task graph rep-

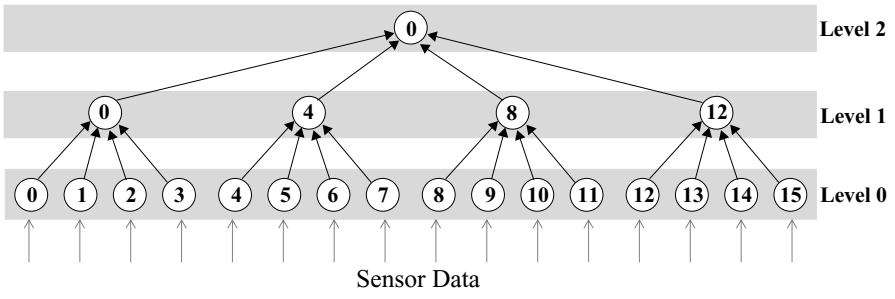

Figure 2: Quad-tree representation of the algorithm

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Figure 3: Example mapping

resentation is an appropriate model because we assume that the pattern of computation and communication is known at design time. Leaf nodes sample at a known frequency, and every 'round' of sampling triggers one execution of the entire task graph. This model might not be suitable for event-driven applications such as target tracking where only the sensor nodes in the vicinity of the target (event) perform the sampling and in-network collaborative signal processing. If a task graph model has to be used for this scenario, the frequency of sampling at the leaf nodes could be expressed in probabilistic terms derived from a knowledge of expected events in the network.

The semantics of our application impose the following design time constraints on the task-to-node mapping.

Coverage: Each node in the virtual topology corresponds to one point of interest in the terrain and accordingly, the number of leaf nodes in the task graph is equal to the number of nodes in the virtual network graph. The coverage constraint states that each leaf node of the task graph (that represents one sampling task) should be mapped to a distinct node of the virtual topology to ensure the desired level of coverage.

Spatial correlation: In our application, the data exchanged between nodes represents boundaries of feature regions. When this data is merged at the parent nodes, the algorithm expects that the information from child nodes represents spatially adjacent geographic extents. That way, maximum data compression can be achieved in terms of representing the results of the processing. Hence, the spatial correlation constraint states that all children of a given node should represent information about a single contiguous geographic extent.

4.2. Role assignment

The virtual topology, cost model, and application graph can be provided as input to any of the numerous task mapping algorithms that exist in literature [6]. Since energy is an important consideration for sensor networks, the optimization criteria for the chosen algorithm will have to reflect new performance metrics such as total energy and/or energy balance. Also, for the mapping to be feasible, constraints such as coverage and spatial correlation will have to be satisfied.

Our virtual architecture exports a grid topology, and a mapping that satisfies both constraints is shown by the labeling of the quad-tree nodes in Fig. 2, which indicates their mapping to the regions of the grid in Fig. 3. As shown in the figures, the terrain of deployment is partitioned into 2×2 blocks and groups of four level 0 nodes of the quad-tree that share the same parent are mapped onto each block.

Only the leaf nodes perform the actual sampling. Hence, the non-leaf nodes can be mapped anywhere in the grid subject to performance optimization. In general, the algorithm designer can express the *relationship between a parent and child as a leader and follower in a collaborative group*, and leave their mapping entirely to the group formation middleware service of the virtual architecture. Evaluating the relative performance of this algorithm compared to some other approach means that the middleware should provide the associated cost for member to leader communication within the group. With our group formation technique described in Sec. 3.2, the latency and energy of transmitting a data packet from a level k follower to the level k leader is proportional to the minimum number of hops separating them in the virtual network graph, assuming shortest path routing. We do not provide a detailed latency and energy analysis in this paper due to space limitations. Interested readers can refer to [16] for a detailed analysis and high-level simulation results of this algorithm on a sensor network architecture very similar to the algorithm described above.

Using the static group formation provided by the virtual architecture, the mapping that will finally occur is shown by the labels of non-leaf nodes in Fig. 2 and corresponding region labels of the grid in Fig. 3. The root node is mapped to location 0, and the four level 1 nodes are mapped to locations 0, 4, 8, and 12 respectively, which are the leaders of their corresponding groups. The mapping exploits the correspondence between the quad-tree structure, and the idea of recursively dividing the topology into quadrants and merging data within quadrants.

4.3. Program Synthesis

The output of the mapping stage is an algorithm specified for a grid topology, which relies on middleware support for group formation, and on communication primitives that can be used by any node to send a message to group leaders at the appropriate level of hierarchy. The next step is to synthesize this algorithm into a program that executes at each node of the grid topology.

