Sorting on Single-Channel Wireless Sensor Networks *

Jacir L. Bordim, Koji Nakano, and Hong Shen

School of Information Science
Japan Advanced Institute of Science and Technology
1-1, Asahidai, Tatsunokuchi, Ishikawa 923-1292, Japan

Abstract

A wireless sensor network is a distributed system which consists of a base station and a number of wireless sensors nodes endowed with radio transceivers. The main contribution of this work is to present a sorting protocol for multi-hop wireless sensor networks. Our protocol sorts n elements which are initially loaded in n sensor nodes that are organized in a two-dimensional plane of size $\sqrt{n} \times \sqrt{n}$. *The sorting protocol proposed here sorts the n elements in* $O(r\sqrt{n})$ *time slots when* $\sqrt{n} > r$, *where r is the transmission range of the sensor nodes.*

keywords: *wireless sensor networks, sorting, bitonic sorting, sensing devices*

1 Introduction

A Wireless Sensor Network (WSN, for short) is a distributed system consisting of a base station and a number of tiny wireless sensing devices that integrate microsensing and short-range communication capabilities. When deployed in large numbers, these devices can measure aspects of the physical environment in great detail. The data being sensed by the sensor nodes in the network is eventually transferred to a base station, where the information can be accessed.

In a *single-hop* WSN, a sensor node can directly communicate with any other sensor node, whereas in a *multi-hop* WSN, the communication between two sensor nodes may involve a sequence of hops through a chain of pairwise adjacent sensor nodes. There is a single-hop communication between the base station and the sensor nodes, while the communication among the sensor nodes can be either single or multi-hop. There are several possible models for WSN's, in this work we consider WSN's where all the sensor nodes in the network are fixed, short-ranged and homogeneous. We assume that the base station and all the sensor nodes have a local clock that keeps synchronous time, perhaps by interfacing with the base station or with a GPS system [13]. All sensor nodes run the same protocol and can perform computations on the data being sensed. As customary, time is assumed to be slotted and all transmissions take place at slotted boundaries [6, 8]. We employ the commonly-accepted assumption that when two or more sensor nodes which are in the transmission range of each other transmit in the same time slot, the corresponding packets *collide* and are garbled beyond recognition.

In this work we address the sorting problem, where n elements are stored in n sensor nodes which are arranged in a two-dimensional square plane of size $\sqrt{n} \times \sqrt{n}$. Sorting is a fundamental problem with an extensive theory and a wide range of practical applications. It is known that a sequential algorithm takes at least $\Omega(n \log n)$ time to sort a sequence of n elements and that optimal algorithms exist which achieve $O(n \log n)$ time [3]. Also, many optimal parallel sorting algorithms have been reported in the literature for different parallel architectures, such as the Parallel Random Access Machine (PRAM) and the Reconfigurable Mesh (RM) [4, 14].

The sorting protocol proposed in this work follows from the work of Nassimi and Sahni [15], which is an adaptation of the bitonic sort. It was shown in [15] that n^2 elements can be sorted in $O(n)$ time on a Mesh-Connected Parallel Computer. Our sorting protocol sorts n elements which are initially stored in n sensor nodes in $O(r\sqrt{n})$ time slots, for $r < \sqrt{n}$, where r is the transmission range of the sensor nodes. For short-transmission ranges (i.e., small values of r), our sorting algorithm matches the time complexity of the algorithm proposed by Nassimi and Sahni [15], which

*Work partially supported by The Hori Information Science Promotion Foundation.

Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'02)

is optimal. However, our protocol can take as much as $O(r^2)$ time slots if $r \approx \sqrt{n}$. This is due to the fact that with a large transmission range, the number of sensor nodes that can transmit concurrently decreases, since they have to lay well apart from each other to avoid interference. As a consequence, the performance of our protocol decreases. Nevertheless, the assumption that the sensor nodes employ short-range wireless communications, less than 100 meters, is customary [5, 12] since it allows the use of tiny and low-powered radio transceivers. In addition, short-range transmissions allows greater radio channel reuse, thus increasing the aggregate bandwidth available. The Piconet [7] project is developing a prototype embedded network that uses short-range (5 meters) radio communications. The sensor nodes in the WSN can use Piconet to enable wireless connectivity. Therefore, one can argue that WSN's will employ short-range radio transmissions to allow communication among the sensor nodes.

