Distributed Perimeter Detection in Wireless Sensor Networks

Fernando Martincic
Department of Computer Science
Wayne State University
Detroit MI (USA)
fernando@wayne.edu

Loren Schwiebert
Department of Computer Science
Wayne State University
Detroit MI (USA)
loren@wayne.edu

July 2, 2004

Abstract

This paper introduces a distributed localized algorithm where sensor nodes determine if they are located along the perimeter of a wireless sensor network. The algorithm works correctly in *sufficiently dense* wireless sensor networks with a minimal requisite degree of connectivity. Using 1-hop and 2-hop neighbour information, nodes determine if they are surrounded by neighbouring nodes, and consequently, if they are located within the interior of the wireless sensor network. The algorithm requires minimal communication between nodes - a desirable property since energy reserves are generally limited and non-renewable.

keywords: *wireless sensor network, distributed algorithms, perimeter detection*

1 Introduction

Advances in miniaturization and low-cost/low-power design have led to active research in large scale deployment of wireless sensor networks. These networks may consist of hundreds, and possibly thousands, of inexpensive disposable sensor nodes capable of sensing their environment and communicating with each other via wireless channels.

Although individually nodes possess limited functionality, inter-node cooperation and coordination makes applications such as monitoring of large areas viable [1, 2]. Areas affected by large-scale phenomena such as seismic disturbances, contaminant flows, and other ecological or environmental disasters can be tracked. In particular, determination when such phenomenon breaches the perimeter of the area monitored by the wireless sensor network is useful. Similarly, directed diffusion [5] is a novel data-centric communications paradigm that allows nodes in a wireless sensor network to perform distributed sensing of environmental phenomena. However, to work correctly, the perimeter of the wireless sensor network must be known *a priori*.

This paper introduces a distributed localized algorithm whereby sensor nodes determine if they are along the perimeter of the wireless sensor network. Nodes use location neighbourhood information to determine if they are enclosed by their neighbouring

nodes. Intuitively, nodes not enclosed lie along the outer edge of the wireless sensor network. The algorithm works correctly in wireless sensor networks that are *sufficiently dense*. In general, sensor nodes physically surrounded by other nodes do not lie along the perimeter of the wireless sensor network. However, since only local 2-hop neighbour information is used, sensor nodes with few neighbours may incorrectly conclude they are located along the perimeter. Thus, while perimeter nodes correctly identify themselves as lying along the outer edge of the wireless sensor network, nodes further inside the network with few neighbours, may incorrectly reach the same conclusion.

The remainder of the paper is organized as follows: Section 2 describes related research dealing with perimeter detection. Section 3 defines terminology used throughout this paper and describes the system model employed. Section 4 describes the algorithm in detail and motivates its discussion with a sample wireless sensor network topology. Limitations of the algorithm are discusses as performance metrics are established that demonstrate the algorithm's scalability as the number of nodes and node density increases. Section 5 presents concluding arguments and outlines future work.

2 Related Work

Chintalapudi *et al.* [4] examine three separate approaches to the problem of localized edge detection of a phenomenon boundary in a wireless sensor network. Using an event predicate, a node determines if it is part of the sub-region covered by the monitored phenomenon. Edges are defined in terms of the set of points in the spatial region that intersect with the interior and exterior areas of the phenomenon under observation.

Their first method is a statistical approach comprised of information gathered from neighbouring nodes, a set of statistics, $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$, that are computed based on the collected information, and a local boolean decision function, $\Psi(\Gamma_1, \Gamma_2, \ldots, \Gamma_n)$, that decides if the sensor lies on the edge of the observed phenomenon.

Their second method borrows from the sizable body of knowledge found in image-processing literature. In image-processing, a high-pass filter retains the high frequencies (i.e., abrupt changes such as edges) in the image and removes all uniformities. Such a filter is approximated and designed to work within the context of a wireless sensor network. Sensor nodes are analogous to pixels in the image, and thus, the filtering technique is applied. However, since sensors exhibit an irregularity of placement, a weighted averaging of the neighbourhood values is utilized to compensate.

Their final method is a classifier-based approach as used in pattern recognition. It relies upon information received from nodes within the phenomenon's interior region being significantly different than data gathered from exterior nodes. This bipartite data allows for classification into two distinct subsets where similar data are in one subset and dissimilar data are in another. The following steps are performed in the classifier-based approach: collect all coordinates and event predicate values within the probing radius, find a line that gives the maximal classifier score (used to partition the nodes with similar values on either side of the line), and if a node is within the radius of tolerance from the defined line, it is an edge sensor.

Nowak *et al.* [3] also propose a technique for edge detection of a phenomenon within a wireless sensor network. They consider measurements obtained from a collection of sensor nodes distributed throughout an area and determine the boundary that lies between two fields of relatively homogeneous measurements.

Their approach involves a hierarchical processing strategy where nodes collaborate to determine a non-uniform rectangular partition of the sensor field adapted to the boundaries of the phenomenon. The partitioning commences with normalization of the sensor field to a unit square with side lengths $1/\sqrt{n}$. The field is then subdivided using recursive dyadic partitioning whereby a region is subdivided into four equal regions, and each subregion thereafter is further subdivided into four equal regions until the entire sensor domain is partitioned into n squares. Nodes collaborate to determine a pruned partition that matches the phenomenon boundary. A final approximation is transmitted to the base station.

