

A Hierarchical Model for Distributed Collaborative Computation in Wireless Sensor Networks *

Mitali Singh and Viktor K. Prasanna
Computer Science Department
University of Southern California
Los Angeles, CA-90089.
{mitalisi, prasanna}@usc.edu

Abstract

*Clustering is an important characteristic of most sensor applications. In this paper we define **COSMOS**, the Cluster-based, heterOgeneous **MO**del for Sensor networks. The model assumes a hierarchical network architecture comprising of a large number of low cost sensors with limited computation capability, and fewer number of powerful clusterheads, uniformly distributed in a two dimensional terrain. The sensors are organized into single hop clusters, each managed by a clusterhead. The clusterheads are organized in a mesh-like topology. All sensors in a cluster are time synchronized, whereas the clusterheads communicate asynchronously. The sensors are assumed to have multiple power states and a wake-up mechanism to facilitate power management. To illustrate algorithm design using our model, we discuss implementation of algorithms for sorting and summing in sensor networks.*

1. Introduction

Distributed sensor networks are becoming the enabling technology for implementation of a large number of parallel-distributed systems that interact with the physical world. Example applications include environment monitoring, target tracking and fire detection [4]. Low cost, adequate coverage, robustness, real time performance, energy-awareness are some of the desired goals in designing sensor applications. A sensor is designed to be small and low-cost and thus individually has limited processing power. However, the aggregated performance of a dense sensor network is capable of meeting the performance requirements of several challenging applications.

A cluster-based network architecture is used in implementation of several collaborative signal processing sensor applications [18]. A cluster consists of a small set of spatially adjacent sensors and a clusterhead. The sensors in a cluster collect data and transmit to the clusterhead, which performs the required data aggregation and computation, and transmits the result either to the base station or to another clusterhead. A cluster-based approach is more time and energy efficient as compared to direct transmissions from all sensors to a distant base station. Moreover, the high correlation between the data from neighboring sensors can be exploited to achieve large amounts of data compression and data aggregation [9] at the clusterhead. It is important to observe that the clusterhead performs more computation than the sensors. Thus, it is required that the clusterhead is more powerful (implying larger battery, higher bandwidth, more memory and a faster processor) than the sensors.

Sensor applications have multi-objective performance requirements. A sensor network is desired to be low-cost and yet capable of meeting stringent performance and robustness requirements of real time applications. These can be met by deployment of a heterogeneous sensor network comprising of a large number of low cost, less powerful sensors and fewer numbers of more powerful clusterheads. The high density of low cost, less powerful sensors provides sufficient coverage, robustness and connectivity in the network, while keeping the network cost low. The clusterheads supplement the network with faster communication and computation capabilities.

The main contribution of this paper is definition of **COSMOS**, the Cluster-based, heterOgeneous **MO**del for Sensor networks. The model assumes a hierarchical network architecture comprising of a large number of low-power sensors organized into spatial clusters, and fewer number of clusterheads. The sensors within a cluster communicate in a time synchronized manner, using single hop communication. The clusterheads form a mesh like topol-

*This work is supported by the DARPA Power Aware Computing and Communication Program under contract no. F33615-02-2-4005.

ogy and communicate asynchronously.

As an illustration of algorithm design using COSMOS, we discuss implementation of sorting and summing algorithms in sensor networks. Sorting is an important kernel for several image processing and data management sensor applications [2]. Sum is representative of a large set of sensor operations such as finding the max or min of a data set, counting the number of detected targets [7], or computing average statistics for the network.

The rest of the paper is organized as follows. Related research is discussed in Section 2. COSMOS is defined in Section 3. Performance metrics and tradeoffs for sensor applications are analyzed in Section 4. The algorithms for summing and sorting are described in Section 5 and we conclude in Section 6.

2 Related Work

Prior work closely related to our research is the Wireless Sensor Network (WSN) model proposed in [1]. The WSN model assumes a network of localized, time synchronized, homogeneous sensors organized in a mesh-like topology over a two dimensional terrain. The model is appropriate for small size networks but is not scalable. Global time synchronization and initialization is not feasible in large-scale sensor networks without inflating the cost of the network (for example providing each sensor with a GPS). COSMOS assumes a cluster-based, heterogeneous network architecture. Sensors are uniformly distributed over a two-dimensional terrain, organized into spatial clusters. All sensors within a cluster are time synchronized with the *clusterhead*. The *clusterheads* form a mesh-like topology and are asynchronous. Each *clusterhead* has a globally unique id but the sensor ids can be reused across clusters.

COSMOS assumes single-hop communication within each cluster. The Broadcast Communication Model (BCM) discussed in [12] has been utilized for design of algorithms that minimize broadcasts in single hop networks. The BCM model has been primarily utilized for minimizing execution time and number of broadcasts in the system. COSMOS also abstracts the energy characteristics of the network and assumes presence of a wake up mechanism to facilitate power management of the sensors. In small size sensor networks, with short range radios (output power up to 20dBm), broadcast energy is of similar magnitude as the reception energy [11]. We define a uniform cost model for energy analysis of single hop clusters. This model assumes that energy dissipated in transmitting, receiving, or computing one unit data is unity.

One of the main assumptions of COSMOS is a cluster-based sensor network. Clustering has been exploited by several signal processing algorithms and communication protocols [18] [16] for sensor networks. Data aggregation

and compression at the *clusterhead* reduces communication costs in the network. COSMOS assumes that *clusterheads* communicate asynchronously and the sensors in a cluster are time synchronized with the *clusterheads*. TinyGALS, a globally asynchronous and locally synchronous model for event-driven embedded systems is described in [5]. Note that TinyGALS is a programming model, while COSMOS is a computation model.