2 Model and Problem Definition

The base station is assumed to be equipped with a large antenna which covers a wide area, so that it can monitor all the sensor nodes under its coverage area. The computation among the sensors is performed in coordination with the base station. A sensor node in a single-hop WSN can tune to a channel to send/receive a packet. At the end of a timeslot, the status of the channel can be: (i) NULL, no packet has been driven into the channel in the current time slot; (ii) SINGLE, exactly one packet has been driven into the channel in the current time slot; or (iii) COLLISION, two or more packets have been driven in to the channel in the current time slot.

When a sensor node transmits a packet with power r, the signal will be strong enough for other sensors to hear it within the Euclidean distance r from the sensor node that originates the packet. Let us observe the channel status of a sensor node. For this purpose, let A be a sensor node in a WSN and let S be the unique sensor node broadcasting in a given time slot. The channel status of A is NULL only if A is outside the transmission range of S. Otherwise, if A is within the radio transmission of S, its channel status is SINGLE. Now, let us consider the case in which two or more sensor nodes are broadcasting at the same time. Clearly, if their transmissions do not interfere (i.e., do not overlap), the channel status of A is as discussed above. In case of overlapping transmissions, the channel status is as follows. The channel status of sensor node A is COLLISION if it is within radio transmission of two or more sensors. Therefore, a sensor node is ensured to receive a packet, only if it lies in the transmission range of the source node and there is no interference from other broadcasts.

In this work, we assume that the sensor nodes in the WSN are organized as a two dimensional square plane of size $\sqrt{n} \times \sqrt{n}$ with coordinates (x, y), $(1 \leq x, y \leq \sqrt{n})$. The plane can be viewed as n cells of unit size 1×1. Let $C(x, y)$, $(1 \leq x, y \leq \sqrt{n})$, denote a cell consisting of all points (x', y'), $(x \leq x' < x+1; y \leq y' < y+1)$. Suppose that each cell $C(x, y)$ has a sensor denoted by $S_{x,y}$. Throughout this work we assume that each sensor node $S_{i,j}$, $(1 \leq i, j \leq \sqrt{n})$, knows its cell location within the grid. Clearly, for any two sensors located in adjacent cells, the farthest distance between them is $\sqrt{5}$. Hence, to ensure the communication between adjacent sensors, a packet must be transmitted with power of at least $\sqrt{5}$ to cover a region of $\sqrt{5} \approx 2.24$. Similarly, sensors in diagonally adjacent cells have distance of at most $2\sqrt{2}$. Thus, a sensor has to transmit with enough power to cover an area of at least $2\sqrt{2} \approx 2.83$ to ensure communication with its neighbors. If the sensors on a WSN of size $\sqrt{n} \times \sqrt{n}$ can broadcast with sufficient power to cover an area of $\sqrt{2n}$, then, any pair of sensors can directly communicate, that is, the WSN essentially allows a single-hop communication. In other words, if the sensors with transmission range r are allocated on a WSN of size $\frac{\sqrt{2}r}{2} \times \frac{\sqrt{2}r}{2}$ $(\approx 0.71r \times 0.71r)$, then a single-hop communication is ensured.

Assume that n elements are stored in n sensor nodes, where each sensor holds exactly one element. Also, let S_i be the sensor node with index i, $(1 \leq i \leq n)$. The *sorting* problem is defined to be the problem of moving the ith smallest element to the sensor S_i, for all $i = 1, 2, \ldots, n$. Our sorting protocol is based upon the work of Nassimi and Sahni [15], which in turn is an adaptation of the Batcher's bitonic sort algorithm [9]. The bitonic sort algorithm sorts a bitonic sequence into nondecreasing order. A sequence $\{a_1, a_2, \ldots, a_{2n}\}$ is said to be *bitonic* if either (i) there is an integer j such that $a_1 \leq a_2 \leq \ldots \leq a_j \geq \ldots \geq a_{2n}$, or (ii) the sequence does not satisfy condition (i) but can be shifted cyclically until condition (i) is satisfied [4, 9].

