OCO: Optimized Communication & Organization for Target Tracking in Wireless Sensor Networks

Sam Phu Manh Tran T. Andrew Yang (yang@uhcl.edu)
University of Houston – Clear Lake
2700 Bay Area Blvd., Houston, Texas 77058, USA

Abstract

Wireless sensor networks (WSN) have affected various military and civilian applications. Target tracking is one of the top applications of WSNs. Efficient computation and energy dissipation are critical requirements for a WSN when used in target tracking. The goals are to maximize the network's lifetime while ensuring accuracy. Existing methods such as the LEACH-based algorithms suffer either complex computations or redundant data/sensor deployment. Those drawbacks result in inefficient energy use and/or computation overhead. Optimized Communication and Organization (OCO) is a method that provides self-organizing and routing capabilities. OCO ensures maximum accuracy, efficient energy dissipation, and low computation overhead. We have conducted simulation-based evaluations to compare the performance of OCO against LEACH and Direct Communication (DC), under various scenarios. This paper discusses OCO, the simulation set-up, the performance metrics, and the analysis of the results.

1. Introduction

Wireless sensor networks (WSNs) have significant impact on the efficiency of military and civilian applications, such as environment monitoring, target surveillance, industrial process observation, tactical systems, etc. In these scenarios, target tracking is one of the most important applications.

The simplest way for target surveillance is to turn on the sensor modules of all nodes in the network and have each node communicate directly with the base. This is the so-called Direct Communication (DC) method [1]. DC, although delivering the best accuracy, is unrealistic in real-world applications, mainly because the base has only a limited number of channels. In addition, the node communication distance is limited, so the DC method is not applicable to a large area. Another problem of efficient target tracking in WSNs is the redundancy issues. Because nodes are typically deployed at random locations (for example, thrown by an airplane), it leads to overlapping among sensing areas of the nodes.

Existing computer network protocols may not be applicable to sensor networks, because sensor nodes are constrained in energy supply, performance, and bandwidth. Existing methods attempting to alleviate these constraints, such as the LEACH-based algorithms [5], however, either suffer redundancy in data and sensor node deployment, or require complex computation in the sensor nodes. Those drawbacks result in energy use inefficiency and/or complex computation overhead. Therefore, there exists a demand for self-organizing and routing capabilities in the sensor network, in order to achieve optimized computation and energy dissipation, and to maximize the lifetime of the sensor network.

In this paper, we present the results of simulation-based evaluations of three algorithms for sensor networks. The first is the Direct Communication (DC) method [1]; the second is the well-known LEACH [5]; the third is a method we devise, OCO (Optimized Communication and Organization). OCO ensures not only maximum accuracy of target tracking, but also efficient energy dissipation and low computation overhead. In the rest of the paper, we first provide an overview of methods related to target tracking in the WSN, followed by a detailed discussion of the OCO method. We then discuss the simulation models and environment that we have built to evaluate the methods. The evaluation results of some of the scenarios are then presented. The paper concludes with a summary and anticipated future work.

2. Related work

According to a survey by Chuang [2], there exist three main approaches for target tracking in WSNs: tree-based, cluster-based, and prediction-based. Tree-based methods organize the network into a hierarchy tree. Alternatively, a sensor network may be represented as a graph, in which the sensor nodes are vertices and the edges are links among nodes that directly communicate with each other. Examples of tree-based methods include DCTC (Dynamic Convoy-Tree-based Collaboration) [4] and STUN (or Scalable Tracking Using Networked Sensors) [3].

In STUN each edge of the graph is assigned a cost, which is computed from the Euclidean distance between the two nodes. Construction of the tree is based on the costs. The leaf nodes are used for tracking and sending collected data to the base through intermediate nodes. The intermediate nodes store a detected object set, and send update information to the base when there is any change in the detected object set. STUN, however, has some limitations. First, the tree in STUN is a logical tree and does not reflect the physical structure of the sensor network; hence, an edge may consist of multiple communication hops and a high communication cost may be incurred.

