Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Sensing clouds: A distributed cooperative target tracking with tiny binary noisy sensors [☆]

Tal Marian [a], Osnat (Ossi) Mokryn [b,*], Yuval Shavitt [a]

[a] School of Electrical Engineering, Tel-Aviv University, Israel
[b] School of Computer Science, Tel-Aviv-Yaffo College, Israel

ARTICLE INFO

Article history:
Received 8 September 2011
Received in revised form 18 October 2012
Accepted 6 May 2013
Available online 5 July 2013

Keywords:
Sensor networks
Distributed algorithms
Target tracking
Tiny binary sensors

ABSTRACT

This paper suggests a novel algorithm for mobile object tracking using wireless sensor networks (WSNs). The paper assumes a future model of WSNs, where a large number of low to medium range inexpensive and noisy sensors are distributed randomly over an area. The distributed algorithm is based on short range communication between neighboring sensors, and is designed to work with very basic low cost binary sensors, that can report only a *sensing, not sensing* value.

Neighboring sensors that sense the object form a *cloud* around the object which is dynamically updated as the object moves. To save energy on reporting a subset of the cloud, the *cloud core*, is elected. A trade-off between the accuracy and the *core* size (namely transmission power) is presented, as well as an extensive simulation study. Our algorithm works well with false negative sensing and up to 10% false positive sensing.

1. Introduction

Sensor networks are formed by the distribution of small sensing devices over an area. These sensing devices have transmission capabilities, and are distributed randomly to create a multi-hop wireless communication system, referred to as Wireless Sensor Network (WSN). WSN-based object locating and tracking is an interesting and live research area, and the tracked objects vary from animals to vehicles. In this paper, we target the tracking of people over a large area for rescue purposes, for example in risky hiking areas, as well as for home security and law enforcement purposes (e.g., tracking suspects infiltrating an area). We assume that the target carries elements that can be sensed by magnetic sensors (like tent poles, hiking equipment, weapons, etc.). Alternatively, a target can be sensed using acoustic, chemical or other types of sensors.

For these tracking scenarios, a new WSN setting is considered lately [1]. According to the new model, a very large number of tiny noisy sensors is distributed over an area, probably through aviation distribution. The large deployment requires that the sensors are of very low cost and limited battery resources and thus low transmission range is used to maximize the sensors lifetime. Popular types of sensors [2] assume equal sensing and transmitting range, gaining up to a few years lifetime. However, the Pearl of Wisdom tiny sensors [1] were designed for a very long life span. In addition, due to the operational requirements of the tracking scenarios, for both homeland security and hiking areas, a large sensitive range is important. Thus, in their model, the main power savings were gained by lowering the sensor transmission range. Hence, these tiny sensors are characterized by a regular sensing range and a very small transmission range, and thus call for a new set of algorithms.

Cost considerations require that the sensors have very basic capabilities, and we assume here that this type of sensors does not measure the strength of the sensed signal, but that its sensing capability simply yield a *sense, not*

[☆] A short version of this paper appeared in IEEEI'10.
[*] Corresponding author. Tel.: +972 544270020.
E-mail addresses: tal.marian@gmail.com (T. Marian), ossi@mta.ac.il (O. (O) Mokryn), shavitt@eng.tau.ac.il (Y. Shavitt).

1570-8705/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.adhoc.2013.05.014

sense reading. We refer to this type of sensors, with a communication range that is several fold lower than the sensing range, and a binary type of report as *simple binary sensors*. Having low cost sensors may also result in false readings, either negative or positive. Thus the challenge to track an object with such sensors is substantial.

We suggest here a novel algorithm for the continuous tracking of a moving object over large areas using a WSN that consists of such inexpensive noisy sensors. To minimize system requirements and cost we develop the algorithm to work with basic sensors that report either *sense, not sense* readings. Our novel distributive algorithmic approach, named *sensing clouds*, is based on a simple local interaction between sensors. The sensors form a self-constructed sensing cloud, that consists of (at least) all sensing sensors around the object. The cloud is updated continuously, to enable accurate and fast tracking of the object. As the object moves, new sensors start sensing it, while other sensors are no longer within a sensing range. As a result, the group of *sensors* forming the *cloud* dynamically changes and the *sensing cloud* moves with the target, keeping the target at the *cloud's core*. We term this algorithm the *tracking algorithm*.