An interesting propriety of the sorting algorithm proposed in [15], is that operations like comparing and exchanging need only to be performed among the elements that belong to the same row or column. Thus, before presenting the details of our protocols, let us first consider an array of size $1 \times n$ consisting of n adjacent cells. Suppose that the transmission range r of each sensor node equals to n. Obviously, the maximum distance between the sensor nodes that are located at the extreme positions of the array (i.e., S_1 and S_n) is

Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'02)
1087-4089/02 $17.00 © 2002 IEEE

$\sqrt{n^2+1}$. In such a case, a single-hop communication cannot be ensured between S_1 and S_n, since the maximum distance between the two exceeds r. On the other hand, a single-hop communication can be ensured between S_1 and S_{n-1} (and also with any other sensor node that lies between them), since the farthest distance between them is less than r. It should be clear from the above that if collision is to be avoided at S_{n-1}, any other sensor node that broadcasts along with S_1 must be apart from S_{n-1} of a distance greater than r.

3 Sorting on Single-hop WSN's

In this section we present a sorting protocol for single-hop WSN's. Suppose that a WSN has m sensor nodes, where all of them lie in the transmission range of each other, and each sensor has a unique ID in the range $[1, m]$. Let S_i denote the sensor node with ID i $(1 \leq i \leq m)$, that holds an element x_i. The m elements can be sorted in $2m$ time slots as follows. For each time slot i, $(1 \leq i \leq m)$, the sensor node S_i broadcasts x_i on the channel and each sensor node S_j, $(1 \leq j \leq m)$, monitors the channel to receive x_i. By comparing x_j to x_i, each sensor node S_j can compute the rank of its element. Once the ranking of each element has been computed, the elements are routed to their final destination, which incurs in additional m time slots. The following lemma summarizes the above discussion:

Lemma 1 *The elements on a single-hop WSN consisting of m sensor nodes, where each sensor node holds one element, can be sorted in $2m$ time slots.*

Clearly, the above result is optimal considering that at any given time slot, only one sensor node can transmit on the channel. Otherwise, a collision occurs and the packets are lost. If the m sensor nodes are arranged in a $\sqrt{m} \times \sqrt{m}$ array, we can rank and sort the elements in a column/row into either increasing or decreasing column/row order in $2\sqrt{m}$ time slots.

Corollary 1 *When m sensor nodes are arranged in a $\sqrt{m} \times \sqrt{m}$ single-hop WSN, a single column/row can be sorted in $2\sqrt{m}$ time slots.*

4 Sorting on Multi-hop WSN's

This section presents a sorting protocol for multi-hop WSN's. To begin with, the $\sqrt{n} \times \sqrt{n}$ array is partitioned into $\frac{\sqrt{n}}{2r} \times \frac{\sqrt{n}}{2r}$ groups of size $2r \times 2r$, which are further divided into 16 blocks of size $\frac{r}{2} \times \frac{r}{2}$. Note that there is a single-hop communication among the sensor nodes located within each block. Furthermore, a sensor node within a block can communicate with a sensor node in a neighboring adjacent block that occupies the same relative position within that block. We assume that n and r are power of two and that $\sqrt{n} > r$. The elements in a row or in a column can be sorted either in increasing or decreasing order. We say that an element is *rejected* if it is against the order in which the array is being sorted. The order in which the array is to be sorted is defined at a later stage in the main protocol. Our sorting protocol comprehends a number of sub-protocols whose details are discussed below.