Although being related to the tree-based methods, cluster-based methods use an algorithm called LEACH (Low-Energy Adaptive Clustering Hierarchy [5]), to build a hierarchy tree for the network. LEACH consists of 2 phases. In the *set-up* phase, sensors may elect randomly among themselves a local cluster head. By doing so, the network may balance energy dissipation across the whole network. The optimal number of cluster heads is 5% of the total number of nodes [5]. After the heads are selected, they advertise to all sensor nodes that they are the new cluster heads. Once the nodes receive the advertisements, each of them decides to which head it would belong.

In the *steady* phase, sensors sense and transmit data to the base through their cluster heads. After a certain period of time spent in the steady phase, the network restarts the set-up phase again.

LEACH is more realistic than DC because it uses multi-hops to communicate. However, LEACH assumes all nodes have enough power to communicate directly with the base. Such an assumption is not true when the sensors spread across a large area. The cluster heads communicate directly with the base, possibly causing channel overload at the base station. Additionally, the cluster heads are randomly elected, so in some areas within the network there may not exist any cluster head.

Prediction-based methods are built upon the tree-based and the cluster-based methods, with added prediction models. The models rely on heuristics based on some of following assumptions [2]: (a) The moving objects will stay at the current speed and direction for the next few seconds. (b) The object's speed and direction for the next few seconds can be deduced from the average of the object's movement history. (c) Different weights can be assigned to the different stages based on the history. PBS (Prediction Based Strategies) [6] and DPR (Dual Prediction-Based Reporting) [7] are examples of prediction-based methods.

3. The OCO method

OCO includes 4 phases. In the *position collection* phase, the base collects positions of all reachable nodes in the network. In the *processing* phase, it applies image processing techniques to clean up the redundant nodes, detect border nodes, and find the shortest path from each node to the base. In the *tracking* phase, the sensor nodes all work together to detect and track intrusion objects. The *maintenance* phase involves re-organizing the network when, for example, a change in the topology of the network occurs, or some of the sensor nodes die (i.e., running out of power).

Position collection phase

When the sensor nodes are first deployed randomly in an area, the base starts by sending a message to its neighbors to gather their IDs and positions, and at the same time advertising its own ID as the parent ID of the neighbor nodes. Each of the base's neighbor nodes, after sending its ID and position to its parent, marks itself as *recognized*, and then performs the same actions as the base by collecting IDs and positions from their neighbors, and advertising itself as the parent node, and so on. Note that, when a node gets the position and ID from a neighbor, it forwards the information to its parent. This way the message will eventually reach the base.

Processing phase

The processing phase consists of three steps: (a) Clean up redundant nodes; (b) Define the border nodes; (c) Find the shortest path from each node to the base.

A redundant node is a node whose sensing coverage zone is occupied by one or more other nodes. Table 1 is the algorithm that removes the redundant nodes.

Table 1. Algorithm for Removing Redundant Nodes

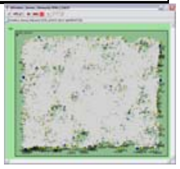
1. Build a geographic image of the network by assigning color value = 1 for all points that is covered by at least one sensor node. The rest of the points are assigned color value = 0.

(**Note**: The sensor network area is defined by a rectangle of (x_min, y_min, x_max, y_max), in which x_min and x_max are the min and max values of x, and y_min and y_max the min and max values of y in the collected positions.)

2. Initialize a list of nodes that are supposed to

cover the whole network area, called *Area_List*. Assign Area_List = null.
3. Add the base node to the *Area_List*.
4. For all the nodes in the area, if a node is not overlapping with any node in the *Area_List*, add it to the *Area_List*. The purpose of this step is to optimize node distribution.
5. For each point in the network area, if the point is not covered by any node in the *Area_List*, add the node that contains the point to the *Area_List*.
6. Nodes that are not in the *Area_list* after the "for" loops in steps 3, 4, and 5 are redundant nodes.

Figure 1 illustrates the network before and after the process of cleaning up redundant nodes. Image (a) is the initial sensor network, showing numerous redundant nodes. Image (b) is the sensor network after redundant nodes have been removed.


(a) before

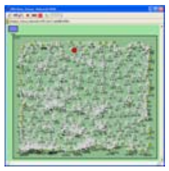

(b) after

Figure 1. Removing redundant nodes

Nodes that are positioned along the border of the network area are called *border nodes*. To identify these nodes, we first apply the border detection algorithm (Table 2) to identify a list of points that traverse the border of the geographic image, called *border points*. Finally, find a minimum set of nodes in the *Area_List* that contain all the border points, which are the *border nodes*.

