A Distributed Algorithm for Waking-up in Heterogeneous Sensor Networks[1]

Akis Spyropoulos, Cauligi S. Raghavendra and Viktor K. Prasanna

Electrical Engineering - Systems
University of Southern California
Los Angeles, California
{spyro, raghu, prasanna}@halcyon.usc.edu

Abstract. In this paper we present a distributed, application-morphable, algorithm for waking up appropriate sensor nodes in a heterogeneous sensor network. We assume a sensor field consisting of a large number of low power, limited functionality, tripwire nodes and a smaller number of powerful, energy-hungry, tracker nodes. Our problem is that when an event is detected by a set of tripwire nodes a specific number of appropriate tracker nodes needs to be woken up. These tracker nodes will subsequently collaborate to perform the sensing task required by the application. Waking up non-suitable tracker nodes or employing more trackers than necessary for a specific task, can lead to significant waste of network resources (e.g. energy). The application indicates the number of nodes that are needed for a sensing task, as well as an optimization function to be used by the algorithm. Therefore, our algorithm is isolated from most application details and is simple and general enough to accommodate a wide range of sensing applications. We prove that our algorithm converges to a uniform optimal global decision for specific classes of optimization functions. Furthermore, we show that it is fast enough (<100ms) to be practical for most sensing applications and exhibits good performance in terms of total messages exchanged. Finally, we demonstrate that our algorithm is very robust, managing to retain its correct and efficient behavior for a wide range of scenarios, even under hostile environmental conditions (e.g. link loss probabilities up to 35%).

1 Introduction

Continuous advances in low power electronics and processor technology have enabled the design of inexpensive sensor nodes capable of performing significant computation and wireless communication. A potentially very large number of such nodes can be deployed either manually or automatically [1] in a field, building or any other area of interest and self-organize into a network to perform a specific application. Various sensor applications have been proposed, ranging from real-time tracking of one or more targets [2] to environmental monitoring [3].

[1] This work is supported by the DARPA Power Aware Computing and Communication Program under contract no. F33615-C-00-1633.

A considerable amount of research effort has been devoted during the past couple of years in the area of sensor networks. However, most of the related work adopts a homogeneous sensor network model consisting of a single type of sensor nodes that collaborate to perform a specific sensing task [13] [14] [15]. It has recently been identified [4] [5] [16] that having different classes of sensor nodes may be beneficent to many sensor applications. These classes may be differentiated in terms of processing capabilities, sensing modalities, communication capabilities and size or cost. Specifically, it is envisioned that a large number of inexpensive, low energy, *tripwire* nodes are going to be deployed along with a smaller number of more powerful, specialized, nodes. These specialized nodes will be capable of performing computation-intensive functions (e.g. ranging, tracking, etc.) or communication-specific functions like routing or long-range communication to base station(s).

In this paper, we assume, without loss of generality, that only one such type of specialized nodes is deployed, which we're going to be referring to as *tracker* nodes, hereafter. It is anticipated that tracker nodes are going to be dissipating one to two orders of magnitude more power than tripwire nodes when all their subsystems (i.e. CPU, radio and sensor) are on [16]. Therefore, tracker nodes will have to be "off" most of the time, when no events occur[2], in order to maximize the operational lifetime of the network. On the other hand, tripwire nodes run with low power and may even be self-sufficient by scavenging energy from the environment [6]. However, they only have limited processing and communication capabilities. Their functionality is expected to be that of detecting events, using some simple threshold detection scheme, and waking up appropriate tracker nodes to perform the sensing task.

Our problem is that when an event is detected by a set of tripwire nodes, a specific number of appropriate tracker nodes need to be woken up, that will subsequently collaborate to perform the sensing task required by the application. Waking up more nodes than necessary could result in considerable waste of resources. For example, it has been shown that performing *line of bearing (LOB)* calculation can be achieved with only three tracking nodes with acceptable accuracy. The extra energy consumed by utilizing more tracker nodes for that purpose is not justified by the potential extra accuracy achieved. In addition to that, different sets of chosen tracker nodes may result in considerably different performance, in terms of accuracy, energy-efficiency, etc. Therefore, it is evident that some coordination scheme is needed among the tripwire nodes to wake up an appropriately sized set of tracker nodes that are optimal in respect to the sensor application's specific goal(s).

In this work, we propose a general, efficient and robust distributed algorithm for the tripwires to collaboratively choose a set of specialized nodes for a specific application. In our algorithm, each tripwire node re-evaluates its local decision on the optimal set of tracker nodes to be woken up, whenever it receives a *decision update packet* from a neighboring tripwire. If it learns of or can construct a superior set of tracker nodes, it informs its neighborhood about it. A tripwire terminates its local run of the algorithm if it does not receive any *decision update packet* within a specified time interval. It assumes that a global resolution has been achieved and proceeds with waking up the tracker nodes. The algorithm takes as input from the application the number of nodes that need to be woken up for a sensing task, as well as an

[2] Typical sensor applications duty cycles may be as low as 5%

optimization function to be used by the algorithm, in order to evaluate the "appropriateness" of a chosen set. This way, the algorithm is isolated from many application details. Its interface is kept simple and general enough to accommodate a wide range of sensing applications and adapt to each application's specific optimization goals. In this respect, our algorithm is *application-morphable*. Our scheme makes only a minimum number of "helpful" assumptions, in order to be in accordance with typical sensor network operational requirements, which may not hold in other wireless environments (e.g. cellular, ad hoc networks). Specifically, tripwire nodes are not required to have any ids and only single-hop broadcast messages are used for all tripwire communications. Only tracker nodes are assumed to have some kind of local or global id.