The algorithm is comprised of two sub algorithms working simultaneously, the first, *Cloud Formation Algorithm*, described in Section 4, builds and maintains the sensing cloud. The second, *Core Election Algorithm*, described in Section 5, deals with the core election process. Given that the sensors do not give a signal strength reading, there is no direct way to identify the set of sensors that form the cloud core and therefore are closest to the target. This challenging task is accomplished by the core election algorithm, and is key for a fast and accurate tracking.

The elected core is then responsible to inform the remote home station outside of the WSN what is the current target location. Many works have been done on the subject of relaying the information to the home station (HS), either by routing it back along the WSN towards the HS or to other sinks [3,4], by routing to a few well powered communication devises that transmit to the HS, or by using low cost optical communication [5]. This paper does not discuss the HS informing method, but rather relies on existing solutions. We do, however, take this extra cost into account, and evaluate the resulted core size. Thus, the core election algorithms, described in Section 5, trade off between the inter-sensor short-range communication and the core size (The sensors who need to inform the HS).

Our simulations of the cloud formation and core election algorithms, under a variety of changing parameters, show an accurate continuous tracking of a moving target. We further introduce false readings, e.g., due to noise and environmental blocking, into the system:

- False positive reading: Sensors may falsely report the sensing of an object, even when not within its influence field. This false sensing is the result of noise and may occur, e.g., due to difficult weather conditions.
- False negative reading: A sensor within the target's influence field reports a non-sensing reading. False negatives are more probable as one moves away from the target, where the signal to noise ratio is lower.

Our simulation results show that the algorithm is robust to false readings of up to 10% of the sensors in the field.

2. Related work

Tracking objects is an important application of WSNs [6–13].

There is an interesting line of research [14,15] that investigates what is termed binary sensors. The sensors in question have a ternary feedback: they can report if a target is moving towards or away from the sensor, or that the sensor does not sense the target. Aslam et al. [14] also use simple binary sensors, which they term proximity sensors, *in addition* to the ternary sensors. They suggest a centralized tracking algorithm using these sensors, where upon sensing an object the sensor reports to a base station whether it is approaching or moving away from it. The base station summarizes the information and tracks the object. In their work, they found that additional information is needed to disambiguate between different trajectories and to identify the exact location of the object. For this purpose, they suggest to attach to each ternary sensor a lower range proximity sensing binary sensor. Thus, when the two sensors report back to the base station, the information obtained is more accurate. Mechitov et al. [16] use location aware binary sensors to locally track the target; their algorithm has a large message complexity due to the need of many sensors to exchange location information to locally track the target using a distributed algorithm.

A distributed ternary sensing approaches were later developed by Wang et al. [17,18], where sensors communicate with close-by sensors, reporting whether they sense an approaching or moving away objects. Each node then calculates the target path over time according to locally gathered information, and places the target on an arc, which is created from its own sensing radius and the radii of two sensing neighboring sensors. The algorithm calculation becomes bothersome when the number of sensing sensors increases.

Liu et al. [19] also suggest a distributed group target tracking. The sensing sensors in their approach flood their SNR value to a radius which is twice the sensing distance, and the sensor with the highest SNR is elected as the group leader.

Lee et al. [20] propose a distributed algorithm for target tracking which aims to reduce the number of messages and the number of message collisions. The key element of their solution is a localization algorithm which is based on estimates of the ratio between the sensing sensors' distance to the target, rather than absolute distance to target estimates. By iteratively updating the estimated location using the distance to target ratio, the algorithm localizes the target with only a small number of sensors participation, thus reducing message collision and minimizing communication overhead. After localization, Lee et al. propose to dynamically adjust the reporting frequency according to the target's movement to reduce the number of report messages. Xiangqian et al. [21] analyze the localization error probability and tracking miss probability in the presence of prediction errors in this model.

Zhang and Cao [7] suggest a different approach with a similar model, namely they also assume that the sensors have a distance measure to the target. Like in our solution they build a tree rooted at the node closest to the target and dynamically move the tree as a new node becomes the closest to the target and hence take the role of the tree root. Our approach is similar to the latter, as we also build trees to efficiently track the target. However, since we use binary sensors, which are simpler and less expensive but cannot provide us distance estimation, our algorithms require more sophisticated mechanisms to identify tree roots. Nevertheless our algorithm is using less messages than the one suggested by Liu et al.