Table 2. Algorithm for Finding the Border

1. For each pixel in the image, check if the color value =1.
2. If true (meaning this pixel belongs to an object), scan all its neighbors to see if any of them having the color value = 0. If true, this pixel belongs to the border.

(Note: To optimize the border nodes, we adopt a *border moving* algorithm, which is based on Euclidean Distance. The image border is moved toward inside of the network area by a half of the sensing radius. By doing so, the number of border nodes will decrease significantly without sacrificing any major characteristics of the network. This change may cause the accuracy of object detection to decrease a little bit, because the objects will be recognized a little bit later. The delay is acceptable though, in light of the gained benefit of reduced number of border nodes.)

The process of identifying the border nodes is illustrated in Figure 2. Diagram a represents the initial collection of sensor nodes in the network. Diagram b represents an intermediate step, where border points are identified. Diagram c represents the resulting border nodes, without displaying the rest of the nodes.

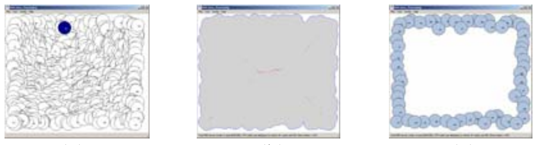

(a) (b) (c)

Figure 2. Detection of border nodes

The algorithm in Table 3 finds the shortest path (the least hops) to the base for each node in the *Area_List*.

Table 3. Algorithm for Finding the Shortest Path

1. Work only with nodes in the *Area_List* (resulted from the *Removing Redundant Nodes* step).
2. Assign parent_ID = 0 for all nodes.
3. Assign parent_ID = the base's ID for all neighbors of the base and add these nodes to a list, called *Processing list*.
4. For each node in the *Processing list*: Consider all its neighbors. If the neighbor having parent_ID = 0, assign the neighbor's parent_ID = the node's ID. Add the neighbor to the processing list.
5. Repeat step 4 until all nodes in the *Area_list* are assigned parent_ID.
6. After the loop, each node in the *Area_list* has a parent_ID. When a node wants to send a message to the base, it just delivers the message to its parent. The message is then continually forwarded until it reaches the base. The algorithm ensures that all the messages will reach the base through a minimum number of hops.

Figure 3 illustrates the forming of routing paths, which are constructed by following the parent ID of each of the nodes until reaching the base. The base is represented as the root of the tree in the diagrams.

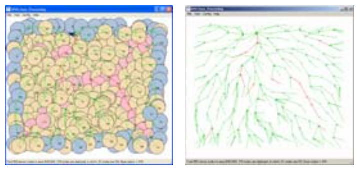


Figure 3. Shortest path from nodes to the base

After the processing phase, the base broadcasts messages to activate the set of *border nodes*. The border nodes have the sensor modules and the radio receiver modules ON (i.e., ACTIVE state).

The redundant nodes are initially OFF (i.e., SLEEP state). Periodically they wake up after a predefined long period to receive commands from the base. If there is no command or the commands are not related to them, they again switch to OFF. The rest of the nodes in the network are *forwarding nodes*, which have their sensor modules OFF but the radio receiver modules ON (i.e., FORWARD state).

Tracking phase

Objects are assumed to have come from the outside. Normally, only the border nodes are ACTIVE. When a border node detects an object, it periodically sends its position information to the base by first forwarding the information to its parent.

When it comes to detecting multiple objects, there exist two different types of sensor nodes. *Type A* can sense distinct multiple objects [8]; *Type B* does not have this capability. Type A of sensor nodes can accurately track each of the objects; thus it only needs to activate its neighbors when a particular object is leaving its coverage area. Type B, on the other hand, can only detect whether there is any object at all within its coverage area. Without the capability to identify each individual object, type B of sensors need to periodically activate its neighbors, assuming one or more of the objects may have left its coverage area.