We use both analysis and simulations to evaluate the performance of our algorithm. We prove that our algorithm always converges to the optimal set of tracker nodes, for a specific class of optimization functions and show that it does so quickly enough (in less than 100ms) to be of practical value in sensor applications with real-time requirements. We further show that the total number of messages transmitted by all tripwires until the algorithm finished is close to an optimal (but not practical) approach. Finally, we demonstrate that our algorithm is very robust, managing to retain its correct and efficient behavior for a wide range of scenarios, even under hostile environmental conditions (e.g. link loss probabilities up to 35%).

In the next section, we identify the sensor network specific operational requirements and formulate the problem. In section 3, we provide a brief description of the range of approaches that can be taken to solve the basic problem posed in this paper and present our proposed solution. Performance results for our distributed algorithm are given in section 4. Finally, in section 5, we conclude the paper and discuss directions for future work.

2 Problem Formulation and considerations

We assume that a large number of tripwire nodes are deployed along with a smaller number of tracker nodes in a field of interest. When no event is detected, all tracker nodes are off, performing no sensing, computation or communication. We assume that the tracker node radio either wakes up periodically and listens for incoming messages or has a low-power paging channel that is constantly on, listening for incoming "wake-up" messages. Tripwires are sensing for events at all times, but may keep their processor and radio off, while no event is sensed. When an event occurs, a number of tripwire nodes, say N, detect this event (e.g. receiving signal power exceeded some threshold). Note that N is not fixed, but depends on the strength and type of signal emitted by the event as well as the tripwire node topology and surrounding environment. For example, an acoustic signal can propagate further than a seismic signal and would therefore result in more tripwire nodes (i.e. larger N) detecting the event. In this work, we assume that these N tripwires are close-by and form a connected network. The issues of how to design and deploy the sensor network, in order to guarantee connectivity or full coverage of the field, with a very high probability, have been dealt with elsewhere [17] [18].

Each tripwire is initially *associated* with zero or more tracker nodes, by means of existing paths/gradients [7], established during some initialization phase, or simply by direct communication. Let M be the total number of trackers that are associated with at least one of the N tripwires ($M < N$). Any tracker j is characterized by a set A_j of attributes, like location, energy level, etc. Tripwire nodes learn about those attributes initially during *association* and subsequently by handshake, when waking up a tracker. Tripwires are not required to have a consistent view of trackers attributes. This may result because of different tripwires inquiring about the same tracker's attributes at different times. However, it is required that at least one tripwire has the correct (most recent) value for the attributes of interest for each of the M trackers.

The problem is to choose a set of K tracker nodes among the M possible ones that is optimal in regards to some optimization function F. Both K and F are application-dependent and are regarded as input parameters to the algorithm. F is an appropriate function of some or all tracker attributes, for the trackers belonging in a specific set of size K. If define as *AR* the range of all allowable values for an attribute set A_j then: $F:AR^K \rightarrow R$ or $F = f(A_1, A_2, \ldots, A_K)$. It captures the application-specific optimization goal(s). For example, if the application's goal is to choose the K trackers that have the maximum amount of remaining battery level, the following function F could be used: $F = E_1 + E_2 + \ldots + E_K$, where E_j is the battery level of tracker j in the chosen set of trackers. As another example, the application may wish to wake-up the *tightest cluster* of K trackers. In this case the application's optimization goal could be captured by $F = \sum_{n=1}^{K-1} \sum_{m=n+1}^{K} \left\| \vec{X}_n - \vec{X}_m \right\|$, where $\vec{X}_n$ is the position of the n-th tracker. The optimization function is required to be simple enough to be calculated efficiently by the limited processing resources of the tripwire node.

Any algorithm devised for this problem should have the following goals:

- Correctness: Each tripwire should come up with the same "good" set of trackers.
- Convergence Speed: Sensor applications usually have hard real-time requirements (e.g. tracking of moving target). Therefore, all tripwire nodes should converge quickly enough to allow the tracking nodes time to meet those requirements.
- Message efficiency: The total number of messages exchanged throughout the decision process is a direct indicator of the total energy spent in the network for communication. Consequently, it should be kept low.
- Fault-tolerance and Robustness: Many sensor applications are mission critical. Consequently, it is necessary that the algorithm remains efficient even when a number of failures occur (e.g. node failures, link losses, collisions, etc.)

In addition to these goals, any solution to this problem as well as to many other sensor-related problems must take into account a number of salient features of sensor networks that are not common in other wireless contexts. Many proposed schemes for sensor network problems have been directly transported from the related discipline of wireless ad-hoc networks or other related areas and adapted for the case of sensor networks. However, specific assumptions have been carried along in some cases with these schemes that do not necessarily hold for wireless sensor networks. Failure to take into account the following idiosyncrasies of sensor networks can lead into impractical and inefficient approaches to solving sensor network specific problems.