4.1 Row and Column Sorting

We begin with a protocol that sorts a bitonic sequence of size κ, where the κ elements are stored in κ adjacent sensor nodes. The details of the protocol are spelled out as follows:

`Protocol Row-Merge`(κ)

1. Let S_i, $(1 \leq i \leq \kappa)$, be the sensor node that stores the element x_i. Also, let $P_1 = \{S_1, \cdots, S_{\kappa/2}\}$, and $P_2 = \{S_{\kappa/2+1}, \cdots, S_\kappa\}$;
2. **if** $\kappa = 2r$ **then** return;
3. Shift the elements from P_2 to P_1;
4. Perform a comparison-interchange on P_1;
5. Shift the rejected elements from P_1 to P_2;
6. In parallel, invoke `Row-Merge`$(\kappa/2)$ for P_1 and P_2;

In each iteration of the above protocol, a sequence of size $\kappa/2$ has to travel κ positions, $\kappa/2$ positions to the left and $\kappa/2$ positions to the right. Note that an element is ensured to be correctly received by a sensor node that is located $r/2$ positions from the sender, since the maximum distance between them does not exceed r. In order to avoid collision with other sensor nodes that are broadcasting at the same time, for each sequence of size $2r$, only one element is allowed to transmit in each time slot. Thus, $(\kappa/2)/2r = \kappa/4r$ elements, can travel simultaneously without interfering with each others' broadcast. Hence, a sequence of size $2r$ takes $(2r\kappa)/(r/2) = 4\kappa$ time slots to travel κ positions. Since we can move the elements of each sequence in parallel, it takes 4κ time slots to move $\kappa/2$ elements κ positions.

We now turn to number of iterations taken by Protocol Row-Merge(κ) to complete its execution. The protocol returns when $\kappa = 2r$, that is, the protocol will be executed for $\log \kappa - (\log r + 2)$ iterations. Thus, the total number of time slots can be computed by:

$$\sum_{i=\log r+2}^{\log \kappa} \frac{4\kappa}{2^{(\log \kappa)-i}} = 4\kappa \cdot \left[2 - \frac{1}{2^{\log \kappa - \log r - 2}}\right]$$

Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'02)
1087-4089/02 $17.00 © 2002 IEEE

$$= 4\kappa \cdot 2 - \frac{4r}{\kappa}$$
$$= 8\kappa - 16r$$

At this point, we have $\frac{\kappa}{2r}$ groups of sequences of size $2r$ that still need to be sorted. For this purpose, Protocol Row-Merge(κ) is slightly modified such that it can sort a row of elements of size $2r$. The main difference is that within each sequence of size $2r$ only one sensor is allowed to broadcast at a time. Consequently, we cannot perform Step 6 in parallel. On the other hand, Step 6 can be executed in parallel for neighboring sequences. Thus, when the size of the input sequence is reduced to $2r$, we can sort the sequence for each group sequentially until the size of the sequence is reduced to $r/2$, we then apply Corollary 1 to sort the remaining sequence. The following two steps need to be modified in Protocol Row-Merge(κ) to sort a bitonic sequence of size $2r$.

---

2. **if** $\kappa = r/2$ **then** return;

6. Invoke `Row-Merge`($\kappa/2$) for P_1 and P_2 sequentially;

Note that two iterations are sufficient to reduce the size of the sequence to $r/2$. In the first iteration, a sequence of size r is shifted 4 hops (two to the left, compare-interchange and shift back). In the second iteration, a sequence of size $r/2$ is shifted 2 hops, which takes $2r$ time slots since we have 2 of such sequences. Then, since a single-hop communication is ensured, a sequence of size $r/2$ can be sorted in r time slots according to Corollary 1. Thus, it takes $6r + 4r = 10r$ time slots to sort a bitonic sequence of size $2r$. Altogether, it takes $8\kappa - 6r < 8\kappa$ time slots to sort a bitonic sequence of size κ. The following lemma summarizes the above discussion:

Lemma 2 *The κ elements stored in κ adjacent sensor nodes on a row on the $\sqrt{n} \times \sqrt{n}$ array can be sorted into either increasing or decreasing row-major order in less than 8κ time slots.*

The Protocol Column-Merge(τ) is defined in a similar way, except that it sorts a bitonic sequence of size τ that is stored in τ adjacent sensors in a column of the $\sqrt{n} \times \sqrt{n}$ array.