Therefore, in a network equipped with type A sensors, an ACTIVE node that has lost an object will turn all its neighbors (forwarding nodes) to ACTIVE (assuming that the 'escaping' object will enter one of the neighbors' sensing areas). (**Note**: We assume that the delay time for a sensor node to activate its neighbors is smaller than the sensing radius divided by the object's speed.) If the neighbor detects any object, it will send its position to the base. And again right after it has lost an object, it turns all its neighbors to ACTIVE. The process will continue as long as an object is detected by the network.

In the second case where type B sensors are used, an ACTIVE node will periodically turn all its neighbors (forwarding nodes) to ACTIVE. If the neighbor detects an object, it will send its position to the base and periodically turn its own neighbors to ACTIVE.

In either case, if activated neighbors detect nothing, they automatically switch to the original state (FORWARD) after a predefined short interval.

Maintenance phase

The purpose of this phase is to reconfigure the network when the need for topology change arises. Example cases of such changes are discussed below.

Case 1. *Exhausted* Nodes: When the energy level of a node is below a threshold, it turns all its children to SLEEP and sends a report to the base. When the base gets the report, it enters the *processing* phase to reconfigure the whole network, with dead nodes being removed and the network restructured.

Case 2. *Damaged* Nodes: After a predefined interval of time, nodes require their child nodes to send their IDs to them. Child nodes that do not report to their parents are assumed to be damaged and will be reported to the base. Similarly, if a child node did not receive any asking from its parent after the predefined interval of time (meaning the parent may be damaged), it will turn to SLEEP mode and wait for further command from the base.

Case 3. *Re-positioned* Nodes: When a node's position changes (probably due to physical events, such as earthquakes, explosion, etc.), it will be considered as damaged by its parent (case 2.). After a node's position is changed, it will: (a) automatically turns to SLEEP mode; (b) Broadcast a message indicating that its position needs to be updated.

Any node that has received the broadcast will forward the information to the base, which then updates the given node's position.

4. Simulation and evaluations

In evaluating the performance of OCO, two other methods are selected as comparisons: the Naive method (DC, Direct Communication) and the cluster-based method (LEACH). The tool that is used for simulation is OMNET++ [9]. It is an open-source, component-based, modular and open-architecture simulation environment with strong GUI support and an embeddable simulation kernel. OMNET++ allows the builder of a simulation environment to place the simulated modules at any place. This capability enables us to simulate the random location feature of a sensor network, as well as build moving objects.

4.1 Models and Tools

The simulation models we have built to test the performance of the sensor network consist of three sub-models: *a sensor-node model*, *a sensor-network model*, and an *intruder-object model*.

The Sensor Network Research Group at Louisiana State University has defined a generic sensor node [11]. Based on this generic design, we have built a simulated sensor node. The *Physical Layer* module of the sensor node represents the physical layer of a sensor node. It is responsible for making connections between the node and its neighbors, and forwarding messages from a higher layer to its neighbors, and vice versa. The *MAC* module represents pre-processing packet layers. It consists of gates and queues. When the queue is full, it deletes some of the oldest messages in the queue to make room for new messages. The *Application* module represents the application layer. Note that, each time after sending a message, the module automatically sends a DECREASE_ENERGY message to the *energy* module (through the *coordinator*) to let the module decrease the energy by a number of energy units.

The *Coordinator* module is an interface to connect all modules together. It categorizes an incoming message in order to deliver it to the right module. The *Sensor* module represents the sensor board in a node. If the SENSOR_SWITCH parameter is ON (=1), the module consumes energy. It is automatically OFF after an interval of sensing nothing. The *Radio* module represents the radio board in a sensor node. If RADIO_SWITCH parameter is ON (=1), the module consumes energy. The *Energy* module represents battery in a sensor node. If the module receives a DECREASE_ENERGY message, it decreases the energy level by a number of energy units.

In a real sensor network, the sensors continuously try to detect the object. In the simulation, the sensing behavior is simulated by first creating connections between the object and the sensor nodes near it. ACTIVE sensor nodes whose sensing zone cover the object will periodically receive from the object a SENSOR_INFO message.

A simulated intruder object needs only two modules: the *ObjectApplication* module on top of the *Physical Layer* module [11]. The *Physical Layer* module is similar to the *Physical Layer* module of the sensor node. However, the connections in the intruder object are re-created after each movement. The *ObjectApplication* module helps the object to move, by reading position data from a text file.