Sensor applications are most often data-centric [8]. Additionally, sensor nodes are expected to be mass produced and deployed in a dense fashion to achieve high redundancy and fault-tolerance. Hence, it may be impractical or unnecessary to assign each sensor node a unique global id. Furthermore, the environment under which a sensor network is going to operate is expected to be very harsh. Shadowing and fading effects, inadvertent interference or jamming as well as adverse ground reflection effects, due to very low antenna heights, result in frequent and random link losses. Consequently, one should not assume "clean" models (e.g. cells, circles, etc.) for the effective communication range for a sensor node's radio and design protocols that are able to cope with the harsh propagation phenomena in a sensor network. Finally, despite the improvement in hardware and electronics technologies, computational power, memory, communication bandwidth and energy are still expensive and limited resources for wireless sensor nodes, even more than they are for wireless PDAs or laptops.

3 Distributed wake-up algorithm

In this section we will present our proposed solution to the problem, which is a fusion-based distributed algorithm. However, we first present the two different edges in the continuum of approaches that can be taken to tackle the basic problem and highlight their advantages and disadvantages, in respect to the 4 goals defined in Sec.2. We do so, in order to provide necessary intuition and justify the decisions we made in our design.

3.1 A baseline distributed approach

A lot of work has been done in the past on a number of different issues related to distributed algorithms and their performance and limitations (see Chandy and Misra [19]). Here we describe a common, simple, approach, which is to have each node acquire global information and individually make a local decision. This way, if all nodes perform the same decision algorithm on the same global set of data, they all will come up with the same set of optimal nodes. In our general application model, this would mean that all N tripwires will need to acquire the necessary data for each of the M trackers, in order to locally choose the optimal set of K trackers, based on F. A general way to reach this state of global knowledge is to have each node flood each local view throughout the network. This exchange of information can be performed either in proactive (e.g. periodical) or reactive (e.g. event-triggered) manner.

An important concern when running any flooding-based algorithm in a wireless, shared media, network is the potential collisions of different update packets transmitted by neighboring nodes at about the same time. One way to deal with this problem would be to use some collision avoidance protocol, like 802.11 [9], which reserves the media before the actual data transmission. The use of the 802.11 protocol has two important drawbacks. First, it is based on the assumption of unique node IDs and can only reserve the media for point-to-point communication. Hence, one should either emulate broadcast by point-to-point transmission to each neighbor or just

broadcast and let the back-off algorithm resolve the collisions that will occur. Either case would increase both delay and total number of messages for the algorithm to converge. Therefore, we choose to enforce some randomization to the basic flooding algorithm, in order to prevent synchronization of updates and avoid potential collisions (to a large extent). Specifically, a node periodically broadcasts update messages (if it has any), choosing in every step a different waiting period, uniformly distributed in [0, T], where T is the maximum amount of time that a tripwire node may wait before broadcasting its update.

Assuming that no link or node failures occur during the data exchange, the algorithm will result in each node calculating the same, optimal, set of trackers. This follows from the fact that each node makes its decision by applying the same decision function on the same set of information. Each tripwire node will broadcast an update it receives (or its own initial local info) in at most time T after its reception. Therefore, if D_N is the diameter of the connected network formed by the N tripwires then it will take at most T_{cv} for all local info messages to propagate throughout the whole network and the algorithm to converge, where T_{cv} is given by $T_{CV} = D_N * T$.

Each of the N tripwire nodes has to receive and forward information about all M trackers, at least once. Therefore, in the worst case (e.g. a different update message propagates throughout the network for each of the M trackers) the total number of messages M_{TOT} that needs to be transmitted will be $M_{TOT} = O(N * M)$.

Finally, flooding algorithms can exhibit very good fault-tolerance and robustness properties. This is because they can discover all alternative paths from each node to another in a straightforward manner. Therefore, even if link conditions or a collision results in loss of some update packet to a node from one path, this packet will probably reach the node from some alternative path if there is any

3.2 A centralized approach

The opposite end in the continuum of approaches is the centralized one. In the general centralized case, some leader for the whole network needs to be elected [10]. This leader is going to collect data from all other nodes, combine them or make a decision based on the global data, and send back to all the nodes (or some specified sink node, depending on the application) the final result or decision. A path from each node to the leader must be pre-established (during initialization or periodic re-configuration) and be known, before data can be gathered at the leader. Different approaches can be taken to establish those paths, based on the specific application requirements and/or optimization goals. One approach would be to enforce a spanning tree on the network rooted on the elected leader. Then, each node can send its local view (e.g. set of associated trackers along with values for their attributes) up the tree towards the leader. Parent nodes combine data from children nodes with their own and further forward it up the tree. When the root node (leader) has collected all necessary info, it can use this to make a decision and forward that back to all nodes through that tree. Such a tree could be constructed, for example, by implementing diffusion routing [7] to establish gradients towards the leader. Alternatively, one could establish a shortest path (i.e. greedy) chain that spans all nodes in the network. Each node will then send its info to the successor in the chain, which in its turn will

combine the received message with its own and forward it to its own successor. When the last node (leader) in the chain is reached, it will make a decision based on the global data and send back this decision through that chain to all nodes. This approach is slower than the spanning tree one, but can be more energy efficient when transmission distances are relatively long or environment is harsh [11].