We use a reactive, event-driven programming model that is supported by state-of-the-art code generation frameworks [8] and programming languages [10] for sensor networks. An asynchronous data flow model of computation is assumed, which means that a process need not wait for all its input data (incoming messages) before computing on them. This is because latency of message delivery is unpredictable in typical sensor networks and some messages might even be dropped. Therefore, it is advisable to structure the program in such a way that incoming information is incrementally processed wherever possible. In our application, since the information represents region boundaries, it can be incrementally merged into the existing aggregated information at that leader.

A manually synthesized program specification for the algorithm is given in Figure 4. A brief description of the working of this program is as follows. Initially ($start = true$), each node decides if it is a feature node and constructs its local data structure to store the information. It then increments the level of recursion (hierarchy) and transmits the information to the appropriate leader node for merging. For all levels higher than 0, a node can expect to receive 3 messages from group leaders at the next lower level of recursion since the network is partitioned into quadrants at every stage and by virtue of the mapping, one of the four incoming messages in the quad-tree representation is from the node to itself. An array is used for the $msgsReceived$ data structure in consideration of the fact that information can be transmitted and processed at different speed in each quadrant.

In a grid of size $\sqrt{v} \times \sqrt{v}$ where $\log \sqrt{v}$ is an integer, all level k leaders are also level $(k-1)$ leaders. A level k leader can receive messages from other level $(k-1)$ leaders before it completes processing messages from level $(k-2)$ leaders in its own quadrant. Hence, messages contain their level information, and on receipt are merged with other messages at that level of recursion. Once a level k leader that is not also a level $(k+1)$ leader sends a message to a level $(k+1)$ leader, it no longer participates in the aggregation process and its level of recursion does not increase. Only the node which performs the final aggregation reaches $maxrecLevel$, triggers the corresponding Action clause, and exfiltrates the entire boundary information (or stores it locally, depending on the end user requirements).

State (initial values) :
start(=*false*), recLevel(=0), maxrecLevel,
mySubGraph[1..maxrecLevel](=NULL),
myCoords, msgsReceived[1..maxrecLevel](=0)
transmit(=*false*)

Message alphabet :
mGraph = {senderCoord, msubGraph, mrecLevel}

Condition : start = true
Action: start = false
compute mySubGraph[recLevel]
from intra-cell readings
transmit = true
recLevel = recLevel + 1

Condition : received mGraph
Action: merge(mGraph,mySubGraph[mrecLevel])
msgsReceived[mrecLevel]++

Condition : transmit=true
Action: message = {myCoords, mySubGraph, recLevel}
if (recLevel = maxrecLevel)
exfiltrate message
else
send message to Leader(recLevel+1)
transmit = false

Condition : msgsReceived[recLevel] = 3
Action: transmit=true
recLevel = recLevel + 1

Figure 4: Synthesized program specification

5. The Runtime System

We now briefly describe some protocols for the runtime system designed to accomplish two main functionalities: emulating the grid topology on the arbitrary network deployment, and binding virtual processes of the synthesized program to real nodes of the underlying network.

5.1. Topology Emulation

The underlying network consists of n identical sensor nodes deployed over a square terrain of side l. Let $G_g = (V_g, E_g)$ denote the virtual network graph (grid). The terrain can be partitioned into non-overlapping equal sized cells each of side c - such that $\frac{l}{c} = \sqrt{v}$, where $v = |V_g|$. Each sensor node has a transmission range of r. Let $SN = \{s_0, s_1, \ldots, s_n\}$ be the set of sensor nodes. The real network can therefore be represented by a graph $G_r = (V_r, E_r)$, where vertices correspond to sensor nodes, and $(i, j) \in E_r$ iff $\delta(s_i, s_j) \leq r$, where δ is the Euclidean distance. We assume G_r is connected. Let NB_i be the set

of neighbors of s_i, where $s_j \in NB_i$ iff $(i,j) \in E_r$. Each node is aware of its (x,y) coordinates in an absolute or relative co-ordinate system, and also knows the outer boundary of the terrain of deployment in the same co-ordinate system. Let s_i^x and s_i^y be the x and y coordinates of node s_i.

Let $MAP : SN \rightarrow I \times I$ associate each $s_k \in V_r$ with a pair of grid coordinates (i,j) where $s_{i,j} \in V_g$ is the node to be emulated. Let $CELL_{i,j} = \{s_k | MAP(s_k) = (i,j)\}$, i.e., the set of nodes that collectively emulate the (i,j)-th node in the virtual grid. In this paper, we assume that the subgraph of G_r induced by nodes in $CELL_{i,j}$ $(\forall i,j)$ is connected.