A sensor network includes a set of sensor nodes. To simulate such a network, we need a module called *manager* [11] to help simulate tasks such as making connections among the nodes, making connections between the nodes and the object, and saving simulation results, etc. To construct the sensor network, the *manager* module starts by first reading data from a file, which stores network configuration information, including sensor node and object positions, tasks, and routing information, etc. It then makes connections among nodes, by checking the coverage zone of all nodes to see if any node is in the zone and, if yes, making connections between the node and the covered nodes.

Each time an object moves, the *manager* module will consider if any node is in the object's zone, within which a node can sense the object. If such a node exists, the *manager* module creates a connection between the object and the sensor node, so that the object can send the SENSOR_INFO message to the node. The *manager* module also handles the broadcast sent by the base, by creating connections between the base and all the nodes. Finally, the manager module controls the power switch (POWER_SWITCH parameter) of all nodes in the network.

4.2 The Metrics

There are overall four types of metrics that are considered when comparing the performance of the three selected methods:

- *Total energy consumption* is the total energy that the network spends in a given scenario.
- *Accuracy* is a percentage of the number of detected object positions of the given method over the number of detected positions of DC. The underlying assumption is that the DC method, due to its direct communication to the base, should exhibit the highest accuracy in detecting objects.
- *Cost per detected point* is the ratio between *energy consumption* and the number of detected positions. It represents the average number of energy units that are spent for a detected position.
- *Time before the first dead node* is the time when the first node of the network runs out of energy. This matrix is a significant indication of the sensor network's 'well-being' or longevity.

4.2.1 Energy consumption calculation

There are three modules contributing to the energy consumption in a node: the radio module, the sensor module, and the MCU (Micro Controller Unit).

The *radio module* is responsible for wireless communication among nodes. A typical radio module used in wireless devices is discussed in [5]. Our modeling of the radio module is based on the model in [5]. In a real device, the transmit module normally stays in sleep mode. It only wakes up when there is any bit that needs to be sent. The receiver module needs to be ON when waiting to receive messages.

The sensor board, the MCU (CPU board, Memory board), and the radio board may work in one of two modes: In the *sleep mode*, the energy dissipation is almost zero; in the *full action mode*, the energy consumed depends on the respective operations. For example, in a MICA2DOT sensor node, the current incurred for the *transmit* operation in the radio module is 12 mA (milli-ampere), while a *write* operation in the logger memory takes 15 mA. [12]

We use the assumptions in [5] as the basis when calculating the energy dissipation for our simulations, which are summarized below.

(Note: *J* means 'Joule'. A *Joule* is the unit for measuring quantity of energy. 1 Watt = 1 Joule/second.)

• Energy consumption for modulating or demodulating one bit: *Eelec = 50nJ/bit*
• Energy consumption for spreading one bit to an area

of radius r = 1 meter (i.e., πm^2):
$\epsilon amp = 100pJ/bit/m^2 = 0.1nJ/bit/m^2$
• *Data rate = 2,000 bits/s*
• *Data package size = 2,000 bits*
• *Signal package size = 64 bits* (Size of advertising, neighbor activation, or maintenance messages)

Due to limited space, detailed derivations of the remaining parameters for calculating energy consumption are omitted. Table 4 summarizes the operations and their respective consumed energy.

Table 4. Summary of WSN energy consumption

Create/Receive a data message	100 µJ
Create/Receive a signal message	3 µJ
Send a data message (d<= 60m)	820 µJ
Send a signal message (d<=60m)	26 µJ
Send a message (d > 60m)	100 µJ + 0.1* d^2
Sensor board (full operation)	66 µJ/s
Radio board (idle/receive mode)	100 µJ/s

Energy consumption calculation in DC

In DC, the sensor boards of all the nodes are in full operation. The states of the various boards are summarized in Table 5.

Table 5. Summary of states in DC

Sensor	Active
Radio	Sleep; wake up for transmitting only.
MCU	Sleep; wake up for creating messages only.

When a node senses an object, it transmits the sensing information to the base directly. Nodes do not need to communicate with each other, so the radio boards are in sleep mode.