Our approach differs in that perimeter detection is independent of any observations made by the the wireless sensor network. No recorded sensor information is used in perimeter detection. Nodes attempt to discover if they lie along the outer edge of the wireless sensor network area strictly based on local neighbourhood information.

3 Model and Definitions

The wireless sensor network is represented by a connected graph $G_w = (V_w, E_w)$ where vertex set V_w denotes the set of all nodes in the wireless sensor network. Edge set E_w contains edge $e = (u, v)$ iff nodes $u, v \in V_w$ communicate directly with each other. It is assumed vertices are uniquely identifiable and no two vertices share the same global coordinates. Furthermore, the following assumptions are made:

(a) A large area is covered by several uniquely identifiable homogeneous sensor nodes that utilize short range radios to communicate with each other.

(b) Sensor nodes are location-aware. This is achieved using GPS, or localization techniques as presented in [6, 7, 8].

(c) Communication between nodes is bidirectional with transmission over long ranges achieved via multiple hops between sensors. Nodes within transmission range are able to communicate with each other.

(d) Messages may be delayed for an arbitrary amount of time, but all messages sent are eventually received and are assumed to be error-free.

The algorithm presented makes use of local neighbourhood information. In particular, nodes only use immediate neighbour and indirect neighbour information during computation.

Definition 3.1 Two nodes $u, v \in V_w$ are *immediate neighbours* iff $(u, v) \in E_w$. The set of all immediate neighbours of node u is denoted by $N_1(u)$.

Definition 3.2 Two nodes $u, v \in V_w$ are *indirect neighbours* (i.e., 2-hop neighbours) iff $\exists u' \in V_w : (u, u') \in E_w \wedge (u', v) \in E_w)$. The set of all indirect neighbours of node u is denoted by $N_2(u)$.

Figure 1 depicts a sample wireless sensor network with 16 nodes. Edges between nodes indicate a shared bidirectional communication link. Node 8 has a valid enclosing cycle $6, 2, 5, 10, 14, 11, 9, 6$ that forms a closed simple polygon, composed of 1-hop and 2-hop neighbours of the node, that encloses it within its bounded interior region.

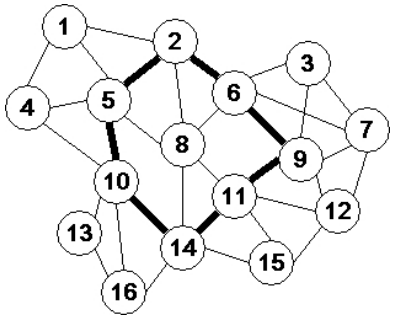

Figure 1: Example of a Valid Enclosing Cycle (bold lines)

Detection of valid enclosing cycles is the basis for ascertaining if a node is not located along the perimeter of the wireless sensor network. Its properties are as follows:

Definition 3.3 A cycle $s = u_1, u_2, \ldots, u_m, u_1$, where $u_i \in V_W$, that forms a closed simple polygon, is a *valid enclosing cycle* of node v, denoted $VEC(v, s)$, if it satisfies the following criteria:

(a) $\forall u_i \in s : u_i \in N_1(v) \vee u_i \in N_2(v)$

(b) $\forall u_i \in s : u_i \in N_2(u) \rightarrow (u_{i-1} \in N_1(v) \wedge u_{i+1} \in N_1(v))$

(c) $(u_i, u_{i+1}) \in E_w$, where $1 \leq i < m$

(d) $(u_1, u_m) \in E_w$

(e) $u_i.angle < v_{i+1}.angle$, where $1 \leq i < m$

(f) node v is contained within the interior region of the induced polygon.

Intuitively, nodes surrounded by other nodes are not on the perimeter of the wireless sensor network. These nodes are referred to as interior nodes. Nodes not enclosed by any valid enclosing cycle are perimeter nodes.

Definition 3.4 A node $v \in V_w$ is an *interior node* iff there exists a valid enclosing cycle s that contains node v within its bounded interior region. Node v is a *perimeter node* iff it is not an interior node.

4 Perimeter Detection

This section describes the algorithm used to detect perimeter and interior nodes. Discussion of the algorithm is motivated with an example. Consider the sample wireless sensor network in Figure 2. Nodes are numbered with unique node IDs and randomly distributed along the grid lines of a 10×10 unit square grid with origin $(0, 0)$ located in the lower-left corner. Coordinates are absolute global coordinates and nodes have a transmission radius of 3 units. Consider node 11 located at coordinates $(4, 4)$. After all nodes have transmitted their location and neighbour information, Figure 3 represents node 11's 1-hop neighbour list sorted by increasing relative angle to node 11. Two nodes with the equal relative angles to node 11 are further sorted by their distance from the node (e.g., node 12 precedes node 13 since it is closer to node 11.)