We claim that adopting any centralized approach for our problem would be conflicting with the fault-tolerance and robustness requirements posed earlier for any algorithm used. Additionally, it is evident that both approaches presented require a number of "helpful" general assumptions to be made, in order to be feasible (e.g. require tripwire nodes to have unique ids, in order to construct the spanning tree or chain). This would ultimately lead in lack of correctness and/or convergence under a broader range of operating conditions. Furthermore, it is well known that centralized approaches do not scale well with large number of nodes, which may be the case in many sensor applications. However, a centralized approach for this problem, albeit impractical in many cases, could be designed to be optimal in theory in terms of specific goals (e.g. energy-efficiency, delay, etc.). Therefore, we will analyze and use the performance of the centralized algorithm as a means to compare our distributed algorithm's performance to the best achievable one.

Since the algorithm is centralized, the leader (i.e. tree root or last node in chain) will eventually receive info by all nodes, assuming no nodes fail and no collisions occur. Therefore, it will be able to make an optimal decision and send it back to all nodes. In the case of the spanning tree, sibling nodes will collide if the try to transmit to their parent at the same time. Therefore, a random waiting interval, uniformly distributed in [0, T'], must be enforced, before each sibling will send its message to the parent. Note that the maximum waiting interval T' that is necessary to avoid collisions between neighboring nodes, with a high probability, depends on the average number of children per node compared to the case of the baseline distributed algorithm, where the maximum waiting time interval T depends on the average node degree (i.e. number of neighbors) in the network. In any case T' is at most equal to T. Let us denote the average number of children per node as d. Then the convergence time for the algorithm will be at most $T_{CV} = 2*T'*\log_d N$. The total number of messages in each case will be equal to $M_{TOT} \approx 2*N$, since each node (excluding leader and leaf nodes) has to send one message "upwards" to the leader and forward back the decision message down the tree or chain.

3.3 A fusion-based distributed algorithm

One drawback of flooding is that it requires each node to maintain a considerable amount of state information, in order to recognize duplicate packets. Additionally, it may propagate non-useful info throughout the network, resulting in a large number of unnecessary transmissions. Specifically, a node will forward a packet containing new info, regardless of whether that info is going to be useful or not in making the final local decision. For example, assume that the tripwires are cooperating in order to choose the 2 trackers with the maximum amount of energy, among the M ones. Assume further that some tripwire j, which has already broadcasted a packet

containing two trackers T1 and T2 with their respective energy levels E1 and E2, receives a new packet containing the following info: {T3, E3}, {T4, E4}. If E1,E2>E3,E4, then T3 and T4 do not belong in the optimal solution constitute non-useful information that should not propagate any further. This last argument is the motivation for our distributed fusion-based algorithm.

Our algorithm consists of one basic fusion-and-update step and a back-off mechanism, designed to optimize performance. First, we're going to present the basic algorithm steps and prove the conditions under which the basic algorithm converges to the correct (i.e. optimal) decision. Then, we're going to present the complete algorithm, along with its termination condition, and discuss how the back-off mechanism affects its performance.

3.3.1 Basic algorithm and convergence requirements

We have already formulated the basic problem, but here we'll summarize the assumptions for the sake of clarity:

- There are N tripwires and M trackers. Choose K trackers among M that optimize some function F (K < M < N)
- The tripwires form a connected network
- Each tracker j is related with a set of attributes A_j (e.g. energy, location, etc.), that are known by each tripwires associated with tracker j.
- $F = f(A_1, A_2, \ldots, A_K)$ is a function of the trackers' attributes belonging in a chosen set of size K.
- Each tracker j is associated with at least one tripwire node.
- Function F unambiguously ranks all $\binom{M}{K}$ choices in some total order.

The basic algorithm is the following and is executed by all tripwires:

Initial Step) *Each tripwire is associated with an initial set of (0 to K) tracker nodes. If more than K tracker nodes are within reach of a tripwire, then the tripwire chooses the K "best" ones using function F. It assumes that this initial set is the optimal set of nodes and broadcasts it.*

Fusion-and-Update Step) *Upon receiving a set of nodes from some neighbor, combine(fuse) the received set with the current one using F, to produce a superior, higher-ranked, set if possible.*

- *If a superior set is produced, update the current set and broadcast it.*
- *If not, ignore the received set and do not broadcast an update*

Lemma 1: If any set of nodes $S1 = \{s_{11}, s_{12}, \ldots, s_{1K}\}$ that contains some number of nodes, say N1, that belong in the optimal set of tracker $OS = \{os_1, os_2, \ldots, os_K\}$, is ranked higher according to F than any other set containing less than N1 nodes in OS, then: The comparison of two sets $S1= \{s_{11}, s_{12}, \ldots, s_{1K}\}$ and $S2= \{s_{21}, s_{22}, \ldots, s_{2K}\}$ at some tripwire node can only yield a set S3 that is ranked equal or higher than both S1 and S2. Note: $os_i, s_{1i}, s_{2i} \in M$

Proof: There are only to possible cases that cover all scenarios:
Case 1) Assume there is some j for which $s_{1j} \in OS$ and there is no i for which $s_{2i} \in OS$, then S1 wins over S2 at this node. Consequently, S1 continues to be the current set, while S2 gets discarded at this node.
Case 2) If there are "optimal" nodes (i.e. nodes belonging in the optimal set OS) in both S1 and S2, then the combination from S1 and S2 that contains the largest number of "optimal" nodes is ranked the highest among any other combination of choices from S1 and S2. Since all combinations are tried, it is guaranteed that the best one among those will be discovered and get chosen (i.e. win) over both S1 and S2, which both get discarded at this node.