Energy Consumption calculation in LEACH

In LEACH, the sensor boards of all the nodes are in full operation. When a node senses an object, it transmits the information to its cluster head, which then forwards the information directly to the base. A cluster head needs to receive messages from its clients, so the radio board of a cluster head is in the receiving mode. The radio boards of other nodes are in sleep mode.

Table 6. Summary of states in LEACH

	Head nodes	**Client nodes**	**Wild nodes**
Sensor	Active	Active	Sleep
Radio	Receive	Sleep; wake up to transmit only	Sleep
MCU	Sleep; wake up to create messages only.	Sleep; wake up to create messages only	Sleep

We know that one of the weaknesses of LEACH is that nodes do not always get invitation to join a cluster when, for example, there is no cluster head in their zone (called *wild nodes*). To simulate such a scenario, the sensor boards of all nodes that do not enroll with any cluster head are turned off. The states of various nodes in LEACH are summarized in Table 6.

Energy Consumption calculation in OCO

In OCO, there are three types of nodes: border nodes, forward nodes, and redundant nodes. The states of various nodes in OCO are summarized in Table 7.

Table 7. Summary of states in OCO

	Border nodes	**Forwarding nodes**	**Redundant nodes**
Sensor	Active	Sleep	Sleep
Radio	Receive	Receive	Sleep
MCU board	Sleep; wake up to create messages only	Sleep; wake up to create messages only	Sleep

The radio boards of all nodes are in receiving mode because nodes in OCO need to exchange data. The sensor boards of all border nodes are in full operation. Forwarding nodes, however, have the sensor boards off. A forwarding node is turned ON when receiving a message from one of its neighbors. The redundant nodes are used as backup nodes. They initially have all boards in sleep mode.

4.2.2 Calculation of object tracking accuracy

According to [10], a sensor network with all nodes in the tracking mode (i.e., the sensor board is in full operation mode) is a useful base for comparison, because it provides the best possible quality of tracking. An example is the DC method. So we consider the total number of detected points in this case as 100%, and call it the *standard number of detected points*. The *accuracy* of each method is a percent ratio between the *number of detected points* of the method and the *standard number of detected point*.

4.2.3 Cost per detected point

Cost per detected point is a ratio between the total *energy consumption* and the *total number of detected points* of the method.

4.2.4 Time before the first dead node calculation

The manager module periodically (every 0.1s) checks all nodes in the network to see if any node has run out energy. If it finds any, the simulation time at that moment will be recorded as the *time before the first dead node*.

4.3 The simulation environment

The simulation environment is built as an area of 640 X 540. The number of nodes in the network is 200, 250, 300, 350, 400, 450, 500,550, 600, 650,700, 750, 800, 850, 900, 950, and 1000, with 2J (Joule) of energy for each node. The sensing radius of each node is 30m and the communication radius is 60m.

Intruder objects are supposed to move by traversing specific paths and come from outside of the network area. Four different paths were used in the simulations. The moving paths of objects are created by drawing images. Figure 4 shows one of the paths. A MATLAB program reads the images and generates appropriate text files of positions of the path images.

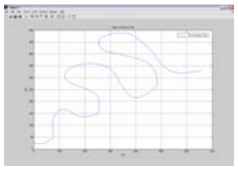


Figure 4. A sample path

5. RESULTS

The Base Cases: no intruder objects

In the base case, there is no intruder object. Two metrics, *energy consumption* and *time before the first dead node*, are measured. The results of running the methods in 9,000 seconds of simulations are shown in Figures 5 and 6. In Figure 5, when the number of nodes increases, the energy consumption of OCO goes to a constant, delivering much better result than the other methods. The reason is that, when the number of nodes increases, the size of the border in OCO decreases (less gaps between border nodes and straighter border line).

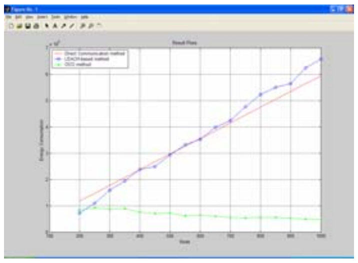


Figure 5. ***Energy Consumption*** **(no intruder)**

Figure 6 shows that OCO lasts longer than LEACH in all cases. (Note: Value of 1,000 means there is no dead node in the network during the 9000 seconds of simulation.)