Algorithms for sorting [3] and finding the sum [1] have been analyzed using the WSN model. We discuss implementation of sorting and summing algorithms to illustrate algorithm design using COSMOS.

3 The COSMOS¹

As the research focus is shifting from design of passive sensor networks to dynamic, self-organizing, and smart sensors systems, in-network computation is becoming increasingly important. Design of efficient algorithms for sensor applications requires a simple and realistic computation model for the underlying sensor network. The Wireless Sensor Network (WSN) model proposed in [1] provides a clean abstraction of several features of a wireless network such as the broadcast medium, range control, and channel interference. The model is suitable for small sized networks but is not scalable. Moreover, it does not support clustering, which is an important feature of most sensor applications.

In this section, we define COSMOS, the **Cluster-based heterogeneous Model for Sensor networks**. The model assumes a hierarchical network architecture comprising of a large number of low-cost, less powerful *sensors*, and a fewer number of higher-cost, more powerful *clusterheads*. We use the term *clusterheads* to distinguish more powerful sensors from low cost sensors. We use the notation *sensors* to represent low power sensors and unitalicized sensors more generally, to represent both types of sensors. The *sensors* primarily perform the task of data collection and low-level signal processing. The *clusterheads* are assigned more intensive tasks such as collaborative computation, data aggregation, and network management. The two level hierarchy permits the network to provide robust coverage and adequate computation power at low cost.

A cluster-based architecture is characteristic of most sensor network applications [16][18]. A clustering approach makes the network more scalable as compared to a network with all homogeneous sensors. For example, *sensor* ids can be reused across clusters, and time synchronization is required only within clusters. Clustering also permits the application to exploit correlation for compression and data fusion. A spatial clustering based approach has been exploited in [16] for designing efficient communication protocols for

¹cosmos (n.): A complex, orderly self inclusive system

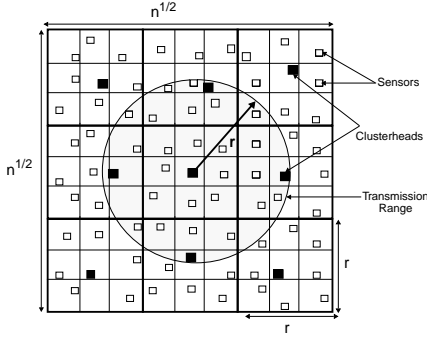


Figure 1. COSMOS

collaborative signal processing applications.

COSMOS assumes uniform distribution of sensors in a two dimensional terrain. The terrain is divided into unit area cells, each containing a sensor. The sensors are organized into single-hop spatial clusters, each containing a powerful *clusterhead*. The size of the cluster is limited by the transmission range r of the *sensors*. The *clusterheads* have transmission range $R \geq r$ and are assumed to be organized in a mesh-like topology (as illustrated in Figure 1). A regular topology facilitates design of efficient algorithms for global collaboration and information exchange. Note that the *clusterheads* are more resourceful as compared to the *sensors*, and can thus accommodate the additional task of network management and topology maintenance.

Sensors communicate over one or more wireless channel(s). A transmission from any sensor with transmission range r , is guaranteed to be received by all sensors located within distance r from the transmitting sensor. A collision occurs if the receiving sensor is in the transmission range of two or more sensors that transmit concurrently using the same channel. Consider the transmissions in Figure 2. Note all the transmissions can take place simultaneously. For example, consider the transmissions from $A \rightarrow C$ and $B \rightarrow D$. Both can take place concurrently since sensor C is not within transmission range of B and D is not within transmission range of A . However, sensors A and E cannot transmit to B at the same time.

Time synchronization over large sized networks is not feasible without significant overheads. We assume that the *clusterheads* communicate asynchronously using message passing. Delays in message delivery are considered to be unpredictable. However, we assume that low-level protocols ensure reliable, ordered delivery of messages in finite time. Moreover, synchronous communication is assumed between all sensors with in a cluster. All *sensors* in a cluster are time synchronized with the *clusterhead* of the cluster they belong to.

Sensors in a wireless network dissipate significant en-

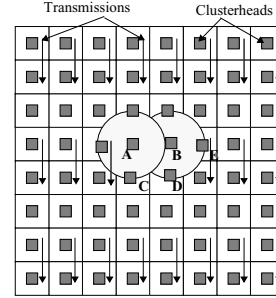


Figure 2. Collision-free transmissions

ergy receiving packets not intended for them or busy waiting. The state of the art systems support multiple power states of the sensors [14]. We assume that a sensor has two power states, active and asleep. Energy dissipation is negligible in the asleep state. COSMOS defines a wake up mechanism that is used to *wake up* (activate) an asleep sensor when required. Two types of wake up mechanisms are defined, an internal mechanism and an external mechanism. The internal mechanism is based upon hardware counters in the sensor. Using this mechanism a sensor can switch off for a specified duration. Energy overheads for maintaining the internal counter are assumed to be negligible. The external mechanism relies on a paging channel that can *page* the sensor to wake up when an event of interest occurs. Note that internal timers are suitable for synchronous networks. However, in an asynchronous environment wake up mechanism based on only internal timers is unreliable. The paging channel can be used to ensure that the destination sensor is awake to receive a transmission.

Implementation of a paging channel may vary in different systems. For example, a sensor may be equipped with low power RF or laser radio for paging [15]. When a sensor is asleep, the processor, data radio, and other components of the sensor are turned off, but the paging channel remains active. When a wake up message is received, the sensor becomes fully functional. We assume that the paging channel is collision-tolerant. This implies that collisions are detected, and treated like wake up messages. A paging channel is a low power channel, and dissipates negligible energy when idle or in receive mode (e.g. a laser radio [6]). A paging channel has limited range (or line of sight for laser) and is thus used to page only near by sensors. Energy dissipation in the paging channel is proportional to the number of wake up messages transmitted over the channel.