Lemma 2: Any non-optimal set of trackers S1 will propagate through the network for a finite number of steps, less than N, before it encounters a tripwire node with another set of trackers S2, the combination of which (S1 and S2) results in a set S3 that is higher-ranked than S1.

Proof: According to the basic algorithm, a set S1 that is received by some node L, will be further propagated (as is) by node L, only if it's superior than L's current set (say S2) and L's current set does not contain any optimal nodes (as shown before). Otherwise, S1 will either "lose" to S2, or produce a higher ranked set S3 and trigger a broadcast of the new set S3. If set S1 is non-optimal then it cannot visit all tripwires N and win over the local choices without producing a new better choice. Otherwise, S1 would be optimal.

Theorem 1: The basic distributed fusion-based algorithm converges after a finite number of steps to the optimal set of trackers OS, if and only if: any set of nodes S1 = $\{s_{11}, s_{12}, \ldots, s_{1K}\}$ that contains some number of nodes, say N1, that belong in OS, is ranked higher according to F than any other set containing less than N1 nodes in OS.

Proof:
Direct) Assume there are I initial sets in the network of tripwires, where $M \le I \le \min\left\{N, \binom{M}{K}\right\}$. There are only $\binom{M}{K}$ possible sets of trackers. Each set may be an initial set or get constructed at some step in the algorithm. According to Lemma 1, at any step of the algorithm the comparison of any two sets of trackers will never yield another set that is lower ranked than the initial two. Furthermore, according to Lemma 2, each set will take at most N steps to prevail (if optimal), produce a better one and trigger its broadcast, or lose. Therefore, it will take at most $\binom{M}{K} * N$ steps for all possible sets (including the optimal one) to propagate and the optimal one to prevail.

Converse) We're going to prove the converse, by the use of a counterexample. Assume the problem is to choose 2 among 4 total trackers. Let the 6 possible pairs be ranked by some F from better to worse as follows: (x1,x2), (x3,x4), (x1,x3), (x4,x2),

(x1,x4), (x2, x3). It is evident that Lemma 1 cannot be applied for this scenario. Assume there are 3 tripwires, A, B, and C, with the following initial sets: A -> (x3, x4), B -> (x1, x4), C -> (x2, x3). All 4 trackers are associated with some tripwire. However, if A broadcasts his set first, and reaches B and C, then (x3, x4) will win over (x1,x4) and (x2,x3) and prevail. After that, there is no way that the basic algorithm can find the optimal solution (x1,x2), since neither x1 nor x2 are included in any existing set any more.

3.3.2 Complete algorithm

The complete algorithm is created by introducing some amount of randomization on each step of the basic algorithm and adding a termination condition for it. Here are the modified steps of the distributed fusion-based algorithm:

Initial Step) *Each tripwire is associated with an initial set of (0 to K) tracker nodes. If more than K tracker nodes are within reach of a tripwire, then the tripwire chooses the K "best" ones using function F. It assumes that this initial set is the optimal set of nodes. The tripwire waits for an amount of time t1, uniformly distributed in [0, T] and then broadcasts the initial set. T is initially set to maxDelay, where maxDelay is a parameter of the algorithm.*

Fusion-and-Update Step) *Upon receiving a set of nodes from some neighbor, compare the received set with the current one using F, to produce a superior, higher-ranked, set if possible.*

- *If superior set produced, update the current set. Reduce the maximum backoff time T: $T = max \{T / d, T_{min}\}$, where d is a parameter of the algorithm. Wait for an amount of time t2, uniformly distributed in [0, T] and broadcast the updated set.*
- *If not, ignore the received set. Continue counting down to the next update broadcast (if one is pending)*

Termination Condition and Actions) *If no update message has been received for maxDelay time since the last update has been received, assume that the algorithm has reached a global decision and should terminate. Compare the initial associated set of trackers with the final chosen set. If there are any common trackers in both sets, send a wakeup message to those nodes. A tracker that successfully receives a wakeup message will broadcast an acknowledgement that it has woken up. This acknowledgement prevents other tripwire nodes, from sending redundant wake-up messages to the same trackers. The tripwire terminates its run of the algorithm when acknowledgements by all associated trackers belonging to the final set have been received.*

The reason for introducing the random waiting is twofold. First, it helps avoid synchronization effects between nodes' transmissions. If a fixed period was to be used for the updates, then concurrent transmissions by nearby nodes would result in a collision and either retransmission of the packets, if the collision gets detected, or else

permanent loss of the transmitted packet. The former would imply an increased number of transmitted messages as well as increased delay. The latter could compromise the correct operation of the algorithm, if the algorithm is not robust enough to overcome a few packet losses of this kind. Second, it tries to dynamically impose some hierarchical structure on the flow of information between tripwires and hence approach the centralized algorithm's performance. However, it does so without requiring any initialization or re-configuration protocol to explicitly enforce this structure beforehand. The price paid for this lack of pre-existing structure is that each link of this structure may potentially be "traversed" more than twice, which is the case for the (optimal) centralized algorithm. Finally, note that only a reasonable amount of computation needs to be performed by each tripwire node during the *fusion-and-update* step. In the general case, all possible combinations resulting from the merging of two sets of tracker would need to be examined, to make sure that the higher-ranked set will be discovered. This would imply an exponential (on K) amount of work. However, for the classes of converging functions that we're interested in, the higher-ranked set can be produced through only a linear (on K) number of combinations checked. We defer addressing the issue of computational complexity of different fusion algorithms for general optimization functions for future work.