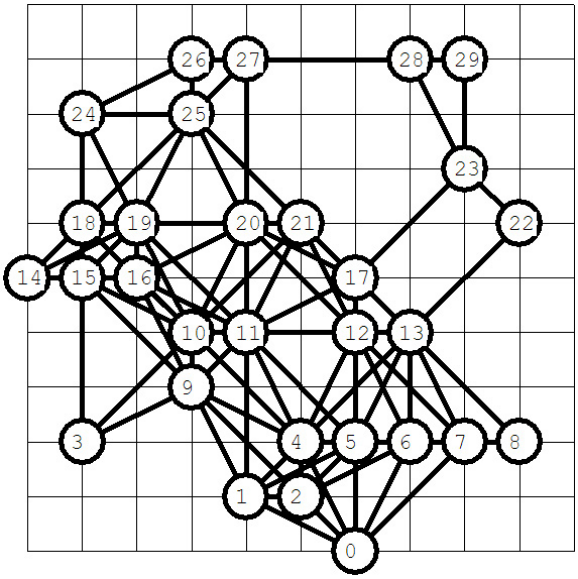

Figure 2: Sample Wireless Sensor Network

Node	Location	Relative Angle	Number 2-hop neighbours
12	(6,4)	0	10
13	(7,4)	0	10
17	(6,5)	26	7
21	(5,6)	63	8
20	(4,6)	90	10
19	(2,6)	136	10
16	(2,5)	154	8
10	(3,4)	180	11
9	(3,3)	224	8
1	(4,1)	270	6
4	(5,2)	297	11
5	(6,2)	316	11

Figure 3: 1-hop Neighbour List for Node 11

Node 11 also maintains a list of its 2-hop neighbours. For each entry in its 1-hop neighbour list, node 11 maintains a list of the node's 1-hop neighbour list. Relative angles of each 2-hop neighbour to node 11 are calculated and stored in this list. Figure 4 is a partial listing the 2-hop neighbour information maintained by node 11.

Using local 1-hop and 2-hop neighbour information, node 11 attempts to detect if a valid enclosing cycle exists that forms a closed simple polygon that encloses the node within its bounded interior region. It begins by setting the first node in its 1-hop neighbour list, in this case node 12, as an *anchor node* and pushes it onto a global stack variable. The (current) anchor node is the start and end node in every valid enclosing cycle. Node 11 then checks if node 12 communicates directly (or via a common 2-hop neighbour) with another of its 1-hop neighbours with a greater relative angle than node 12. Node 12's 1-hop neighbour list is searched exhaustively, and in this case, node 17 is the first 1-hop neighbour that satisfies this criteria.

Node 17 is pushed onto the stack and another 1-hop neighbour of node 11, with

Node 12	Loc.	∠ to Node 11
13	(7,4)	0
17	(6,5)	90
21	(5,6)	117
20	(4,6)	136
11	(4,4)	180
10	(3,4)	180
4	(5,2)	243
5	(6,2)	270
6	(7,2)	297
7	(8,2)	316

Node 13	Loc.	∠ to Node 11
22	(9,6)	21
17	(6,5)	26
21	(5,6)	63
12	(6,4)	0
11	(4,4)	270
4	(5,2)	297
5	(6,2)	316
6	(7,2)	327
7	(8,2)	334
8	(9,2)	339

Node 17	Loc.	∠ to Node 11
23	(8,7)	36
21	(5,6)	63
20	(4,6)	90
11	(4,4)	270
12	(6,4)	0
5	(6,2)	316
13	(7,4)	0

Figure 4: Partial 2-hop Neighbour List for Node 11

a greater relative angle than node 17, is found that communicates directly (or via a common 2-hop neighbour) with node 17. The first node to satisfy this criteria is node 21. Node 21 is pushed onto the stack and another 1-hop neighbour of node 11 with a relative angle greater than node 21 that satisfies similar criteria is searched. This depth-first search process continues until a cycle is detected or node 11's 1-hop neighbour list is exhausted. Eventually, node 11 detects a valid enclosing cycle composed of nodes $12, 17, 21, 20, 19, 15, 9, 1, 5, 12$.

Since every node in a valid enclosing cycle communicates directly with the preceding and subsequent nodes, a cycle $s = v_1, v_2, \ldots, v_m$ generates a subgraph $G_c = (V_c, E_c)$, where $V_c = \{v_1, v_2, \ldots, v_m\}$ consists of the the set of nodes in the cycle and edge set $E_c = \{(v_i, v_{i+1}) \mid 1 \leq i < m\} \bigcup \{(v_m, v_1)\}$ consists of the set of edges that join two consecutive nodes in the cycle together. Clearly, $G_c \subseteq G_w$ and defines a simple closed polygon that is analyzed to see if it encloses the node. In this case it does, and consequently, node 11 is an interior node.

If no cycle is detected with the current anchor node, the stack is cleared, the subsequent node in the 1-hop neighbour list is set as the anchor node (i.e., node 13), and the process is repeated. Only nodes with relative angles between 0 and 180 degrees are set as anchor nodes. This is because enclosing polygons must contain at least 3 nodes where at least one node lies above the node and at least one node lies below the node.

4.1 Perimeter Detection Algorithm

Figure 5 is the (simplified) pseudocode for the perimeter detection algorithm. A random backoff timer *BackoffTime* is initialized and used to help prevent nodes from broadcasting their location information simultaneously. *Timer* is the variable compared against the backoff timer to determine when the next transmission occurs. Variable *NL* is a list

of the node's 1-hop neighbours, sorted in ascending order by relative angle to the node. *OnPerimeter* is a boolean variable, set to *true*, if the node is on the perimeter of the wireless sensor network. It is *false* otherwise.

```
      // for every node in the wireless sensor network
1     BackoffTime = random number > 0
2     Timer := 0; NL := {}; OnPerimter := true; GetGlobalCoordinates()
3     do
4        Timer := Timer + 1
5        if msg received from node v then
6           if v ∉ NL then
7              create node u
8              u.id := v.id; u.x := v.x; u.y := v.y
9              u.angle := CalculateAngle(u.x,u.y); NL.InsertNode(u)
11          fi
12          u := NL.FindNode(v.id); u.CopyNeighbourList(v)
14          OnPerimeter := IsPerimeterNode()
15       fi
16       if Timer > BackoffTime then
17          Broadcast(); set new random BackoffTime; Timer := 0
19       fi
20    od
```