A *sensor* has one unit of memory. A time step is defined as the time taken by a *sensor* to transmit, receive or locally compute one unit of data. A *sensor* is assumed to have $p \geq 1$ communication channels, but in a single time step, it can tune to at most one communication channel. The

clusterhead is assumed to be more powerful than a *sensor*. It has memory $m \geq r^2$, bandwidth $b \geq r^2$, and computational power $c \geq r^2$. This implies, that a *clusterhead* can receive or transmit b data units and compute c data units in a single time step. A *clusterhead* is assumed to have $P \geq 1$ communication channels.

The execution time is a measure of the total computation and communication time of an algorithm. As is customary, for the sake of analysis only, we assume that all *clusterheads* have the same clock frequency. A time step is defined as the number of cycles required to compute one unit of data by a *sensor*. The computation time is well defined, but the communication time may vary in an asynchronous environment. In design of asynchronous systems the communication time complexity is measured as the longest chain of messages that occur before the algorithm terminates [10]. For our analysis, we define the time complexity as the weighted longest chain of computation and communication in the network. The weights are defined as follow. Unit weight is defined for each computation and communication at the *sensor*. Computation of one unit of data at the *clusterhead* has weight $1/c$, and communication of a single data unit has weight $1/b$.

The total energy dissipation in the network is the sum of the communication, computation, and paging energy. The communication energy has two components, the transmission energy and the reception energy. A simple model for energy dissipation per unit of data is given by $\alpha + \beta r^\gamma$ [11]. Here, α is the range independent parameter, whose value depends on the electronics of the system at transmitter and receiver. βr^γ represents the range dependent radiation power of the radio with transmission range r . For small sized networks, α dominates over the range dependent energy component [11]. Thus, for small sized networks the transmission and reception energy is constant independent of the transmission range, and we normalize it to unity. For large-sized networks energy dissipation for transmission is dominated by the radiation energy and is proportional to r^n . We assume that one unit of energy is dissipated for computing one unit of data. Energy dissipation over the paging channel is proportional to the number of wake up messages transmitted over the channel.

To summarize, our model assumes the following:

1. The network consists of n sensors, uniformly distributed over a two dimensional terrain. There are two types of sensors in the network, a large number of low-cost, low-power *sensors*, and a smaller number of powerful *clusterheads*.
2. The *sensors* have transmission range r , and are organized into disjoint clusters of size r^2 . Each cluster is managed by a distinct *clusterhead*.
3. The *clusterheads* are organized in a mesh-like topology of size $\sqrt{n}/r \times \sqrt{n}/r$ and have transmission range

$R \geq r$.

4. The network has been localized and initialized. All *clusterheads* have globally unique ids. The ids of the *sensors* in a cluster are unique only within the cluster they belong to. .
5. No global clock exists in the system. The *sensors* in a cluster are time synchronized with their *clusterhead*. The communication between *clusterheads* is asynchronous using message passing.
6. Each *sensor* has unit memory, unit processing power, and unit bandwidth. The time taken to transmit, receive, or locally compute one unit of data is defined as one time step.
7. Communication between the sensors is over one or more wireless communication channel(s). A transmission from any sensor with range r is guaranteed to be received by any other sensor within distance r from the transmitting sensor. A collision occurs if a receiving sensor lies with transmission range of two or more sensors that transmit at the same time.
8. A *sensor* has $p \geq 1$ communication channel(s), which are shared with other sensors within the cluster. In one time step a *sensor* can transmit or tune to at most one communication channel.
9. The *clusterheads* have $P \geq 1$ communication channel(s). In one time step, a *clusterhead* can transmit or tune to at most b communication channels.
10. The *clusterheads* are more powerful than the *sensors*. Each *clusterhead* has memory of size $m \geq r^2$ and communication bandwidth $b \geq r^2$. This enables it to transmit or receive b data elements in one time step, either from (to) another *clusterhead* or b sensors (using distinct channels) in its cluster. A *clusterhead* also has more processing power than a *sensor*. In a single time step it can process $c \geq r^2$ units of data.
11. Communication within a cluster is single hop. The *clusterheads* communicate among themselves in a single hop or using multiple hops depending on range R .
12. All sensors have two power states. They can be *active* or *asleep*. Energy dissipation is zero in the asleep state.
13. All sensors have hardware counters that can be utilized for *timed* power state transition. The energy dissipated by the counters is negligible.
14. The sensors are also equipped with low power, short-ranged radios that are used for *paging* nearby sensors. All sensors within a cluster, and adjacent *clusterheads* can page each other directly. Paging messages to distant sensors can be routed through intermediate *clusterheads*. Energy dissipation over the paging channel is proportional to the number of paging messages transmitted over the channel.

15. A paging message is assumed to be of the following four types (a) Wake up a specific sensor, (b) Wake up all sensors in a specific cluster, (c) Wake up all sensors in a given cluster with id smaller than the transmitted id, and lastly, (d) Wake up all sensors in a specific with id larger than the transmitted id.
16. The execution time of an algorithm is the longest weighted computation and communication chain.
17. The energy dissipated by the *sensors* within a cluster follows the *uniform cost model*. This model assumes that energy dissipation for transmission, reception or local computation of one unit of data is unity.
18. We assume that the path loss exponent $\gamma = 2$. The energy dissipation of the clusterheads follows the *square cost model*. This model assumes that energy dissipation for reception or local computation of one unit of data is unity. Energy dissipation for transmission of one unit of data is R^2 for the clusterheads.
19. The total energy dissipation in the system is the sum of the computation, communication, and paging energy.
20. All messages transmitted are assumed to reach the destination sensor in finite time. Messages are delivered in the order that they were transmitted.