Arora et al. [22] present a system approach for the design of sensor networks for tracking objects. The work examines three possible object types and compares multiple sensor types. Sensors send binary reports to a centralized classifier, which uses time slots and known influence fields to distinguish true readings from false detections and to track the objects.

3. Model

For the purpose of detecting the intrusion of a target into an area and track its movement, we explore here a simple and practical sensing model, where sensors are truly binary, namely they can report either sensing or no sensing of the object. Thus, these sensors are low cost devices, with very-low transmission capabilities and rather limited battery lifetime. For these reasons, we require that inter-sensor communication is distributed and short-distanced, and require only sensing sensors and their close vicinity to participate in the algorithm. A subset of these sensors then decide on a result and send it back to the HS.

Formally, we model a rectangular *Field* in the *xy*-plane, the field area (A_F) can be calculated using the *x*,*y* dimensions. *Sensors* are scattered uniformly at random over the field. Each sensor is capable of communicating with other sensors within its local or inter-sensor communication range, d_c. We refer to two sensors whose distance from one another is less than d_c as *neighbors*.

3.1. Modeling sensing and sensing errors

Since sensors detect targets by measuring emitted energy signals, we modeled the sensing of targets according to a widely adopted signal model [23,19]. Suppose the distance between a sensor and the target is d and the maximal target's influence field is d_s, the sensor is said to sense the target with probability:

$$P_{sense} = \begin{cases} 0 & \text{if } d > d_s \\ 1 - (d/d_s)^2 & \text{otherwise} \end{cases} \tag{1}$$

$1 - P_{sense}$ is the false negative sensing error probability, which decreases as d^2 away from the target, namely quadratically.

Sensor measurements may also be corrupted by noise which causes sensors to enter a FP state. In order to account for FP sensing, we initially set each sensor outside of the target influence field to falsely sense the target with a desired probability. Next, we let each sensor outside the target influence field to change periodically its sensing mode according to a two state Markov chain that favors keeping the sensor's previous sensing state.[1] This way, FP readings are both random and continuous in time.

In our model all sensors have the same sensing and communication capabilities. Clearly, from our initial requirements, a target's influence field, d_s, should be larger than the sensor's local communication range (with its neighboring sensors), namely $d_s > d_c$. To obtain connectivity we require that, on average, sensors have at least N neighbors. Given A_F, d_c and N the number of sensors needed in the field, S, is defined by:

$$S = \frac{A_F \cdot (N+1)}{\pi d_c^2} \tag{2}$$

4. Cloud formation algorithm

4.1. General

We first present the cloud formation algorithm, which identifies a *core* of sensors that are closest to the target. The algorithm does so without any distance, SNR or proximity readings, for either a static or a moving target, in the presence of false readings. When the target is moving, the cloud is continuously updated and repeatedly identifies a core. Later, in Section 5 we show how the algorithm selects only a subset of the cloud core nodes.

The algorithm is activated when the sensing status of a sensor changes, either from *sensing* to *not sensing* or vice versa. Each sensor then notifies its neighbors of the change, and the process of defining a cloud is activated. The algorithm starts by identifying the cloud edge and assigning the sensors along the edge a level of one. This can be easily done locally by the sensors that sense the target but have non-sensing neighbors. The algorithm progresses towards the cloud's core by assigning each additional sensor a level which is higher by one from the level of its own lowest leveled neighbor. This can be done again with only local status exchange among neighboring sensors. At the end of this process the center of the cloud, *the cloud's core*, has the highest level assignment. An example of the algorithm level assignment can be seen in Fig. 3.

As the target moves, new sensors enter the influence field while others stop sensing the target. This creates a change in the cloud's edge that is propagated within the cloud, forming an updated core on-the-fly, and enabling real time tracking of the target.