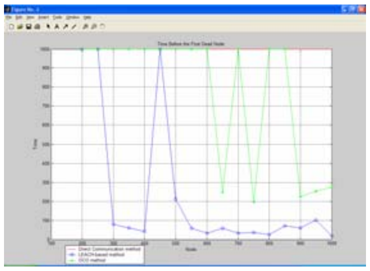


Figure 6. ***Time before the first dead node*** **(no intruder)**

Tracking Cases A: one intruder object

In this case, the simulation results are divided into four cases. Each of the cases represents the collected metrics given a particular path of the moving object, with the moving speed being 10 points/s. In these scenarios, nodes in OCO only need to activate their neighbors when the object is leaving its coverage area. Due to the limited space, evaluation results of this case are not included. The results, however, are compatible with the 'multiple objects' cases below.

Tracking Cases B: multiple intruder objects

Scenarios in this case are similar to the 'single object' case above, with the difference that multiple intruder objects are present in the environment. Each of the objects traverses a different path, with the same speed at 30 points/s. *Type B* sensors are used in the simulations. As shown in Figure 7, when the number of nodes is greater than 300, OCO consumes much less energy than the others, and it appears to reach a constant when the number of nodes is 500 or higher. At 1,000 nodes, the energy dissipation of OCO is about 1/4 of DC and 1/6 of LEACH. (Note: The reason that the energy consumptions of LEACH and OCO appear to fluctuate at 600 and 900 is that, in these cases, the base happens to be far from the moving paths. As shown, even in those cases, OCO is more stable than the others.)

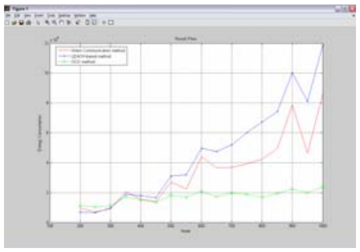


Figure 7. Energy consumption

Figure 8 shows that the *accuracy* of OCO is compatible to DC in most of the cases. LEACH, although showing very good results when the number of nodes is higher than 700, exhibits less stable results when the number of nodes is lower.

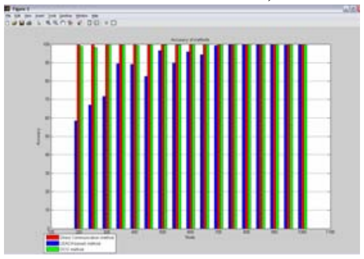


Figure 8. Accuracy comparison

Figure 9 shows the cost per detected object position. OCO's cost is the lowest in all the cases when the number of nodes is greater than 300, and the costs appear to remain constant. The cost of LEACH appears to increase relative to the number of nodes.

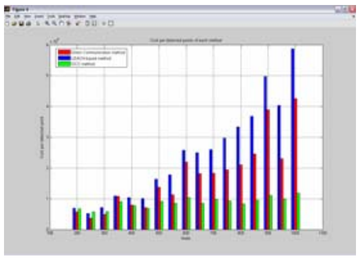


Figure 9. Cost per detected point

As shown in Figure 10, the *time before the first dead node* appears to be "fluctuating" across the cases, mainly because that, depending on the relative distance between the intruding object and the base, one of the nodes may run out of energy early. Still, OCO appears to last longer than LEACH in all cases.

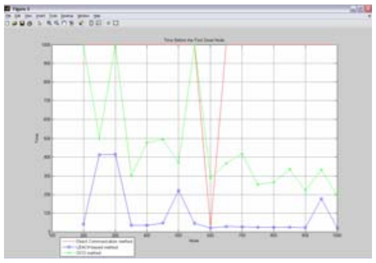


Figure 10. ***Time before the first dead node***

6. Summary and future work

We have devised a method, OCO, for efficient target tracking in wireless sensor networks, and have evaluated its performance in various simulation scenarios against two other methods (DC and LEACH). Based on the evaluations, OCO appears to consume less energy than the other methods while achieving superior accuracy. The main strengths of OCO include its efficiency and easy maintenance, meaning that, when too many nodes have exhausted their energy, new nodes can be refilled to the tracking area and the OCO method will be able to dynamically build up a new network.