Note the size of the cluster is an important model parameter. It determines the ratio of the number of *clusterheads* to *sensors* in the network, and thus the overall cost of deployment. We assume the cluster size to be r^2 , where r is the transmission range of the *sensors*. The size of the cluster imposes constraints on the memory and bandwidth of the clusterhead. We observe that most sensors meet these specifications. Consider for example, the architectural specifications of the PASTA [14] sensor.

The PASTA sensor is being designed for target detection and classification in a battlefield. This application requires deployment of two types of sensors. Low-power *tripwires* and the PASTA sensors. The *tripwires* monitor the field for seismic and acoustic signals. If a perceived signal is higher than a threshold value, lightweight signal processing is performed to reduce false alerts. On detecting an event of interest the nearby PASTA sensors are woken up to run computationally intensive algorithms for target detection and classification.

The PASTA sensor is equipped with a PXA250 processor, 64MB SDRAM, and a low power radio along with other components. One example of a low power radio is the RFM TR1000 radio, which has transmission range of 20 meters and data rate 115.2 Kbps. A sensor network can have density ranging from 0.1 to 20 sensors/ m^2 , depending on the application. Thus, a cluster size can be as large as 8000. The memory and bandwidth of the PASTA sensor is sufficiently large to aggregate data from the *tripwires* in the cluster.

4 Performance Metrics

The computational model discussed in the previous section can be used for analyzing system design and algorithms for sensor networks. System design involves selection of parameters such as density of network and ratio of *clusterheads* to *sensors*. The aim is to keep the cost of the network low, while ensuring that the network is capable of meeting the application specifications such as response time, robustness, and lifetime. Algorithm design for sensor networks is challenging, as most applications require multi-objective optimizations. Some of the metrics may be orthogonal to each other. For example, reducing energy may increase overall execution time for some applications. In this section, we define various performance metrics relevant to sensor networks and discuss the tradeoffs.

1. **Time:** Time is an important metric for designing algorithms for sensor networks. The run time of an algorithm depends on several factors such as the network diameter, total number of transmissions, and the amount of parallelism in the algorithm. The transmission range of a sensor can be increased to reduce the network diameter. However, it increases the interference in the network, decreasing the number of transmissions that can take place concurrently. The tradeoff must be well analyzed for design of time efficient algorithms. Throughput and latency are more relevant metrics to sensor applications that are periodic in nature. We analyze the summing algorithm in Section 5.2 for time performance.
2. **Response Time:** Several real time applications impose stringent requirements on the response time of the system. Response time is defined as the time taken by the system to respond to an input or stimuli. Consider for example a sensor network deployed over a battlefield that periodically monitors the environment for enemy intrusion. On detecting an intruder, a sensor must immediately report to the base station. Information may become redundant if the time for reporting the event is high, as the enemy may have moved away from the area. Increasing the transmission range of the sensors may reduce response time but increase energy dissipation in the system.
3. **Overall Energy Dissipation:** The lifetime of a sensor network is determined by the battery power of the sensors. Sensor applications aim at minimizing overall energy dissipation in the system. Algorithms designed for energy optimality may not be time optimal. Energy versus time tradeoffs are discussed in Section 5.2.
4. **Energy Balancedness:** It is desired that energy dissipation in a sensor network is uniform. Consider a sensor network consisting of n sensors with overall energy

dissipation $E(n)$. An energy balanced algorithm satisfies the property that each sensor dissipates $E(n)/n$ units of energy. This property ensures that the sensor network remains fully functional for the entire duration, without early die out of some over used, critical sensors. Energy-balancedness has been investigated for algorithm design in single hop networks in [17]. An algorithm using COSMOS is energy balanced if energy dissipation is same for all *clusterheads* and also same for all *sensors*. Note the *clusterheads* may dissipate more energy than the *sensors*. The sorting algorithm discussed in Section 5.1 is energy-balanced.

5. **Network Lifetime:** Sensor applications often require deployment of sensors in remote areas, where redeployment is costly. The density and types of sensors determine both the network lifetime and the cost. The cost versus network lifetime tradeoff must be evaluated at system design time. Network lifetime can also be improved by designing energy efficient algorithms. Moreover, it must be ensured that the network does not collapse due to failure of some small number of sensors.
6. **Broadcasts:** Minimizing number of broadcasts has several advantages [12] such as reduction in the communication energy, lesser interference, and enhanced security in the network. Consider for example communication between the *clusterheads* in COSMOS. Energy dissipation for broadcasting a data unit consumes energy R^2 , where R is the transmission range of the *clusterheads*, whereas energy dissipation for computation or reception of one unit of data is unity. Reduction in broadcasts in the system improves network security as it prevents eavesdropping and data collection by an intruder. However, for small-sized single hop clusters minimizing broadcast may not be sufficient for energy optimization. The algorithm must also ensure efficient power management of the sensors to reduce reception energy.
7. **Robustness:** Sensors and communication links in a network may fail due to power failure or harsh environmental conditions. Hardware redundancy (replication of sensors) provides a simple mechanism for improving network robustness, but increases the network cost. A heterogeneous back up scheme has been proposed in [8], where one type of resource is backed up by another. For example, increasing computation if the communication link becomes less reliable. Multimodal data fusion can also be exploited to recover from errors in the system. COSMOS promotes data clustering at the *clusterheads*. The spatial and temporal correlation and data redundancy can be exploited by the *clusterhead* to recover from errors or sensor failures within a cluster. COSMOS assumes a mesh-

like organization of the *clusterheads*. Link or sensor failures may disrupt the mesh topology completely or partially. Robustness of an algorithm can be measured in terms of the number of sensors or link failures it can tolerate. Design of robust algorithms requires further investigation.