Figure 5: Perimeter Detection Algorithm

Variables i, j, k, l, u and v represent nodes. Variable *NL* is the 1-hop neighbour list maintained by the node. For an arbitrary node u, its ID, x-coordinate, y-coordinate, and relative angle are denoted $u.id$, $u.x$, $u.y$, and $u.angle$, respectively. The relative angle is calculated using trigonometry and the Euclidean distance between both nodes. Figure 6 demonstrates the difference between upper and lower nodes. Nodes with a relative angle between 0° and 180° are *upper* nodes (shaded region). Conversely, nodes with a relative angle between 181° and 360° are *lower* nodes. Variable *anchor* is the 1-hop neighbour node that is the first node in the current cycle being constructed. To ensure detected cycles begin and end with the anchor node, its relative angle is incremented by 360°, as necessary, to ensure its relative angle is greater than any lower node in the cycle.

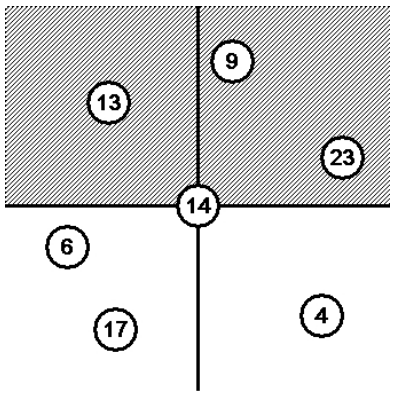

Figure 6: Lower and Upper Nodes

GetGlobalCoordinates determines the node's global coordinates [6, 7, 8]. *CalculateAngle* approximates and returns the relative angle of a node to the current node.

For example, consider a node located at global coordinates $(1, 1)$. Assume it has a 1-hop neighbour located at global coordinates $(3, 2)$. Thus, the relative angle returned by the function is 27°. *InsertNode* inserts a node into neighbour list *NL* so that nodes in it are sorted by increasing relative angle. *FindNode* returns the node with the corresponding node id passed as an argument to it. *CopyNeighbourList* updates the neighbour list of a node by first clearing out its existing neighbour list and replacing it with the neighbour list of the node passed as an argument. Finally, *IsPerimeterNode* determines if the node is located along the perimeter of the wireless sensor network.

The first few lines in the algorithm initialize the data structures perform location discovery, via the call to *GetGlobalCoordinates*. The algorithm then enters a loop that processes received messages from other nodes and periodically broadcasts its location and 1-hop neighbour information (initially empty) to nearby sensor nodes. Upon reception of a broadcast message from a newly discovered neighbour, the receiving node copies the sending node's information and neighbour list, calculates its relative angle to the node, and inserts it into neighbour list *NL*. If the node is already present in *NL*, the sending node's 1-hop neighbour list information is updated locally to reflect any changes. *IsPerimeterNode* (Figure 7) is then invoked which returns a true value if the node lies along the perimeter of the wireless sensor network. The function first ensures at least three immediate neighbours exist. It then systematically selects an upper node from *NL*, sets it as the current anchor node, and pushes it onto a global stack. Upper 1-hop neighbours in *NL* are set as anchor nodes, in turn, until a cycle is found or all upper nodes in *NL* are exhausted.

```
    bool IsPerimeterNode()
1     if |NL| < 3 then return true fi
2     for each node i ∈ NL
      where i.angle ≤ 180
3       stack.clear()
4       anchor := i
5       stack.push(i)
6       if FindCycleRec() then
        return false fi
7     rof
8     return true
    end IsPerimeterNode
```

(a)

```
    bool FindCycleRec() {
1     node i = stack.top()
2     if i =anchor then
3       if stack.size() > 3 then
4         if VerifyCycle() then
          return true fi
5       fi
6     fi
7     if i.angle ≤ 180° then
8       Case 1: (See Figure 8)
        // 1-hop neighbour with
        a relative angle ≤ 180°
9     else
10      Case 2: (See Figure 9)
        // 1-hop neighbour with
        relative angle > 180°
11    fi
12  return false
    end FindCycleRec()
```

(b)

Figure 7: Functions IsPerimeterNode and FindCycleRec

Recursive function *FindCycleRec*, where the bulk of the processing occurs, is called from *IsPerimeterNode*. It performs a brute-force depth-first search through the node's 1-hop and 2-hop neighbour lists to determine if a valid enclosing cycle exists. The search begins with the anchor node and continues with a counterclockwise sweep of the surrounding interconnected 1-hop and 2-hop neighbours until anchor node is reached again or all nodes are exhausted. If a cycle is detected (i.e., the anchor node is reached

again), it is verified to ensure the node is contained within the interior of the corresponding polygon. Clearly, a minimum of three nodes are necessary to enclose any node. Furthermore, at least one upper node and one lower node are required in every valid enclosing cycle.

FindCycleRec examines the node at the top of the stack and copies it to local variable i. If node i is the anchor node and more than three nodes are in the node stack, the detected cycle is verified to determine if it is a valid enclosing cycle. If the cycle encloses the node, the function returns true and the node is an interior node. Otherwise, two cases are distinguished: node i is an upper node or node i is a lower node.