A simple, cell-based protocol can be used to emulate G_g on G_r. We assume that localization and neighbor discovery has occurred and each node can compute $MAP(s_i)$ and knows the number and location of its one-hop neighbors. The routing table at node s_i is a function $RT_{s_i} : DIR \rightarrow NB_i \cup \{NULL\}$, where $DIR = \{No, Ea, We, So\}$ is the set of directions in the oriented grid. Some entries of the routing table can be filled in using the initially available information. For example, $RT_{s_i}(No)$ can be associated with s_j if $\exists s_j \in NB_i$ s.t. $s_j \in CELL_{i-1,j}$. If there is no node in NB_i that lies in a neighboring cell, the routing table entry is $NULL$. Now each node s_i now tries to discover multi-hop paths to neighboring cells that are associated with $NULL$ entries in RT_{s_i}. Each node s_i initially broadcasts its own (small) routing table to all its neighbors. When a node s_i receives a message from some $s_j \in NB_i$ where $MAP(s_j) \neq MAP(s_i)$, the message is ignored. If $s_j \in NB_i$, $MAP(s_j) = MAP(s_i)$ and $\exists d \in DIR$ such that $RT_{s_j}(d) \neq NULL$ and $RT_{s_i}(d) = NULL$, s_i sets $RT_{s_i} = s_j$. Since new nodes can be added to the network or existing nodes can leave or fail, the above protocol should execute periodically. The user can choose any routing protocol implemented on the oriented grid using the routing table to forward messages between adjacent cells of the grid.

This topology emulation protocol is time and energy-efficient because (i) the path setup in all cells occurs in parallel, (ii) messages cross at most one cell boundary before being suppressed, and (iii) the latency is proportional to the maximum, over all cells, of the length of the longest path between pairs of nodes in a cell.

5.2. Binding virtual processes to physical nodes

To execute the mapping on the real network, a mechanism is required where the functionality of the v nodes of the virtual topology is somehow mapped onto the n nodes of the underlying, real network. We assume that $n > v$ and describe one way of accomplishing this mapping, using the same cell-based approach as for topology emulation.

Let $gc_{i,j}$ denote the geographic center of cell i,j and $gc_{i,j}^x$ and $gc_{i,j}^y$ denote its x and y coordinates. Since each node s_i knows its own coordinates, the cell size, and the boundary of the terrain of deployment, it can compute $gc_{i,j}^x$ and $gc_{i,j}^y$. Also, it can compute its Euclidean distance to the center of the cell. Each node maintains a flag $leader$ initially set to $TRUE$. Each s_i now broadcasts $\delta(s_i, gc_{MAP(s_i)})$ to its neighbors. As in the path setup phase, messages crossing cell boundaries are suppressed. If node s_i receives a value from $s_j \in NB_i$ that is less than its own δ value, it sets $leader = FALSE$ and broadcasts the updated value to all $s_j \in NB_i$. Eventually, the only node whose $leader = TRUE$ will be the one that has not received δ values lower than its own from any of its neighbors, and hence is the node closest to the geographic center. This node can start executing the program specified for node $s_{i,j}$ in G_g. The choice of the node closest to the center of the cell as leader is an effort to align the problem geometry and the network geometry as closely as possible. Residual energy level or more sophisticated metrics could also be employed depending on the particular application, especially if the role of leader is to be periodically rotated among nodes in the cell.

6. Related Work

The state-centric programming framework proposed in [12] has a similar motivation as our work: to define an appropriate mental model that application developers can use to program distributed sensor networks. The framework is based on the concept of collaboration groups which abstract common patterns in application-specific communication and resource allocation. We consider the state-centric framework complementary to our proposed methodology in the sense that it provides programming primitives and a means of implementing the synthesized algorithms on the target network. The UW-API [15] is an example of a communication library for distributed computations in sensor networks, motivated by the MPI library [9]. Similar to MPI, some of the UW-API primitives are to be invoked by a single sensor node and others are for collective communication, to be invoked simultaneously by a group of nodes in a geographic region. All operations take place on regions, which can be created using specific primitives. Even barrier synchronization is supported for the sensor nodes that lie within a region.

There are other efforts in the community that are addressing algorithm development [5, 11], task allocation [7, 21], middleware services [19, 20], and programming models [1, 2] for sensor networks. However, we are not aware of any coherent top-down methodology to simplify and ultimately automate the design and synthesis of networked sensing applications.