8. **Quality of Service:** The quality of service can be measured in a variety of ways such as result accuracy, data transmission rate, and probability of false alarms. Collecting larger data samples and performing more computation can improve precision of results, but at cost of higher time and energy. Algorithms must evaluate the tradeoffs between the QoS requirements of the sensor application and the energy and time overheads.
9. **Cost:** Low cost is a desiderata for deployment of sensor network systems. The two level hierarchical structure proposed in COSMOS assumes a large number of low cost *sensors* and a smaller number of powerful *clusterheads*. The network cost can be reduced by reducing the ratio of powerful *clusterheads* to *sensors*, while ensuring that the network has adequate computation power.
10. **Security:** Network security is an important concern in hostile environments such as the battlefield. Several researchers have focused on design of lightweight protocols for security in sensor networks. However, security should also be considered as an important parameter for designing algorithms for sensor networks. Randomization can be exploited to reduce predictability of the algorithm. The algorithm must refrain from broadcasting sensor ids over the network. Reducing the total number of transmissions in the network, also prevents attacks from hostile eavesdroppers.

5 Illustrative Algorithms

Sorting and summing are important problems in sensor networks. Several protocols require sensors to be sorted in order of remaining power or nearness to a monitored target. The sum operation is used for obtaining global statistics about the network. For example, counting the number of targets in a field [7]. In this section we discuss implementation of these two algorithms using COSMOS.

5.1 Bitonic Sort

The bitonic sort algorithm has been analyzed for time optimal implementation in a mesh connected computer system [13], and a wireless sensor network using the WSN model [3]. These implementations consider a homogeneous, time synchronized system. We discuss implementation of bitonic sort in cluster-based architecture, using COSMOS, where communication between the *clusterheads* is

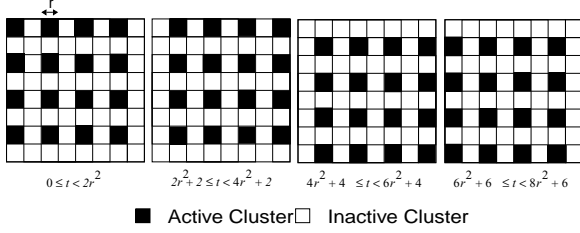


Figure 3. Schedule for data clustering

asynchronous. For sake of illustration, we assume $b = c = r^2$, $R = r$ and a single communication channel.

The algorithm can be implemented using the following three steps. In the first step data is aggregated at the *clusterheads*. Next, sorting is performed between the *clusterheads*, and finally the data is redistributed to all the *sensors*. The procedure is detailed in the following subsections.

5.1.1 Data Aggregation

The r^2 time synchronized *sensors* in a cluster broadcast their data values one by one. Each *sensor* maintains a rank counter, which is incremented on hearing a data value smaller than its own. After r^2 time steps each *sensor* knows the rank of its data element. The data is transmitted in rank order to the *clusterhead*. In $2r^2$ time steps, the *clusterhead* contains the sorted data vector of size r^2 . All *sensors* in the cluster are switched off.

To avoid collisions, data aggregation must not take place simultaneously in neighboring clusters. This is prevented by using the four phase aggregation schedule described in Figure 3. Data aggregation occurs only in the active clusters. The *sensors* in the inactive cluster are asleep, while the *clusterheads* are in receive mode. The *clusterheads* utilize asynchronous message passing to mark the start and completion of the four phases. Reliable delivery of messages is ensured by lower level protocols. Note messages are small in size and overheads due to message loss or re-transmissions are negligible. The schedule is implemented as follows.

Initially, all clusterheads in the odd row and column aggregate data. At end of data aggregation, a clusterhead in the active cluster transmits a finish token to adjacent *clusterheads* in the same row. Any *clusterhead* in an inactive cluster activates its cluster once it has received the finish token from both adjacent *clusterheads* in the same row. Similar steps are used for cluster activation for the remaining two phases. At end of the second phase, *clusterheads* that receive token from adjacent clusterheads in the same column become active. In the last phase *clusterheads* that receive token from both adjacent *clusterheads* in the same row become active.

The time complexity of the algorithm is $8r^2 + 6$. Data clustering takes time $8r^2$ and 6 time steps are required to notify neighboring *clusterheads* to activate their clusters.

5.1.2 Sorting among Clusterheads

Next we adapt the bitonic sort algorithm [13] for sorting the data stored at the *clusterheads*. Each *clusterhead* contains a sorted vector of size r^2 and the *clusterheads* are asynchronous. Given two data vectors a and e , the **comparison-interchange** instruction is defined as follows.

$\forall_{(1 \leq i \leq r^2)}$ **If** (SIGN * ($a[i] - e[i]$) < 0) swap ($a[i], e[i]$)
Note, this instruction executes in one time step as $c = r^2$.

1. **Row and Column Sorting:** This procedure sorts a bitonic sequence of $k \times r^2$ data elements stored in a row (or column) of k adjacent *clusterheads*. Its implementation in an asynchronous network is as follows.

RowMerge (k)

1. Let S_1, \dots, S_k denote the k adjacent *clusterheads*.
 2. **If** $k = 1$ goto Step 7.
 3. Shift data vectors from $S_{k/2+1}, \dots, S_k$ to $S_1, \dots, S_{k/2}$.
 4. Perform comparison-interchange on $S_1, \dots, S_{k/2}$.
 5. Shift rejected vectors from $S_1, \dots, S_{k/2}$ to $S_{k/2+1}, \dots, S_k$.
 6. Invoke in parallel RowMerge ($k/2$) for $S_1, \dots, S_{k/2}$ and $S_{k/2+1}, \dots, S_k$.
 7. End.
-

Figure 4. Pseudo code for RowMerge (k)

Consider one iteration of the algorithm, data shifts hop-by-hop in *reverse* direction from *clusterhead* S_j to $S_{j-k/2}$, where $j > k/2$. Comparison-interchange takes place at $S_{j-k/2}$, and the rejected data vector is shifted hop-by-hop in *forward* direction to S_j . To prevent collisions, the transmit-receive pairs must be interleaved by an idle *clusterhead* (see Figure 5).