If node i is an upper node (Figure 8(a)), its 1-hop neighbour list is searched to see if there exists a node j, with a greater relative angle than node i. Node j is not allowed to be the node trying to be enclosed, be present in the node stack, or be the anchor node (otherwise the ensuing cycle consists entirely of upper nodes). If node j is a 1-hop neighbour of the (to be enclosed) node, it is pushed on the stack and *FindCycleRec* is recursively invoked. Otherwise, node j is a 2-hop neighbour and two additional subcases are considered (Figures 8(b) and (c)). If all neighbour nodes are exhausted without finding a valid enclosing cycle, a *false* value is returned.

In subcase 1a (Figure 8(b)), node j is a upper 2-hop neighbour of the node to be enclosed. Neighbour list *NL* is searched to find if there exists a 1-hop neighbour node k, with a greater relative angle than node i that is connected to node j. Node k must also have a greater relative angle than node j. If a node k is found that satisfies this criteria, it is checked to ensure it is not the anchor node. This is so cycles consisting strictly of upper nodes are avoided. Node j and node k are pushed on the stack and *FindCycleRec* is recursively invoked. In subcase 1b (Figure 8(c)), node j is a lower 2-hop neighbour node. Neighbour list *NL* is searched to find if a node k, connected to node j, with a greater relative angle than node i and node j exists. In this case, at least one lower node is part of the cycle. Therefore, node k may be the anchor node. If a suitable node k is found, node k and node j are pushed on the stack and *FindCycleRec* is recursively invoked.

In case 2 (Figure 9(a)), node i is a lower 1-hop neighbour node. Node i's neighbour list is searched to ensure the node trying to be enclosed is discarded. It then checks to see if the node communicates directly with the anchor node. If it does, the anchor node is pushed onto the stack and *FindCycleRec* is recursively invoked, where the detected cycle will be verified (since the top node in the stack is the anchor node). Otherwise, node i's neighbour list is searched to find if a node j, with a greater relative angle than node i can communicate directly with node i. If a 1-hop neighbour node is found that satisfies this criteria, it is pushed onto the stack and *FindCycleRec* is recursively invoked. Otherwise, node j is a 2-hop neighbour node and a special subcase (Figure 9(b)) must be considered where node j is possibly an upper node connected to the anchor node.

In subcase 2a (Figure 9(b)), node j is a 2-hop neighbour node of the node to be enclosed. If node j is an upper node, it is verified to ensure it is directly connected to the anchor node. If it is, then node j and the anchor node are pushed onto the stack and *FindCycleRec* is recursively invoked. Otherwise, node j is a lower 2-hop neighbour node and a search is done for a node k that is a 1-hop neighbour node with a greater relative angle than node i and node j. If such a node k is found, both node j and node k are pushed on the stack and *FindCycleRec* is recursively invoked.

4.2 Limitations

Consider the sample wireless sensor network depicted in Figure 10. Shaded nodes represent interior nodes after the algorithm has executed.

Nodes 17 and 21 have erroneously concluded they are perimeter nodes. This is an inherent limitation of detecting valid enclosing cycles with only 1-hop and 2-hop neighbour information. Clearly, the simple polygon induced by cycle $23, 28, 27, 20, 12, 13, 22, 23$ encloses node 17 within its bounded interior region and the simple polygon induced by cycle $17, 23, 28, 27, 20, 17$ encloses node 21 within its bounded interior region. However, the connection between node 27 and 28 is not determinable in either case based on the local information stored at either node.

This limitation gives rise to the notion of a *sufficiently dense* wireless sensor network. Interior nodes must be positioned such that there exist at least two distinct paths from a node to its 2-hop neighbours in the underlying graph of the wireless sensor network. When this condition exists, enclosing cycles are properly detected, and consequently, nodes correctly assess if they are located along the perimeter of the wireless sensor network.

4.3 Performance Analysis

Testing was done with a custom simulation testbed written in C++ with statistical data gathered to analyze the runtime performance and scalability of the algorithm. Several parameters were varied for each set of trials. These include the size (expressed as a length and width) of the sensor area network, sensor node transmission radius (expressed as normalized units), and the total number of nodes in the wireless sensor network. For every set of parameters, ten trial runs were conducted and the results averaged. Nodes were randomly distributed along grid points throughout the sensor field with at most a single node at any grid line intersection. Thus, a sensor field with a length of 10 units and a width of 10 units has a 100 sensor node capacity. This model allows for the simulation to be executed for increasingly denser wireless sensor networks. Statistical data collected from the system included the total number of 1-hop and 2-hop neighbours, the total number of bits transmitted and received in the system, and the total number of instructions executed by all sensor nodes in the wireless sensor network. This data was averaged and *per node* results were calculated.

Figure 11 displays the average number of bits transmitted per node averaged for a transmission radius between 2 and 4 units. As the wireless sensor network density increases, there is a marked increase in the average number of bits transmitted. This is due to the increased number of 1-hop and 2-hop neighbours present for each node. As the wireless sensor network area increases, the average number of bits transmitted per node remains constant when the sensor node density remains constant. This is attributable to the localized and distributed characteristics of the algorithm. The results indicate the highly scalable nature of the algorithm.

Figures 12 and 13 display the average number of bits received per node and the number of operations performed per node, respectively, averaged for a transmission radius between 2 and 4 units. The results are similar to the average number of bits transmitted with an marked increase in bits received and operations performed as the density of the wireless sensor network increases, and a constant number of bits received and operations performed as the area of the sensor network increases but the sensor

node density remains constant.