4 Performance

We have simulated our algorithm using a high-level, event-driven, sensor network simulator we're building at USC. The tripwire topology assumed in all scenarios is a NxN *random grid*. A random grid has a grid-like structure, but with the following differences: Each vertex of the grid has a random skew from its position in a perfect grid, in order to model typical deployment processes that provide adequate coverage of the sensor field. Additionally, some links in the grid may not exist. Finally, extra bypass links may exist in the grid. The latter two model random propagation effects (e.g. shadowing, fading, ground reflection, etc.) that result in non-predictable connectivity effects in real sensor network [12]. The tracker nodes are uniformly distributed in the field. In the following paragraphs, we present the performance of our algorithm in terms of message complexity, convergence delay and fault-tolerance/robustness. All results are averaged over a large number of runs for various scenarios.

4.1 Message complexity

In Fig.1, we depict the number of total messages transmitted by all tripwire nodes involved, until the algorithm converges, as a function of the number of tripwire nodes (N) and the number of trackers (M). It is evident that the number of total messages grows linearly with N, but is not sensitive to M. Specifically, we can see that our algorithm is more efficient and scalable than the distributed baseline (flooding) algorithm, whose message complexity grows linearly with M as well. Furthermore, the total number of messages is in the order of 3N for all M and N, which is quite close to the optimal 2N of the centralized approach.

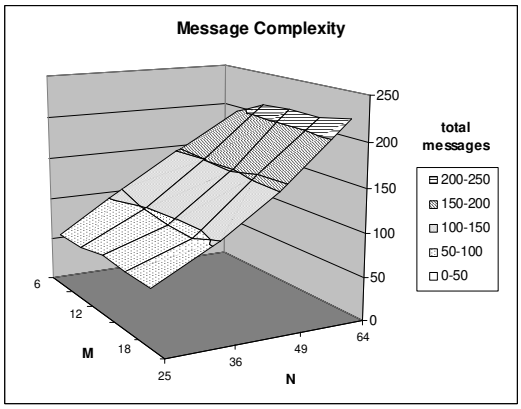

Fig.1. Number of total messages needed for the algorithm to converge as a function of the number of tripwires (N) and trackers (M). An NxN random grid topology is assumed, with N from 5 to 8. The number of trackers K to be chosen is fixed to 3.

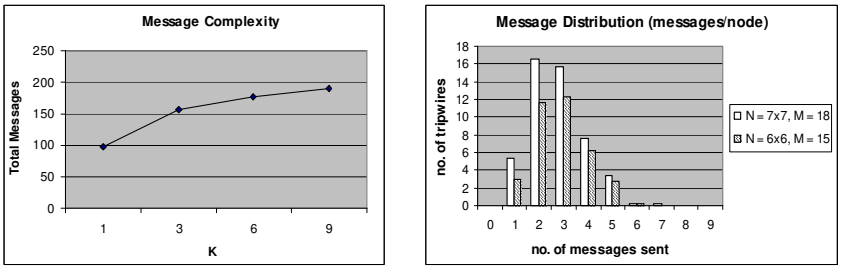

Fig.2. The graph on the left shows the number of total messages as a function of K. N is fixed to 49 (7x7 random grid) and M is fixed to 18. The graph on the right depicts the distribution of number of messages per tripwire node for two different scenarios.

In Fig.2 we examine another two aspects of the algorithm's behavior, namely the algorithm's dependence on the number K of trackers to be chosen and its load-balancing behavior. We can see that the number of total messages increases with increasing K. However, this dependency is not linear, as in the case of N. This behavior is ratified as follows: While K approaches M the number of extra messages per increase of K decreases, since info about a larger percentage of the tracker nodes is already included in the message. Finally, it is evident that our algorithm is able to balance the load of total messages among all participating tripwire nodes. This is very important in order to avoid depleting any specific tripwire's energy and maximize the network's operational lifetime.

4.2 Convergence Delay

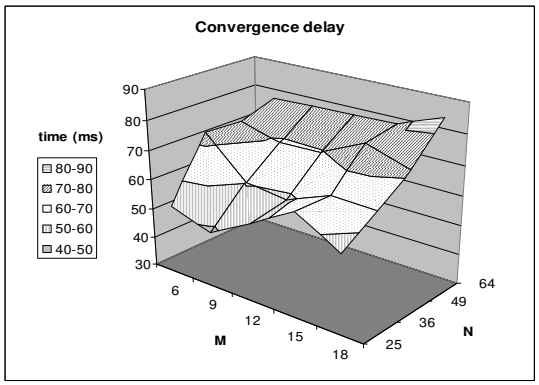

Fig.3. Convergence delay of distributed wake-up algorithm as a function of the number of tripwire nodes (N) and tracker nodes (M). K is fixed to 3.