A single iteration (as described above) can be implemented as follows. All *clusterheads* maintain variables k , f , g , and T . Here, k denotes the number of *clusterheads* on which the algorithm is invoked in the current iteration. Variables f and g count the number of data vectors sent in forward and reverse directions. At the beginning of each iteration $f = g = 0$. Consider *clusterhead* S_j with $j > k/2$. It is required to transmit $k - j + 1$ data vectors reverse and forward. Thus, $g = k - j + 1$ signifies end of reverse phase, and

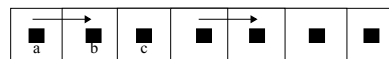


Figure 5. Transmissions in a row

$f = k - j + 1$ marks the end of forward phase for this *clusterhead*. A *clusterhead* with id $j \leq k/2$, transmits $j - 1$ packets reverse and forward. Note $f = g > 0$ indicates end of one iteration and at this point the *clusterhead* decrements k to $k/2$. The algorithm terminates when $k = 1$ at each *clusterhead*. A *clusterhead* transmits when $T = 1$. At start of algorithm $T = 1$ for all *clusterheads* S_j where $(j \bmod 3) = 0$, and $T = 0$ for the rest. The *clusterheads* execute the following steps.

```

If ( $j \leq k/2$ )
  If ( $g < j - 1$ ) SendReverse
    elseif ( $f < j - 1$ ) SendForward
      else  $k = k/2, g = f = 0$ 
    elseif ( $g < (k - j + 1)$ ) SendReverse
      elseif ( $f < (k - j + 1)$ ) SendForward
        else  $k = k/2, f = g = 0$ 

```

SendReverse: A *clusterhead* S_j with $T = 1$, transmits d data packets (each of size r^2), and sets $T = 0, g = g + d$. It transmits $FinT = j$ to S_{j-1} and S_{j+1} . S_{j-1} performs comparison-interchange on received data if the data is destined to itself. It transmits $FinR = j - 1$ to S_{j-2} and S_j . On receiving $FinR = j - 1$ and $FinT = j - 3$, S_{j-2} transmits $T = 1$ to S_{j-1} , implying that S_{j-1} can begin transmitting.

SendForward: Any *clusterhead* S_j with id j and $T = 1$, transmits d data packets, and sets $T = 0, f = f + d$. It transmits $FinT = j$ to S_{j-1} and S_{j+1} . S_{j+1} performs comparison-interchange on the received data, if the data is destined to itself. It transmits $FinR = j + 1$ to S_{j+2} and S_j . On receiving $FinR = j + 1$ and $FinT = j + 3$, S_{j+2} transmits $T = 1$ to S_{j+1} , implying that S_{j+1} can begin transmitting.

Each iteration of the above algorithm requires $k \cdot r^2$ transmissions. Only $k/3$ *clusterheads* transmit in a given time step. Thus, each iteration requires $3k$ time steps. Each transmission has an overhead of two transmissions for accessing the channel in procedures *SendForward* and *SendReverse*. Total time taken is $5k$.

Let $T(k)$ denote the time complexity of the algorithm.

$$T(k) = T(k/2) + 5k \text{ if } k > 1 \text{ and } T(1) = 1.$$

This implies $T(k) = 10k$. Note that *ColumnMerge* (k) can be implemented in a similar manner in time $10k$.

2. **Vertical Merge Sort:** This procedure sorts in increasing (or decreasing) row-major order a $J \times K$ array comprising of two vertically aligned $J/2 \times K$ arrays.

The procedure *ColumnMerge* (J) can be invoked in parallel on all columns without collisions. Figure 2 depicts the transmissions at a given time step. Clusterhead A transmits to C and B transmits to D . Note C does not observe interference from B , and only receives signal from A . The time complexity is $10J$.

VerticalMerge (J, K)

1. **for** all columns in parallel **do** *ColumnMerge*(J).
 2. **for** all rows in parallel **do** *RowMerge*(K).
 3. End.
-

Figure 6. Pseudocode for *VerticalMerge* (J, K)

TwoColumnMerge (J)

1. Let $S_{1,j}, \dots, S_{J,j}$ be J *clusterheads* for any column j .
 2. Compare-interchange the vectors in each *clusterhead*.
 3. **If** $J > 1$
 - (a) Exchange the rejected vectors of $S_{1,j}, \dots, S_{J/2,j}$ with accepted vectors of $S_{J/2+1,j}, \dots, S_{J,j}$.
 - (b) In parallel perform *TwoColumnMerge*($J/2$) on the *clusterheads* $S_{1,j}, \dots, S_{J/2,j}$ and $S_{J/2+1,j}, \dots, S_{J,j}$.
 4. End.
-

Figure 7. Pseudocode for *TwoColumnMerge* (J)

The procedure *RowMerge* (K) can be invoked on all rows in parallel and the time complexity is $10K$. However, before invoking this procedure, it must be ensured that all *clusterheads* in the chosen row have completed procedure *ColumnMerge* (J). This is accomplished by an additional step *Terminate* (K) between Steps 1 and 2 of *VerticalMerge* (J, K).

Terminate (K): Let us consider K *clusterheads* in any row denoted by S_j , where $1 \leq j \leq K$. When procedure *ColumnMerge* (J) terminates for S_K , it transmits $Fin = K$ to S_{K-1} . Each S_j with $1 < j < K$, on termination of procedure *ColumnMerge* (J) on its respective column, checks if it has received $fin = j + 1$, and if so, it transmits $fin = j$ to S_{j-1} . When S_1 receives $fin = 2$, it transmits $fin = 1$ to all the *clusterheads* in the corresponding row. On receiving $fin = 1$ a *clusterhead* in the row, begins procedure *RowMerge* (J). The execution time for this algorithm is $2K$ since the variable fin traverses the whole row twice. Thus, the total time for execution of this procedure is given by $10J + 10K + 2K = 10J + 12K$.