The number of bits transmitted and received give a lower bound for the amount of data relayed. In the simulation, two rounds of communication take place. In the first round, neighbour discovery takes place whereby nodes announce their presence and broadcast their location information. In the second round, nodes gather 1-hop neighbour information collected in the neighbour discovery and transmit this data to their neighbours. Consequently, all nodes in the wireless sensor network have complete 1-hop and 2-hop neighbour information. Since it is assumed nodes within transmission radius of a broadcasting node receive all broadcast messages flawlessly, no retransmissions are required.

Broadcast messages consist of a unique 16-bit node ID, 16-bit global x-coordinate, 16-bit y-coordinate, and an 8-bit quantity that represents the number of the transmitting node's 1-hop neighbours. For each immediate neighbour, its 16-bit node ID, 16-bit x-coordinate, 16-bit $y - coordinate$, and an 8-bit quantity indicating the number of (2-hop) neighbours it has is sent. The neighbour list of each 1-hop neighbour (i.e., 2-hop neighbour information) is also transmitted with a 16-bit node ID, 16-bit x-coordinate, and 16-bit y coordinate sent for each (2-hop) neighbour. Finally, a 16-bit CRC is appended to every broadcast message for error-checking purposes.

The instructions metric is an estimate of the number of operations executed during computation. Nodes increment local counters on every iteration through their data structures and whenever a comparison or calculation (i.e., relative angle) is performed. Therefore, the results are consistent across different simulation runs and provide an indication of the amount of work done by individual sensor nodes as the parameters vary.simulation runs and provide an indication of the amount of work done by individual sensor nodes as the parameters vary.

5 Conclusions

A distributed localized algorithm for perimeter detection of a wireless sensor network was presented in this paper. Metrics were established and analyzed for the algorithm in order to establish its performance and scalability. The algorithm exhibits desirable characteristics well-suited to wireless sensor networks. Specifically, the highly localized and distributed nature of the algorithm minimizes the amount of data maintained locally and the quantity of data relayed between sensor nodes. This is critical in wireless sensor networks due to the inherent energy constraints present.

The limitation of the algorithm where nodes incorrectly determine that they lie along the perimeter of the sensor network when neighbouring nodes are sparse is currently under research. Unfortunately, the use of centralized techniques quickly becomes infeasible in large wireless sensor networks due to the combinatorial explosion of sensor nodes to be examined as the size of the network increases. A scalable localized distributed algorithm to correctly detect enclosing cycles in all cases is left as an open problem.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. *Wireless Sensor Networks: A Survey.* Computer Networks, 38(4):393422, March 2002.

[2] G.J. Pottie and W.J. Kaiser. *Wireless Integrated Network Sensors*. Communications of the ACM, vol.43, pp. 51-58, May 2000.

[3] R. Nowak and U. Mitra. *Boundary Estimation in Sensor Networks: Theory and Methods*. Proceedings of the First International Workshop on Information Processing in Sensor Networks, April 2003.

[4] K. K. Chintalapudi and R. Govindan. *Localized Edge Detection in Sensor Fields*. In IEEE International Workshop on Sensor Network Protocols and Applications, pp. 59-70, May 2003.

[5] C. Intanagonwiwat, R. Govindan, and D. Estrin. *Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks*. ACM International Conference on Mobile Computing and Networking (MOBICOM'00), 2000.

[6] D. Niculescu and B. Nath. *Ad-Hoc Positioning System (APS)*. Proceedings of GLOBECOM, San Antonio, November 2001.

[7] J. Albowitz, A. Chen, and L. Zhang. *Recursive Position Estimation in Sensor Networks*. ICNP'01, 2001.

[8] Andreas Savvides, Chih-Chieh Han, Mani B. Strivastava. *Dynamic fine-grained localization in Ad-Hoc networks of sensors*. Proceedings of the seventh annual International Conference on Mobile Computing and Networking, July 2001.

A Compiled Simulation Data

The data presented in the subsequent charts is the aggregated data averaged for each trial run. In particular, the number of bits transmitted per node, the number of bits received per node, and the number of instructions executed per node are complied for sensor networks of increasing density that cover an area 10×10, 20×20, 30×30, 40×40, and 50×50 units in size.

```
1    for each node j ∈ i.NL
2      if j = this ∨ j ∈ stack ∨ j = anchor then continue fi
5      if j ∈ NL
         if j.angle ≥ i.angle then
6          stack.push(j)
7          if FindCycleRec() then return true fi
         fi
10     else // node j is a 2-hop neighbour
12       if j.angle ≤ 180° then
13         Case 1a: Figure 8(b) // upper 2-hop neighbour
16       else
17         Case 1b: Figure 8(c) // lower 2-hop neighbour
20       fi
21     fi
22   rof
```

(a)

```
//   node j is a 2-hop upper node
1    for each k ∈ NL where
     k.angle ≥ i.angle
2      if k ∈ i.NL ∨ k.angle < j.angle
       then continue fi
3      for each l ∈ k.NL
4        if j = l ∧ k ≠ anchor then
5          stack.push(j); stack.push(k)
6          if FindCycleRec() then
             return true fi
7          stack.pop(); stack.pop()
8        fi
9      rof
10   rof
```

(b)

```
//   node j is a 2-hop lower node
1    for each k ∈ NL where
     k.angle > i.angle
2      if k ∈ i.NL ∨ k.angle < j.angle
       then continue fi
3      for each l ∈ k.NL
4        if j = l then
5          stack.push(j); stack.push(k)
6          if FindChainRec() then
           return true fi
7          stack.pop(); stack.pop()
8        fi
9      rof
10   rof
```