Lemma 3 *The τ elements stored in κ adjacent sensor nodes on a column of the $\sqrt{n} \times \sqrt{n}$ array can be sorted into either increasing or decreasing column-major order in less than 8τ time slots.*

4.2 Vertical Merge Sort

The Protocol Vertical-Merge(τ, κ) sorts into either increasing or decreasing row-major order an array of size $\tau \times \kappa$ which is composed of two vertically aligned arrays of size $\tau/2 \times \kappa$, where one is in increasing row-major order and the other is in decreasing row-major order. It has been shown in [15] that two vertically aligned arrays can be sorted by column-merge followed by row-merge since all columns are bitonic, and after executing column-merge, all rows are bitonic. For further details, we refer the reader to [15].

`Protocol Vertical-Merge`(τ, κ)

1. **for** all columns in parallel **do** `Column-Merge`(τ);
2. **for** all rows in parallel **do** `Row-Merge`(κ);

The total number of time slots of the Protocol Vertical-Merge(τ, κ), accordingly to Lemma 3 and Lemma 4, is Vertical-Merge(τ, κ) $= 8(\tau + \kappa)$ time slots. Since we can only process one of the $2r$ lines/columns at a time, the total number of time slots is less than $16r(\kappa + \tau)$. The above results are summarized in the following lemma.

Lemma 4 *Two vertically aligned arrays of size $\tau/2 \times \kappa$, where one is in increasing row-major order and the other is in decreasing row-major order can be sorted in less than $16r(\kappa + \tau)$ time slots.*

4.3 Horizontal Merge Sort

The Protocol Horizontal-Merge(τ, κ) sorts an array of size $\tau \times \kappa$ which consists of two horizontally aligned adjacent arrays of size $\tau \times \kappa/2$. One of these arrays is sorted into increasing row-major order and the other into decreasing row-major order. Before showing the details of the Protocol Horizontal-Merge, we first introduce the Protocol TC-Merge(τ) (Two-Column-Merge), which sorts a bitonic sequence of 2τ elements stored in a column of τ adjacent sensors. The bitonic sequence $(x_1, \cdots, x_{2\tau})$ is stored in sensor S_i $(1 \leq i \leq \tau)$ such that each sensor holds two elements, x_i and $x_{i+\tau}$, of the bitonic sequence. After sorting, each sensor S_i will contain the elements x_{2i-1} and x_{2i}. The details of the protocol are listed below:

`Protocol TC-Merge`(τ)

1. Let S_i, $(i \leq 1 \leq \tau)$, be the sensor node that store the elements x_i and $x_{i+\tau}$. Also, let $P_1 = \{S_1, \cdots, S_{\tau/2}\}$, and $P_2 = \{S_{\tau/2+1}, \cdots, S_\tau\}$;
2. Compare-interchange the elements in each sensor;

3. **if** $\tau = 2r$ **then** return;

Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'02)
1087-4089/02 $17.00 © 2002 IEEE

4. Exchange the rejected elements of P_1 with the accepted elements of P_2;
5. In parallel, invoke TC-Merge($\tau/2$) for P_1 and P_2;

The proof of correctness of the above protocol can be found in [15]. The analysis of Protocol TC-Merge(τ) is similar to the Protocol Row-Merge, except that here the elements are exchanged instead of being shifted to the left and then to the right. Thus the total number of time slots for the Protocol TC-Merge(τ) is $8\tau - 16r$. Proceeding as we did before, another $6r$ time slots are necessary to reduce the size of the sequence from $2r$ to $r/2$. Recall that each sensor node now holds two elements. Hence sorting each column of size $r/2$ takes $2r$ time slots according to Corollary 1. Thus, Protocol TC-Merge(τ) takes, altogether, $8\tau - 2r < 8\tau$ time slots.

Lemma 5 *The τ elements stored in τ adjacent sensor nodes on a column of the $\sqrt{n} \times \sqrt{n}$ array can be sorted into either increasing or decreasing order in less than 8τ time slots.*

We now have all the necessary tools to present the protocol Horizontal Merge. The details of the protocol are as follows:

`Protocol Horizontal-Merge`(τ, κ)

1. Let $C_1, C_2, \cdots, C_\kappa$ represent the κ columns and also let $P_1 = \{C_1, \cdots, C_{\kappa/2}\}$, and $P_2 = \{C_{\kappa/2+1}, \cdots, C_\kappa\}$
2. Move the elements in P_2 to the corresponding sensors in P_1;
3. For each column $C_1, \cdots, C_{k/2}$ perform `TC-Merge` (τ);
4. Move the rejected elements in P_1 to the corresponding sensors in P_2;
5. In parallel, invoke `Row-Merge`($\kappa/2$) for each of the 2τ rows, each containing $\kappa/2$ adjacent sensors. The 2τ rows are obtained by splitting each original τ into two.