The basic time unit in our algorithm and implementation is assumed to be the time it takes to transmit a single message. All other waiting interval parameters of our algorithm are calculated as multiples of that time. Therefore, the convergence time of the algorithm is not absolute, but is measured in terms of the basic time unit. However, in order to demonstrate the feasibility and practical value of our algorithm, we present results for its convergence delay, assuming a reasonable message transmission time, derived from current applicable technology specifications. Specifically, we have assumed that the message size is equal to 40 Bytes and transmission rate is 1Mbps for our simulations. The exact message size will depend on K, as well as the attributes that needed by function F. Nonetheless, 40-50 Bytes is a representative value, considering that K will not be too large and F is required to be relatively simple. Faster radios would reduce this time, while large Ks or a function F that depends on a large number of tracker attributes would increase it.

In Fig.3 we show the time it takes for the algorithm to converge (i.e. until all tripwires have decided on the final/optimal set of trackers to be woken up) as a function of N (no. tripwires) and M (no. of trackers). It is evident, as in the case of message complexity, that convergence delay is mainly affected by the total number of tripwires N and not as much by the total number of trackers. In all scenarios presented, the convergence time is less than 100ms. Therefore, we conclude that the delay overhead of our waking up algorithm is not very significant and the algorithm is fast enough to allow time to the trackers to do the "useful" work.

4.3 Fault-tolerance & robustness

In Fig.4 we examine the behavior of our algorithm under increasing link loss probability (i.e. probability that a message, that should otherwise be received correctly, is lost). It is evident that the algorithm is able to perform very well even in

very harsh environments. It manages to overcome almost all losses and perform correctly up to a "threshold point" (~35%), where significantly sized patches of non-optimal decisions start to appear in the field. Such high link loss probabilities may seem unrealistic for the typical sensor network. However, it is interesting to study the behavior of the algorithm under such high losses, since they could realistically model more hostile environments, where adverse ground-propagation effects or jamming may be present.

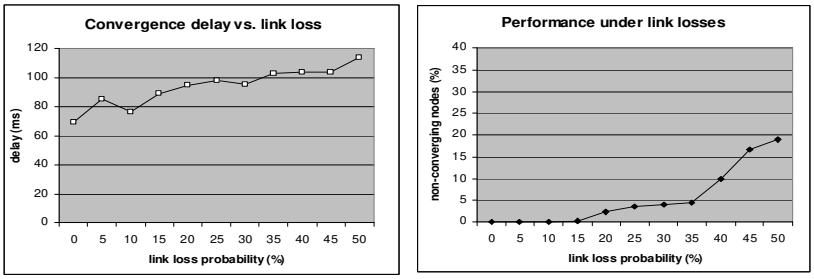

Fig.4. the left graph depicts the percentage of tripwire nodes that fail to converge to the optimal set of trackers, as a function of the link loss probability, while the graph of the right shows how link loss probability affects the convergence delay of the algorithm.

Intuition for why this threshold point exists is provided in Fig.5. Some link losses result in avoiding some collisions that would otherwise have occurred. Consequently, initially the number of total losses (collisions + link losses) does not grow rapidly and our algorithm can overcome them, by discovering (or re-discovering) alternate paths. However, after a specific point/threshold the number of alternate paths in the topology as well as well as any inherent redundancy in the algorithm is not enough to probabilistically overcome the majority of message losses. At this point, the algorithm starts to allow considerable decision inconsistencies among tripwire nodes.

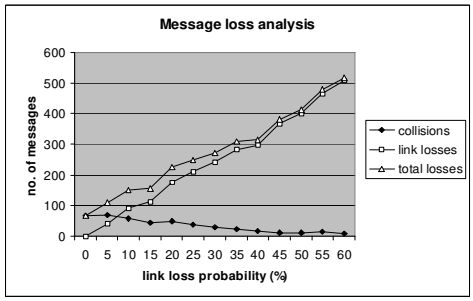

Fig.5. Message loss analysis. All lost messages (colliding messages or messages lost due to bad link conditions) are counted in terms of messages that should have been received

A final observation that needs to be made is the following. A few (<10%) incorrect tripwire decisions do not necessarily result in incorrect overall behavior. According to the termination step, each tripwire will attempt to wake up a tracker only if that tracker belongs in both its initial and its final set of trackers. In most of the scenarios

observed, only a few non-optimal (but high-ranked) tracker nodes, if any, will manage to not lose to the optimal ones after the end of the algorithm. Therefore, the probability of some tripwire having a non-optimal tracker in its final set and having that same tracker in its initial set is low, if the total number of misbehaving tripwires is also low. Consequently, in most cases where more than 90% of the tripwires converge, zero or one extra (non-optimal) trackers are expected be woken up.

4.4 Performance of fusion-based algorithm for general optimization functions

We have used simulations to evaluate our algorithm's behavior for general optimization functions, that is, functions that do not provide a partial or total ordering of the different sets. Such functions appear in scenarios like, for example, in the search for the tightest cluster of trackers or functions that are non-trivial combinations of distances between trackers, battery level of trackers and other tracker attributes. Those functions are not covered by Theorem 1, and therefore our algorithm does not provably converge to the optimal set. However, our simulations indicate that our algorithm manages to produce the correct, optimal, set of trackers for the vast majority of trial scenarios and a range of non-conforming optimization functions. Furthermore, it does so with no noticeable extra overhead. We plan to look further into such optimizations functions from a theoretical point of view, in future work.