(c)

Figure 8: Case 1: Node i is an Upper Node

```
1    for each j ∈ i.NL where
     j.angle > i.angle
2      if j =this then continue fi
3      if j =anchor then
4        stack.push(j)
5        if FindCycleRec() then
         return true fi
6      fi
7      if j ∈ stack then continue fi
8      if j ∈ NL then
9        stack.push(j)
10       if FindCycleRec() then
         return true fi
11       stack.pop()
12     else
13       Case 2a: (See Figure 9(b))
         // j is a 2-hop neighbour
14     fi
15   rof
```

(a)

```
1    if j.angle ≤ 180 then
2      if j.angle < anchor.angle ∧
       j ∈ anchor.NL then
3        stack.push( j ); stack.push(anchor)
4        if FindCycleRec() then
         return true fi
5        stack.pop(); stack.pop()
6      fi
7      continue (next iteration)
8    fi
9    for each k ∈ NL where k.angle > i.angle
10     if k ∈ i.NL ∨ k.angle < j.angle then
       continue fi
11     for each l ∈ k.NL
12       if j = l then
13         stack.push(j); stack.push(l)
14         if FindCycleRec() then
           return true fi
15         stack.pop(); stack.pop()
16       fi
17     rof
18   rof
```

(b)

Figure 9: Case 2: Node i is a Lower Node

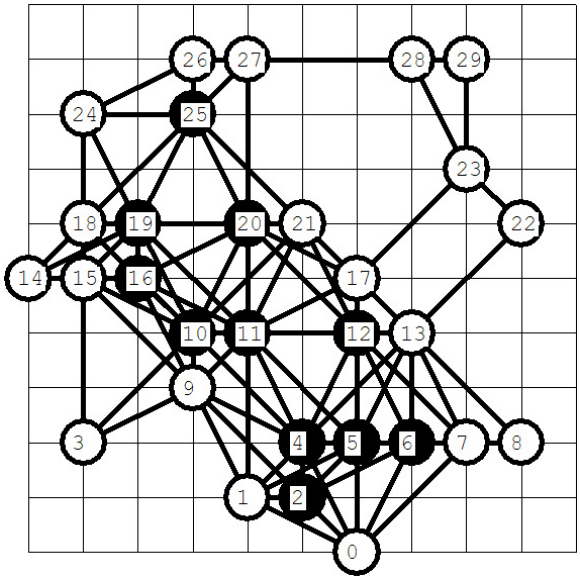

Figure 10: Shaded Interior Nodes in Sample Wireless Sensor Network

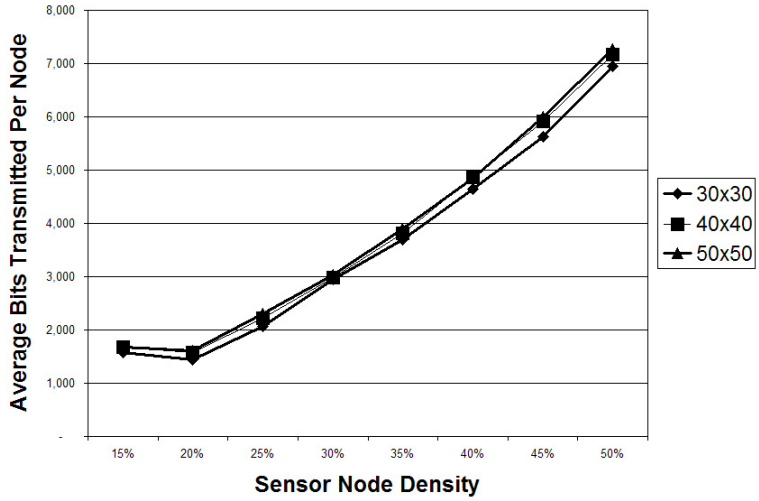

Figure 11: Average Number of Bits Transmitted Per Node

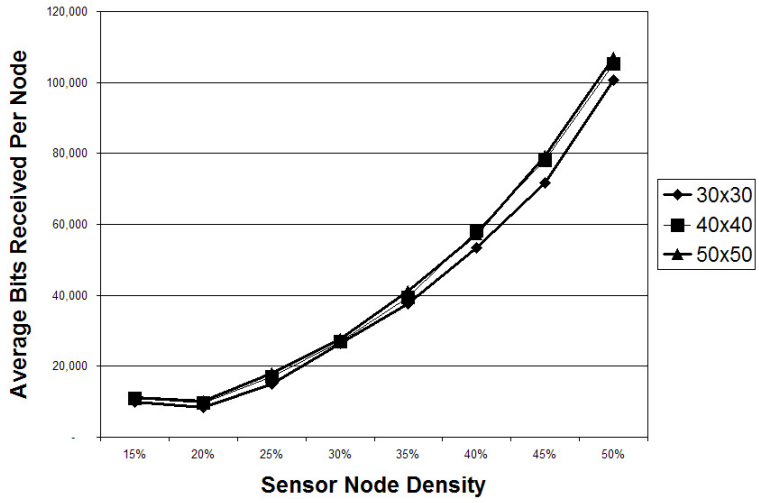

Figure 12: Average Number of Bits Received Per Node

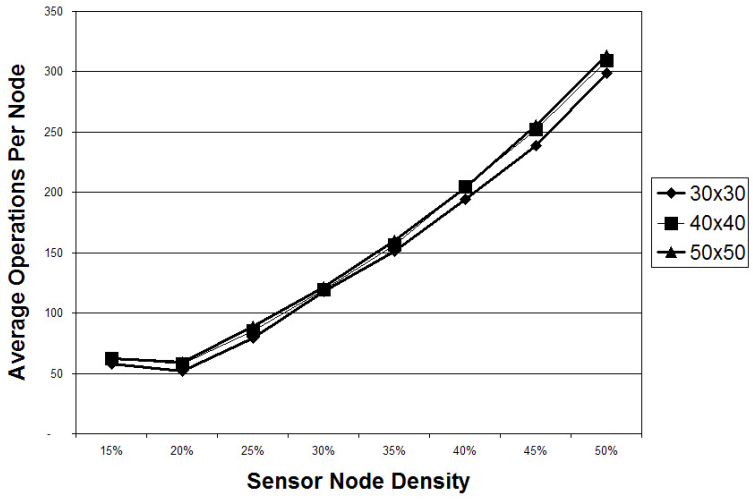

Figure 13: Average Number of Operations Performed Per Node

Radius	Nodes	10	15	20	25	30	35	40	45	50
2	Trans.	256	396	439	586	701	876	1,433	1,672	1,846
	Rec.	277	727	866	1,558	2,065	2,955	7,009	8,973	10,304
	Op.	5	9	10	16	20	27	51	61	69
3	Trans.	660	958	1,654	2,041	2,597	3,489	4,594	5,875	6,699
	Rec.	1,891	3,640	8,886	12,854	17,973	28,600	44,624	65,303	78,206
	Op.	19	31	60	77	101	141	190	248	285
4	Trans.	570	1,530	2,776	4,539	6,080	7,600	10,150	12,552	14,937
	Rec.	1,389	7,707	20,312	43,241	67,975	94,869	148,259	203,350	265,147
	Op.	15	54	108	187	256	324	439	549	659

(a) 10×10 Sensor Field

Radius	Nodes	40	60	80	100	120	140	160	180	200
2	Trans.	374	626	714	774	989	1,351	1,566	1,892	2,227