Recall that routing the elements κ positions, $\kappa/2$ to the left in step **2** and $\kappa/2$ to the right in step **4**, take 4κ time slots. The total number of time slots of Protocol Horizontal-Merge(τ, κ) is given by:

$$\begin{aligned} HM(\tau, \kappa) &= TC(\tau) + RM(\kappa/2) + 4\kappa \\ &= 8(\tau + \kappa), \end{aligned}$$

where HM, TC, and RM, stand for Horizontal-Merge, TC-Merge and Row-Merge, respectively. The following lemma summarizes the above results:

Lemma 6 *Two horizontally aligned arrays of size $\tau \times \kappa/2$, where one is in increasing row-major order and the other is in decreasing row-major order, can be sorted in less than $16r(\kappa + \tau)$ time slots.*

4.4 WSN-Sort

We are now in a position to show the sorting protocol that sorts n elements stored in n sensor nodes which are arranged in a two-dimensional array of size $\sqrt{n} \times \sqrt{n}$.

`Protocol WSN-Sort`($\sqrt{n}, \sqrt{n}$)

1. $\kappa \leftarrow 2r$;
2. Sort all groups of size $2r \times 2r$;
3. **while** $\kappa < \sqrt{n}$ **do**
4. Execute `Horizontal-Merge`($\kappa, 2\kappa$) in parallel for each array of size $k \times 2k$;
5. Execute `Vertical-Merge`($2\kappa, 2\kappa$) in parallel for each array of size $2\kappa \times 2\kappa$;
6. $\kappa \leftarrow 2 \cdot \kappa$;
7. **end while**

For the protocols Horizontal-Merge and Vertical-Merge to work properly, it is necessary to satisfy their initial conditions, that is, some subarrays must be sorted into increasing order and others into decreasing order. The order into which the array has to be sorted in steps **2** and **5** is defined by $\lfloor \frac{j-1}{\kappa} \rfloor$, and by $\lfloor \frac{i-1}{\kappa} \rfloor$ for step **4**, where i and $j, (1 \leq i, j, \leq \sqrt{n})$, represent the sensor's row and column indexes, respectively. If the result is even for all sensors on which comparison-interchanges are being executed, then the subarray is sorted into increasing row-major order, otherwise, it is sorted into decreasing row-major order. We now turn to analysis of the number of time slots taken by protocol Sorting. Clearly, step **2** can be computed in $O(r^2)$ time slots. The number of time slots for **while-loop** is is given by:

$$\begin{aligned} S(\sqrt{n}, \sqrt{n}) = S(\frac{\sqrt{n}}{2}, \frac{\sqrt{n}}{2}) &+ HM(\frac{\sqrt{n}}{2}, \sqrt{n}) \\ &+ VM(\sqrt{n}, \sqrt{n}) \\ &= S(\frac{\sqrt{n}}{2}, \frac{\sqrt{n}}{2}) + 56r\sqrt{n} \\ &\leq 112r\sqrt{n} \\ &= O(r\sqrt{n}), \end{aligned}$$

where S, VM, and HM, stand for WSN-Sort, Vertical-Merge, and Horizontal-Merge, respectively.

Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'02)
1087-4089/02 $17.00 © 2002 **IEEE**

Thus, for $r < \sqrt{n}$, the total number of time slots to sort an array of $\sqrt{n} \times \sqrt{n}$ elements, where each element is stored in a sensor node, is $O(r\sqrt{n})$ time slots.

Lemma 7 *Let a WSN consist of n elements stored in n sensor nodes, which are arranged in a two-dimensional array of size $\sqrt{n} \times \sqrt{n}$. When $\sqrt{n} > r$, the n elements can be sorted in $O(r\sqrt{n})$ time slots.*

When $r \geq \sqrt{n}$, the two-dimensional array will be already sorted after step **2**, and hence, it takes $O(r^2)$ time slots to sort the array. The following corollary summarizes this discussion.