The sensor network usually operates in hostile environments. Therefore it is critical to add security features to OCO. Part of our future work is to add authentications and other security features to OCO.

7. Acknowledgement

The authors are partially supported by the Institute for Space Systems Operations (ISSO), and the National Science Foundation (DUE 0311592).

8. References

[1] Guo, Weihua, Zhaoyu Liu, and Guangbin Wu (2003). "An Energy-Balanced Transmission Scheme for Sensor Networks". *Dept. of Software and Information Systems - Univ. of North Carolina at Charlotte*. Retrieved 9/8/2005 at http://www.cens.ucla.edu/sensys03/proceedings/p300-guo.pdf

[2] Chuang, S. C. (5/26/2005) . "Survey on Target Tracking in Wireless Sensor Networks ". *Dept. of Computer Science – National Tsing Hua University*. Retrieved 11/8/2005 at http://mnet.cs.nthu.edu.tw/paper/934355tbl/050526--Survey%20on%20Target%20Tracking%20in%20wireless%20sensor%20newworks.pdf.

[3] Kung, H. T., and D. Vlah. (2003) "Efficient Location Tracking Using Sensor Networks." *WCNC, March 2003*. Retrieved 11/7/2005 from http://www.eecs.harvard.edu/~htk/publication/2003-wcnc-kung-vlah.pdf

[4] Zhang, Wensheng, and Guohong Cao (9/2004), "DCTC: Dynamic Convoy Tree-Based Collaboration for Target Tracking in Sensor Networks". *IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, VOL. 3, NO. 5, SEPTEMBER 2004*. Retrieved 11/7/2005 from http://mcn.cse.psu.edu/paper/zhang/Twireless04.pdf

[5] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan (2000). "Energy-Efficient Communication Protocol for Wireless Microsensor Networks". THE HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, JANUARY 4-7, 2000, MAUI, HAWAII. Retrieved 6/20/05 from http://academic.csuohio.edu/yuc/mobile03/0403-heinzelman.pdf

[6] Yingqi Xu Winter, J. Wang-Chien Lee (2004). "Prediction-based strategies for energy saving in object tracking sensor networks" Mobile Data Management, 2004. Proceedings. 2004 IEEE International Conference. Retrieved 11/7/2005 from: doi.ieeecomputersociety.org/10.1109/MDM.2004.1263084

[7] Xu, Y., Winter, J., Lee, W.-C. "Dual prediction based reporting for object tracking sensor networks" MOBIQUITOUS 2004. Retrieved 11/7/2005 from: doi.ieeecomputersociety.org/10.1109/MOBIQ.2004.1331722

[8] Andreas Savvides Mani Srivastava. "A SELF-CONFIGURING LOCATION DISCOVERY SYSTEM FOR SMART ENVIRONMENTS". Retrieved 11/7/2005 from: http://www.eng.yale.edu/enalab/publications/cpcn_chapter.pdf

[9] OMNET Official Website. "OMNET++ User Manual Version 3.2". Retrieved from: http://www.omnetpp.org/doc/manual/usman.html

[10] Sundeep Pattem, Sameera Poduri, and Bhaskar Krishnamachari (2003). "Energy-Quality Tradeoffs for Target Tracking in Wireless Sensor Networks". DEPARTMENT OF ELECTRICAL ENGINEERING AND DEPARTMENT OF COMPUTER SCIENCE,UNIVERSITY OF SOUTHERN CALIFORNIA. Retrieved 7/10/05 from http://www-scf.usc.edu/~pattem/PattemKrishnamachari_Tracking.pdf

[11] Sensor Network Research Group at Louisiana State University (2/1/2005), "Simulating Wireless Sensor Networks with OMNeT++". Retrieved 11/7/2005 from http://bit.csc.lsu.edu/sensor_web/final_papers/SensorSimulator-IEEE-Computers.pdf

[12] Chih-Yu Lin, Wen-Chih Peng, and Yu-Chee Tseng (2004). "Efficient In-Network Moving Object Tracking in Wireless Sensor Networks". Department of Computer Science and Information Engineering - National Chiao Tung University. Retrieved 11/20/2005 from http://www.csie.nctu.edu.tw/~yctseng/papers.pub/sensor02-tracking-ieee-tmc.pdf