	Rec.	708	1,875	2,338	2,617	3,798	6,432	8,097	10,876	13,925
	Op.	9	18	21	24	32	47	57	71	85
3	Trans.	893	1,287	1,775	2,664	3,617	4,753	6,140	7,332	9,026
	Rec.	3,412	6,043	10,174	18,760	30,460	46,351	68,961	90,002	123,135
	Op.	29	45	65	104	146	197	260	315	392
4	Trans.	1,481	2,709	4,109	6,391	8,490	11,024	14,879	18,190	22,839
	Rec.	7,321	19,456	37,391	74,137	113,179	166,646	266,500	359,546	509,511
	Op.	52	106	167	270	365	481	658	810	1,025

(b) 20×20 Sensor Area

Radius	Nodes	90	135	180	225	270	315	360	405	450
2	Trans.	360	499	621	859	1,100	1,354	1,667	1,977	2,312
	Rec.	652	1,225	1,764	3,044	4,614	6,368	8,895	11,584	14,740
	Op.	8	13	18	27	37	47	61	74	89
3	Trans.	812	1,382	2,010	2,846	4,148	5,081	6,432	7,873	9,672
	Rec.	2,979	7,043	12,353	21,109	38,247	51,277	73,251	99,251	135,599
	Op.	25	49	75	112	170	212	273	339	422
4	Trans.	1,528	2,863	4,068	6,395	9,283	13,119	16,449	19,753	25,946
	Rec.	8,137	21,590	35,433	71,895	127,443	215,947	301,043	99,726	612,051
	Op.	55	112	165	270	402	577	730	883	1,168

(c) 30×30 Sensor Area

Radius	Nodes	160	240	320	400	480	560	640	720	800
2	Trans.	365	507	673	912	1,128	1,404	1,696	2,014	2,376
	Rec.	674	1,252	2,033	3,444	4,782	6,828	9,104	11,882	15,392
	Op.	8	13	20	29	38	50	62	76	92
3	Trans.	833	1,389	2,130	2,961	4,122	5,266	6,756	8,238	9,968
	Rec.	3,041	6,954	13,494	22,339	37,323	53,776	78,974	106,039	141,064
	Op.	26	49	81	117	169	220	288	356	435
4	Trans.	1,642	3,124	4,987	7,818	10,082	13,386	17,670	21,634	26,944
	Rec.	9,065	24,748	50,200	101,438	146,522	222,527	341,341	459,237	643,795
	Op.	59	124	207	335	438	589	786	969	1,216

(d) 40×40 Sensor Area

Radius	Nodes	250	375	500	625	750	875	1,000	1,125	1,250
2	Trans.	358	495	675	910	1,142	1,394	1,716	2,025	2,383
	Rec.	632	1,178	2,070	3,413	4,879	6,689	9,269	11,975	15,361
	Op.	8	13	20	29	39	49	63	76	92
3	Trans.	795	1,362	2,213	3,137	4,204	5,371	6,715	8,433	10,155
	Rec.	2,756	6,629	14,488	24,575	38,270	55,408	77,360	109,450	144,871
	Op.	24	48	84	125	173	225	286	365	444
4	Trans.	1,639	3,182	5,016	7,772	10,310	14,165	17,635	21,525	27,307
	Rec.	8,958	25,413	49,929	99,394	150,437	242,239	336,792	452,547	647,866
	Op.	59	126	208	333	448	626	784	965	1,231

(e) 50×50 Sensor Area