Corollary 2 *Let a WSN consist of n elements stored in n sensor nodes, which are arranged in a two-dimensional array of size $\sqrt{n} \times \sqrt{n}$. When $r \geq \sqrt{n}$, the n elements can be sorted in $O(r^2)$ time slots.*

5 Conclusions

In this work we presented a sorting protocol for wireless sensor networks. The sorting protocol discussed in here is an adaptation of the parallel sorting algorithm proposed by Nassimi and Sahni [15], which is based on Batcher's bitonic sort algorithm [9]. Our protocol sorts n elements which are initially loaded in n sensor nodes arranged in a two-dimensional plane of size $\sqrt{n} \times \sqrt{n}$ in $O(r\sqrt{n})$ time slots without the need of involving the base station. We have also shown that future applications of wireless sensor networks are very likely to employ short-range radio communications (i.e., small r). If this is the case, our protocol matches the time complexity of the optimal sorting algorithm proposed in [15]. We have also shown an optimal sorting algorithm for single-hop WSN's. However, it remains to be shown whether or not our results are optimal when $1 \ll r \ll \sqrt{n}$.

References

[1] Abramson, N., *Multiple Access Communications: Foundations for Emerging Technologies*, IEEE Press, New York, 1993.

[2] Abramson, N., *Multiple access in wireless digital networks*, Proceedings of the IEEE, 82, (1994), 1360–1370.

[3] Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[4] Akl, S. G., *Parallel Sorting Algorithms*, Academic Press, Inc.,1985.

[5] Asada, G., Dong, M., Lin, T.S., Newberg, F., Pottie, Kaiser, W.J., and Marcy, H.O., *Wireless Integrated Network Sensors: Low Power Systems on a Chip*, Proceedings of the 1998 European Solid State Circuits Conference.

[6] Bar-Yehuda, R., Goldreich, O., and Itai, A., *Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection*, Distributed Computing, 5, (1991), 67–71.

[7] Bennett, F., Clarke, D., Evans, J. B., Hopper, A., Jones, A., Leask, D., *Piconet- Embedded Mobile Networking*, IEEE Personal Communications, Vol 4 No 5, October 1997, pp 8-15.

[8] Bertzekas, D., and Gallager, R., *Data Networks*, Second Edition, Prentice-Hall, 1992.

[9] Betcher, K. E., *Sorting Networks and Their Applications*, in Proc. AFIPS 1968 SJCC, vol. 32, Montvale, NJ:AFIPS Press, pp. 307-314.

[10] Bhuvaneswaran, R. S., Bordim, Jacir L., Cui, J., and Nakano, K., *Fundamental Protocols for Wireless Sensor Networks*, International Parallel and Distributed Processing Symposium (IPDPS), April 2001.

[11] Estrin, D., Govindan, R., Heidemann, J., and Kumar, S., *Next Century Challenges: Scalable Coordination in Sensor Networks.* In Proceedings of the Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99), Seattle, Washington, August 1999.

[12] John Heidemann and Nirupama Bulusu, *Using Geospatial Information in Sensor Networks*, In Proceedings of the Workshop on Intersections between Geospatial Information and Information Technology Arlington, VA, USA, National Research Council. October, 2001.

[13] Kaplan, E. D., *Understanding GPS: principles and applications*, Artech House, Greenwich, 1998.

[14] Miller, R., and Stout, Q. F., *Parallel Algorithms for Regular Architectures: Meshes and Pyramids*, The MIT Press, 1996.

[15] Nassimi, D. and Sahni, S., *Bitonic Sort on Mesh-Connected Computer*, IEEE Transactions on Computers, vol. c-27, NO. 1, January 1979.

[16] Thompson, C. D., and Kung, H. T., *Sorting on a Mesh-Connected Parallel Computer*, Communications of the ACM, vol 20, NO. 4, April 1977.

Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'02)
1087-4089/02 $17.00 © 2002 **IEEE**