Fig. 1 gives a detailed description of the algorithm. In the network initialization phase (not described), sensors initialize their state (setting *level* to zero, *active* to false, and *core* to false), and find their neighbors by broadcasting a *hello* message. Once a sensor has at least 10–30% neighbors that sense the target, denoted in the algorithm by

[1] The probability to stay in the same state is 0.8 while the probability to change state is 0.2.

```
Algorithm cloud formation for node i

 1. for  own sensing status change:
 2.    broadcast  "Sensing (status)"

 3. for  "Sensing (status)" message from node j:
 4.    $N(j).s \leftarrow status$
 5.    if $(!active)$and$(|\{\imath | N(\imath).s = true\}|/|N(\imath)| \geq Th_a)$
 6.      $active \leftarrow true$
 7.      $level \leftarrow 1$
 8.      broadcast  "Level (level)"
 9.    if $(active)$and$(|\{\imath | N(\imath).s = true\}|/|N(\imath)| < Th_a)$
10.      $active \leftarrow false$
11.      $level \leftarrow 0$
12.      broadcast  "Level (level)"

13. for  "Level (lvl)" message from node j:
14.    $N(j).level \leftarrow lvl$
15.    if$(active)$
16.      $level \leftarrow \min_{\imath \in N}\{N(\imath).level\} + 1$
17.      if (level changed)
18.        broadcast  "Level (level)"
```

Fig. 1. Cloud formation algorithm.

Th_a, the sensor becomes *active* and is part of the cloud (Section 4.2 describes the trade-offs in choosing Th_a value). Sensors in the *active* state set their *level* to the minimum level of their neighbors plus one. When an *active* sensor updates its level it becomes a clouds core candidate, the *core election algorithm* will determine whether the core candidate will elect itself as a core member.

The *core* sensors are responsible to convey their position to the HS; this task is out of the scope of this paper. The algorithm is highly efficient in local messages, as each sensor sends one local message each time it updates its level.

When the target moves, sensors change (increase or decrease) their levels and some edge sensors no longer have enough sensing neighbors according to Th_a, and become *not active*. These sensors then reset their *level* to zero, and broadcast the change. New sensors become *active* and start updating their *level* when part of their neighbors enter the influence field. This constant change causes a constant *level* updating process in the cloud, resulting in a highly dynamic tracking.

If the target is still and there are no sensing changes then once the core is formed no messages are sent. The algorithm is reactivated when at least one of the sensors detects a change in its sensing and informs its neighbors.

When *active*, a sensor reacts to every *level* update. However, it changes its own level and therefore sends a message only when its minimal neighbor changes its own level. Hence, while the algorithm has high sensitivity to movements, it reacts only to movements that are approx. of the order of the low transmission range. For example, for a sensing range of 50 m and a transmission range of 8 m, a sensor will change its level on average for every 8 m of a target movement.

4.2. Dealing with sensing noise

When considering real life conditions, sensing errors must be taken into account. For example, in acoustic sensors, weather conditions often create sensing problems, due to heavy rain and lightnings. These may confuse sensors and create a *false positive* (FP) detection, in which sensors falsely report that they are sensing a target. On the other hand, even in perfect conditions, sensors within the target's influence field may not sense it, causing *false negative* (FN) readings. Specifically, for many types of sensors, the sensing probability quadratically decreases with the distance, so the furthest the sensor is from the target, the higher is the probability that although it is within the target's influence field, it would not sense the target. Our algorithm minimizes the effect of both *false positive* readings and *false negative* readings on the tracking accuracy.

To deal with the FN detection problem and make the algorithm robust to possible fluctuations in a single sensor's readings, sensors determine whether they are part of the sensing cloud (and hence *active*) based on the neighborhood sensing information, rather than their own. Under this condition, a single sensor or a small group of sensors within a sensing range but which fail to sense the target will still take part in the level negotiating process and hence will contribute to the target's tracking effort.

The FP detection problem derives from ambient conditions and therefor is uniformly distributed across the sensing field. It results in scattered clouds of low core level across the entire sensing field, as a result of the definition of a cloud. Clearly, the more sensors participates in a cloud, the better the core is defined. Hence, to become *active* a sensor needs only a few neighbors that sense the target (regardless of its own reading). To reduce the sensitivity of the algorithm to falsely sensing sensors, a sensor may become *active* and participate in the sensing cloud only if a certain percentage (determined by the parameter Th_a) of its neighbors are sensing. In a perfect conditions scenario, no sensors falsely detect a target, and therefore setting Th_a to the percentage representing a single neighbor results in a rather large cloud with a rather small core that converges very close to the target, due to symmetry. Using high Th_a values, the sensing cloud may become unsymmetrical and too small to converge, and the tracking becomes less accurate. Taking false detection into account, we found that Th_a should be set to at least 20% of a sensors' neighbors, thus reducing the algorithm sensitivity to momentary mistakes.