7. Discussion

End to end application design for large scale sensor networks is a complex process. A significant fraction of this complexity is at the networking layer because a host of interacting low level services and protocols are required for the sensor network to be functional. In addition, the end user is required to coordinate the execution of different tasks at the application level, and their interactions with each other and with the physical world through the sensing interface. System wide energy performance has to be optimized for extending the network lifetime, and issues such as fault tolerance must also be handled. State of the art protocol-centric approaches assume an omniscient end user who has a good understanding both of the application domain and of wireless networking issues, and who is capable of energy efficient and robust cross-layer design and customization of the protocol stack. There is now a growing body of research on high level abstractions and middleware that aim to hide low level networking details from the application developer, and thereby reduce the design complexity.

Design methodologies that employ multiple layers of abstraction to reduce design complexity risk doing so at the expense of performance of the implementation compared to hand-optimized designs. We believe, however, that the obvious and significant benefits of this approach - rapid first-order analysis of algorithms and greatly reduced complexity of programming - outweigh performance considerations, especially where hand-optimization at reasonable cost might not be possible.

References

[1] T. Abdelzaher, B. Blum, Q. Cao, D. Evans, J. George, S. George, T. He, L. Luo, S. Son, R. Stoleru, J. Stankovic, and A. Wood. EnviroTrack: An Environmental Programming Model for Tracking Applications in Distributed Sensor Networks. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, 2004.

[2] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, and R. Han. MANTIS: System Support for MultimodAl NeTworks of In-situ Sensors. In *2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 50–59, 2003.

[3] H. M. Alnuweiri and V. K. Prasanna. Parallel architectures and algorithms for image component labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(10):1014–1034, 1992.

[4] R. Barr, J. Bicket, D. Dantas, B. Du, T. Kim, B. Zhou, and E. Sirer. On the need for system-level support for ad hoc and sensor networks. In *Operating Systems Review, ACM*, April 2002.

[5] R. S. Bhuvaneswaran, J. L. Bordim, J. Cui, and K. Nakano. Fundamental protocols for wireless sensor networks. In *International Parallel and Distributed Processing Symposium (IPDPS) Workshop on Advances in Parallel and Distributed Computational Models*, April 2001.

[6] S. H. Bokhari. *Assignment Problems in Parallel and Distributed Computing*. Boston: Kluwer Academic, 1987.

[7] B. Bonfils and P. Bonnet. Adaptive and decentralized operator placement for in-network query processing. In *Information Processing in Sensor Networks (IPSN)*, 2003.

[8] E. Cheong, J. Liebman, J. Liu, and F. Zhao. TinyGALS: A programming model for event-driven embedded systems. In *ACM Symposium on Applied Computing (SAC)*, March 2003.

[9] MPI Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications and High performance Computing*, 8(3/4), 1994.

[10] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of Programming Language Design and Implementation (PLDI)*, 2003.

[11] B. Krishnamachari and S. Iyengar. Bayesian algorithms for fault-tolerant event region detection in wireless sensor networks. *IEEE Transactions on Computers*, 53(3), 2004.

[12] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao. State-centric programming for sensor-actuator network systems. In *IEEE Pervasive Computing*, 2003.

[13] R. Min and A. Chandrakasan. Top five myths about the energy consumption of wireless communication. *Mobile Computing and Communications Review*, 6(4), 2003.

[14] K. Nakano, S. Olariu, and A. Zomaya. Energy-efficient routing in the broadcast communication model. *IEEE Transactions on Parallel and Distributed Systems*, 13(2):1201–1210, December 2002.

[15] P. Ramanathan, K. C. Wang, K. K. Saluja, and T. Clouqueur. Communication support for location-centric collaborative signal processing in sensor networks. In *DIMACS Workshop on Pervasive Networks*, May 2001.

[16] M. Singh, A. Bakshi, and V. K. Prasanna. Constructing topographic maps in networked sensor systems. Technical Report CENG-2004-09, Department of EE-Systems, Univ. of Southern California, June 2004.

[17] M. Singh, A. Pathak, and V. K. Prasanna. Constructing and maintaining a clustered mesh topological infrastructure in sensor networks. Technical Report CENG-2004-08, Dept of EE-Systems, Univ. of Southern California, May 2004.

[18] M. Singh and V. K. Prasanna. Energy-optimal and energy-balanced sorting in a single-hop wireless sensor network. In *International Conference on Pervasive Computing and Communications (PERCOM)*, March 2003.

[19] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, March 2004.

[20] Y. Yu, B. Krishnamachari, and V. K. Prasanna. Issues in designing middleware for wireless sensor networks. *IEEE Network*, 18(1), 2004.

[21] Y. Yu and V. K. Prasanna. Energy-balanced task allocation for collaborative processing in wireless sensor networks. *(accepted by) Mobile Networks and Applications (MONET), special issue on Algorithmic Solutions for Wireless, Mobile, Ad Hoc and Sensor Networks*, 2004.