5 Conclusions and future work

In this paper, we have presented a general fusion-based distributed algorithm for waking up in a heterogeneous sensor network, consisting of two types of sensor nodes, namely tripwire and tracker nodes. In our algorithm, tripwire nodes coordinate among each other to decide on an optimal set of a specific number of trackers to perform some sensing task. The number of trackers to be woken up, and an optimization function to be used to evaluate the appropriateness of a set, are provided by the application. We have proven the convergence of our algorithm for specific classes of optimizations functions. Furthermore, we have demonstrated that our algorithm is efficient, in terms of message complexity and convergence delay, for the majority of realistic scenarios. Finally, we have shown that our algorithm exhibits a very robust and fault-tolerant behavior, making it applicable for sensor applications operating in hostile environments, where adverse propagation effects like shadowing, fading and jamming are the rule, rather than the exception.

In the future, we're planning to further explore the behavior of the algorithm for different classes of optimization functions. Specifically, we would like to identify ways to approximate *non-converging* optimization functions with *converging* ones, and devise appropriate variations of the basic algorithm, in order to achieve near-optimal results for a wider range of function classes. Finally, we plan to extend our algorithm and provide a complete framework for event-driven sensor applications where, waking up appropriate tracker nodes when an event is detected, routing the result(s) back to the sink(s) and potentially cueing of a moving event/target, are all handled uniformly and efficiently.

6 References

1. J. Kahn, R.H. Katz, and K. Pister, "Emerging Challenges: Mobile Networking for Smart Dust," *in Journal of Communication and Networks*, Sept. 2000, pp. 188-196.
2. F. Zhao, J. Shin, and J. Reich, "Information-Driven Dynamic Sensor Collaboration for Tracking Applications," *in IEEE Signal Processing Magazine*, March 2002.
3. A. Mainwaring, J. Polastre, R. Szewczyk, and D. Culler, "Wireless Sensor Networks for Habitat Monitoring," *in Proceedings of ACM 1st International Workshop on Wireless Sensor Networks and Applications (WSNA '02)*, Atlanta, September 2002.
4. J. Nemeroff, L. Garcia, D. Hampel, and S. DiPierro, "Application of Sensor Network Communication," *in Proceedings of IEEE Milcom 2001, Communication for network-centric operation: Creating the information force,* Vol. 1, 2001, pp. 336-341.
5. M. Singh, and V. Prasanna, "A Hierarchical Model for Distributed, Collaborative Computation in Wireless Sensor Networks," *submitted to 5th Workshop on Advances in Parallel and Distributed Computation Models 2003,* Nice, France, April 2003.
6. J. Rabaey, J. Ammer, T. Karalar, S. Li, B. Otis, M. Sheets, and T. Tuan, "PicoRadios for Wireless Sensor Networks: The Next Challenge in Ultra-Low-Power Design", *Proceedings of the International Solid-State Circuits Conference*, San Francisco, CA, February 2002.
7. C. Intanagonwiway, R. Govindam, and D. Estrin, "Directed Diffusion: a Scalable and Robust Communication Paradigm for Sensor Networks," *in Proceedings of 5th ACM/IEEE Conference on Mobile Computing and Networking (MOBICOM 2000),* August 2000.
8. S. Ratnasamy, D. Estrin, R. Govindan, B. Karp, S. Shenker, L. Yin, and F. Yu, "Data-centric Storage in Sensornets," *in Proceedings of First Workshop on Sensor Networks and Applications (WSNA),* Atlanta, September 2002.
9. IEEE Local and Metropolitan Area Network Standards Committee, Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, IEEE standard 802.11-1999.
10. K. Nakano, and S. Olariu, "A survey on leader election protocols for radio networks," *in Proceedings of the 6th Internation Symposium on* Parallel Architectures, Algorithms and Networks, 2002 (I-SPAN '02). May 2002.
11. S. Lindsey, C. S. Raghavendra, and K. Sivalingam, "Data Gathering Algorithms in Sensor Networks Using Energy Metrics," *IEEE Transactions on Parallel and Distributed Systems*, September 2002.
12. D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker, "Complex behavior at scale: An experimental study of low-power wireless sensor networks," Technical Report CSD-TR 02-0013, UCLA, 2002.
13. M. Singh, and V. Prasanna, "Optimal Energy-Balanced Algorithm for Selection in a Single Hop Sensor Network," *submitted to 1st IEEE International Workshop on Sensor Network Protocols and Application (SNPA '03)*, May 2003.
14. R. S. Bhuvaneswaran, Jacir L. Bordim, J. Cui, Bordim, and K. Nakano,"Fundamental Protocols for Wireless Sensor Networks," *in International Parallel and Distributed Processing Symposium - IPDPS 2001*, San Francisco, April 2001
15. S.Iyengar, and Q. Wu, "Computational Aspects of Distributed Sensor Networks," *in Proceedings of the Fifth International Conference on Information Fusion*, July 2002.
16. PASTA project. http://neptune.east.isi.edu/pipermail/pasta/
17. Feng Xue and P. R. Kumar, "The number of neighbors needed for connectivity of wireless networks," *Submitted to Wireless Networks*, April 2002.
18. S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. Srivastava, "Coverage Problems in Wireless Ad-hoc Sensor Networks," *in Proc. of IEEE INFOCOM*, 2001.
19. M. Chandy, and J. Misra, "Parallel Program Design: A Foundation," Addison-Wesley 1988.