Fig. 2 presents the relative part of falsely activated sensors in the field as a function of the FP detection percentage, for three different Th_a values. In all these scenarios the target was tracked with a negligible tracking error as can be seen in Fig. 2b. Fig. 2 shows that for $Th_a = 30\%$, namely, a sensor becomes active if at least 30% of its neighbors sense the target, and FP readings of up to 10% of the sensors, less than 10% of the field is *active*, in this scenario almost no low core clouds are created. On the other hand, if we set Th_a to a small value of 10% we can see the increase

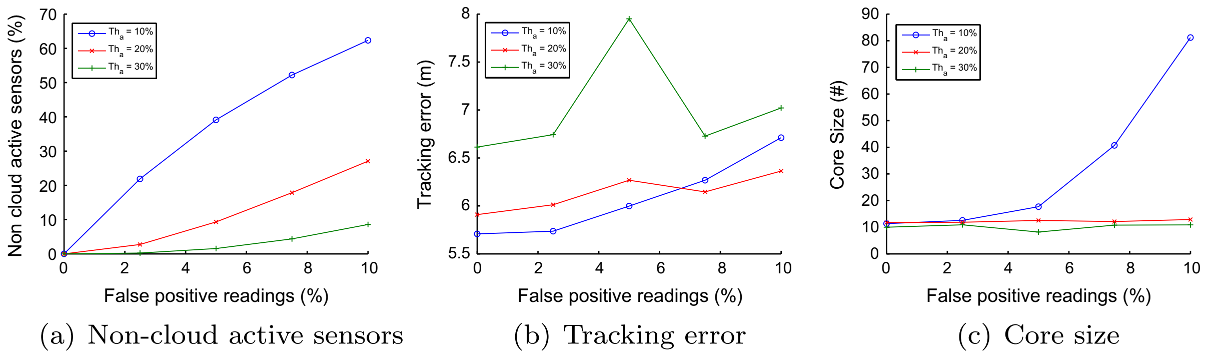


Fig. 2. Algorithm performance as a function of false positive readings for different Th_a values.

in non-cloud active sensors and according to Fig. 2c, get an increase in falsely elected core members. Since Th_a is chosen as a percentage of each sensor's neighbors, the field's density has no influence on Th_a calibration.

To help avoid the remaining low core clouds, we take a realistic approach, based on known algorithm parameters. Given the target's influence field, d_s, and the sensors local communication range, d_c, the resulted cloud has on the average $L = d_s/d_c$ different levels. Sensors are aware of both their communication range, d_c, and average sensing range for the desired target, d_s, since it is part of their design. Hence, when a sensor changes its level and becomes a core candidate but finds that its level is considerably lower than the ratio L, it can safely assume that it belongs to a cloud that is a result of some false readings or is leveled low in a target cloud, and avoid considering itself as a core member by not running *core_election* algorithm. Thus, small clouds, which are the result of FP sensing, do not elect a core and hence, do not report location to the HS. We set the L_{th} level, which determines the minimal level for a sensor to be a valid core candidate, to $L/2$. Clouds which do not have at least $L/2$ levels are then ignored. To summarize, a sensor determines that it is a core candidate and runes *core_election* only if its level is higher than the predetermined threshold, L_{th}.

Fig. 3 is a Voronoi diagram depicting a snapshot of the sensor's levels of a sensing cloud around a target during tracking. The cloud was formed under rather rough conditions with 10% of false positive (FP) reports at any given time (FP) and with the statistical FN errors described earlier. The core consists of the few sensors at level five, all closest to the target, which is denoted by a small circle, with the cloud formed around it.

Fig. 4 shows the tracking of a target under the same conditions as in Fig. 3. The target trajectory is denoted with the solid line, its speed is 60 km/h (about 17 m/s). The tracked target location, as determined by the algorithm, is denoted by the red '+' marks. In Figs. 3 and 4 the target's influence field is 50 m and the inter-sensor communication range is 10 m.