3. **Two Column Merge:** The procedure *TwoColumnMerge* (J) sorts a bitonic sequence $(d_1, d_2, \dots, d_{2J})$ stored in a column of J adjacent *clusterheads*, where a *clusterhead* in row i contains data vector d_i and d_{i+j} initially. At the end of the procedure a *clusterhead* in row i contains vectors b_{2i-1} and b_{2i} , where $(b_1, b_2, \dots, b_{2J})$ represents the resultant sorted sequence. Note the analysis and implementation of this procedure is similar to *RowMerge* (J), and thus time taken is $10J$.
4. **Horizontal Merge Sort:** This procedure sorts a $J \times K$ array, composed of two horizontally adjacent $J \times K/2$ arrays. The data movement is similar to

HorizontalMerge (J, K)

1. Let the K columns be C_1, C_2, \dots, C_K .
 2. Move in parallel data from *clusterheads* in columns $C_{K/2+1}, \dots, C_K$ to corresponding *clusterheads* in columns $C_1, \dots, C_{K/2}$.
 3. For each column $C_1, \dots, C_{K/2}$ perform in parallel *TwoColumnMerge* (J).
 4. Move, in parallel, the rejected data vector back to *clusterheads* in $C_{K/2+1}, \dots, C_K$.
 5. **If** $K > 2$ invoke in parallel *RowMerge* ($K/2$) for each of the $2J$ rows of size $K/2$
 6. End.
-

Figure 8. Pseudocode for HorizontalMerge (J, K)

BITONIC-SORT (\sqrt{n}/r)

1. $K=1$
 2. **While** $K < \sqrt{n}/r$ **do**
 - (a) In parallel invoke *HorizontalMerge* ($K, 2K$) for each array of size $K \times 2K$.
 - (b) In parallel invoke *VerticalMerge* ($2K, 2K$) for each array of size $2K \times 2K$.
 - (c) $K = 2K$.
 3. End.
-

Figure 9. Pseudocode for BITONIC-SORT (n)

RowMerge (K), which requires time $10K$. At end of procedure *SendReverse*, a procedure similar to *Terminate* (J) is invoked to ensure all *clusterheads* in a column are ready to execute *TwoColumnMerge* (J). Similarly, *Terminate* (J) is invoked to check if phase *SendForward* can begin. The time complexity of this procedure is $10J + 10K + 2J + 2K = 12(J + K)$.

5. **Bitonic Sort:** Lastly, we describe the algorithm bitonic sort, which sorts n data elements stored in the *clusterhead* mesh of size $\sqrt{n}/r \times \sqrt{n}/r$. Each *clusterhead* stores r^2 data elements. The implementation of *HorizontalMerge* ($K, 2K$) and *VerticalMerge* ($2K, 2K$) has been discussed earlier. A procedure similar to *Terminate* (K) can be used to indicate end of the previous step. The time complexity of *HorizontalMerge* ($K, 2K$) is $36K$. Time complexity of *VerticalMerge* ($2K, 2K$) is $44K$. *Terminate* requires $6K$ time steps since it circulates the variable *fin* along the row and the column. The time complexity of the algorithm is $T(\sqrt{n}/r) = T(\sqrt{n}/2r) + 86\sqrt{n}/r \leq 172\sqrt{n}/r = O(\sqrt{n}/r)$.

5.1.3 Data Dissemination

Lastly, each *clusterhead* wakes up the *sensors* in its cluster and transmits the data. To avoid interference, an activation schedule similar to the data aggregation step can be used.

Time to execute this step is $4r^2 + 6$. From the above analysis, we conclude that bitonic sort can be implemented in a locally synchronous, globally asynchronous, cluster-based, hierarchical sensor network in time $O(r^2 + \sqrt{n}/r)$. Prior analysis [3] using the WSN model demonstrated implementation of the above algorithm in time $O(r^2 + r\sqrt{n})$. Our results show that clustering can improve the time performance of the algorithm. The optimal performance is achieved for $r = n^{1/6}$, which implies a cluster size of $n^{1/3}$.

Clustering also reduces the message passing overheads in an asynchronous network. In the above algorithm r^2 data units are sent in each transmission. If two transmissions are required for handshake between sensors, the relative overhead is only $2/r^2$. The algorithm discussed in [3] considers transmissions of size one data unit each. The handshake overheads are $2/1$ which is 200%.

5.2 Finding the Sum

A time optimal algorithm for finding the sum utilizing the WSN model has been discussed in [1]. In this section we discuss an adaptation of the algorithm using the COSMOS model. Our goal is to highlight energy versus time tradeoffs for the algorithm. We assume $b = c = r^2$. We consider a hierarchical network of n sensors, each containing one data element. The goal is to find the sum of all these elements. The algorithm is described as follows.

In the first step all *clusterheads* find the sum of the sensors within their clusters using the schedule described in Section 5.1.1. The sensors in each cluster are time synchronized, and can thus use internal timers as the wake up mechanism. They are asleep when not transmitting. The total time taken is r^2 . One unit of energy is dissipated at each *sensor*. Each *clusterhead* receives r^2 data elements and computes the sum, and pages the adjacent *clusterheads*. Paging energy is $O(n/r^2)$. The overall energy dissipation is $O(n)$ and time taken is $O(r^2)$.

Next, the *clusterheads* assign $R = r^2$. They divide themselves into blocks of size R^2 each and summing takes place in the (interleaved) blocks in parallel. The time taken for this step is $R^2/b = O(r^2)$. Energy dissipation for communication is $n \cdot R^2/r^2 = O(n \cdot r^2)$. The computation energy is $O(n/r^2)$. The *clusterheads* wake up the neighboring *clusterhead* using the paging channel. Each paging message is transmitted over r hops. Thus, paging energy is $O(n/r)$. The total energy dissipation is $O(n \cdot r^2)$.

In the third step, the $n/(r^6)$ *clusterheads* that contain the partial sums, divide themselves into groups of size k^2 each. The transmission range of each *clusterhead* is fixed to $R = r^2$. The partial sum for each group is computed as follows. The partial sum of each row is found in parallel. Each time a *clusterhead* receives a partial sum, it adds its own value, pages the neighbor, transmits sum, and goes

asleep. Next, the partial sum along the last column in each group is calculated. The time complexity of this step is $O(k/(b.R)) = k/r^4$. Each transmission dissipates $O(r^4)$ energy. Each *clusterhead* transmits $O(1)$ data elements. Paging energy is $O(n/r^5)$. Thus, the overall energy consumed for this step is $O(n/r^2)$.

In the last step, the transmission range of the *clusterheads* is increased to $R = \sqrt{n}$. The $n/(r^6.k^2)$ *clusterheads* transmit sequentially to the destination sensor, where the final sum is computed. Each transmission requires $O(n)$ energy. Each paging message is transmitted over $k.r$ hops. Thus, total paging energy is $O((n/r^6.k^2).(k.r))$. Total energy dissipated for this step is $n^2/(r^6.k^2)$. Time taken for this step is $n/(r^6.k^2.b) = n/(r^8.k^2)$.

For $k = (n/r^4)^{1/3}$, the algorithm runs in time $O(r^2 + (n/r)^{1/3}.1/r^5)$, with energy $O(n.r^2 + n^{4/3}/r^{10/3})$.

Note the above algorithm is time efficient but neither energy optimal nor energy balanced. Some *clusterheads* dissipate more energy than the others. An energy optimal, energy balanced algorithm can be achieved as follows. Repeat the first step of the above algorithm. Next sum across all rows in parallel, followed by summing across the last column of the *clusterheads*. The algorithm requires time $O(r^2 + \sqrt{n}/r^3)$, but reduces energy dissipation to $O(n)$.

6. Conclusion

Sensor networks can be viewed as loosely coupled, parallel-distributed systems. Extensive research has been done in the past for design of algorithms for parallel-distributed systems. The existing algorithms can be adapted for sensor networks (as illustrated in this paper using algorithms for sorting and summing). However, implementation and analysis of these algorithms requires a computation model. We observed that clustering is an important feature in sensor networks, and this motivated us to define COSMOS, a cluster-based, computational model for sensor networks. The model also considers issues such as network scalability and cost of deployment.

Performance evaluation of algorithms for sensor networks requires definition of new metrics. We defined energy-balancedness in [17], as a measure of uniform energy dissipation in the network. Robustness is a critical issue in these networks. Our future work will focus on definition and analysis of fault tolerance in sensor systems.

References

- [1] R. S. Bhuvaneshwaran, 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.
- [2] P. Bonnet, J. E. Gehrke, and P. Seshadri. Towards sensor database systems. In *International Conference on Mobile Data Management (MDM)*, January 2001.
- [3] J. L. Bordim, K. Nakano, and H. Shen. Sorting on single-channel wireless sensor networks. In *International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN)*, May 2002.
- [4] J. C. Chen, K. Yao, and R. E. Hudson. Source localization and beamforming. *IEEE Signal Processing Magazine*, pages 30–39, March 2002.
- [5] 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.
- [6] The Smart Dust Project (Macro Motes). <http://robotics.eecs.berkeley.edu/~pister/SmartDust/>.
- [7] Q. Fang, F. Zhao, and L. Guibas. Counting targets: Building and managing aggregates in wireless sensor networks. Technical Report P2002-10298, Palo Alto Research Center, June 2002.
- [8] F. Koushanfar, M. Potkonjak, and A. Sangiovanni-Vincentelli. Fault tolerance in wireless ad-hoc sensor networks. In *IEEE Sensors*, June 2002.
- [9] B. Krishnamachari, D. Estrin, and S. Wicker. Impact of data aggregation in wireless sensor networks. In *International Workshop on Distributed Event-Based Systems*, July 2002.
- [10] L. Lamport and N. Lynch. Distributed computing: Models and methods. *Formal Models and Semantics, Handbook of Theoretical Computer Science, Elsevier Science Publishers*, B(2):1157–1199, 1990.
- [11] R. Min and A. Chandrakasan. Top five myths about the energy consumption of wireless communication. *Mobile Computing and Communications Review*, 6(4), 2003.
- [12] K. Nakano, S. Olariu, and J. L. Schwing. Broadcast-efficient protocols for mobile radio networks with few channels. *IEEE Transactions on Parallel and Distributed Systems*, 10:1276–1289, 1999.
- [13] D. Nassimi and S. Sahni. Bitonic sort on mesh connected computer. *IEEE Transactions on Computers*, C-27(1):2–7, January 1979.
- [14] The Power Aware Sensing, Tracking and Analysis (PASTA) Project (PACC/ Phase II).
- [15] J. Rabaey, J. Ammer, T. Karalar, S. Li, B. Otis, M. Sheets, and T. Tuan. Picoradios for wireless sensor networks: The next challenge in ultra-low-power design. In *International Solid-State Circuits Conference*, February 2002.
- [16] 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.
- [17] 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.
- [18] M. Younis, M. Youssef, and K. Arisha. Energy-aware routing in cluster-based sensor networks. In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, October 2002.