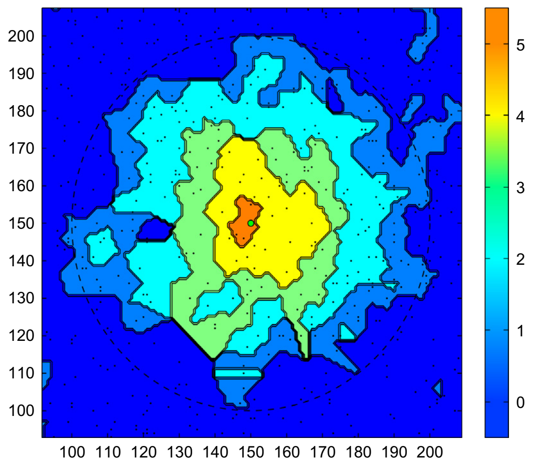


Fig. 3. Basic cloud view.

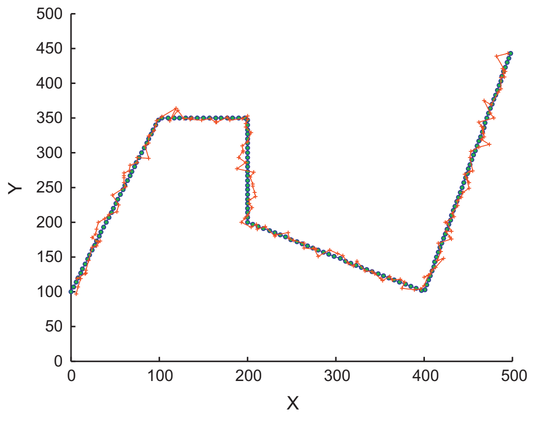


Fig. 4. Tracking example.

5. Core election algorithm

5.1. General

We present here the core election mechanisms that enable only a small subset of the sensors in the cloud, preferably the closest to the target, to inform the distant home station (HS) of the target's current position.[2] The added mechanisms will choose a group of nodes, termed

[2] As described before, in this paper we do not deal with this transmission and rely on previous solutions.

Pivots, that will be the reduced core. The additional messages used in these mechanisms enable a more educated *core* election resulting in a smaller group of sensors (Pivots) which are the closest to the target, thereby reducing dramatically the amount of high cost transmissions to the HS.

A simple approach for core selection is to find sensors with local maximum of their level parameter and select them as the core, i.e., a sensor is elected to the cloud's core if it does not hear of a direct neighbor with a higher level. Hence, right after it broadcasts its level (see Fig. 1), it compares its level to its neighbors levels, as registered in its local data, and if there is no higher leveled neighbor it sets itself as a core node and transmits to the HS. This solution does not require additional message exchange. However, the amount of transmissions to the HS (i.e., number of core node) can be rather high, deriving from a large number of sensors falsely declaring themselves as core. A false election happens when a sensor does not have immediate neighbors of a higher level, but higher leveled sensors exist in the cloud. This happens, for example, when their immediate neighbors are all of the same level, but higher level sensing sensors exist outside of their receiving range. To prevent this false self election, some means of propagating the highest level within the cloud is needed.

We compared our algorithms to the PIF flooding algorithm [24], where each node that has no neighbors of a higher level starts a flooding of its level and ID to the entire cloud. A flooding is terminated if it reached a node with a higher level. In addition to a core selection, at the end of the process the entire cloud is aware of the current highest level. However, we found that not only that this method results in a very expensive inter-sensor message exchange, it also caused a noticeable delay in the core election and update process, and many times did not manage to converge.

In our core election rule, called *Pivot core election* or *P*, the core is elected based on a global information flooding of candidates for Pivot. Like before, a sensor that does not have a neighbor with a higher level declares itself a candidate for the core, and begins a flooding process of *Pivot messages*, carrying its level across the cloud. Cloud sensors can choose a Pivot, a reference sensor that represents a better candidate to serve as part of the cloud's core (i.e., A higher leveled sensor) and avoid falsely electing themselves as core members, following these rules: If the maximal Pivot level a sensor can find is the same level as its own, the sensor will elect itself as a Pivot and inform its neighbors with a Pivot message. Sensors that do not upgrade their Pivot variable due to a Pivot message, stop propagating the message and thereby reduce the total number of messages needed to keep the cloud informed of its maximal level. Fig. 5 describes the algorithm.

In the network initialization phase (not in the figure), the sensors reset their own and their neighbors database Pivot status (setting *Pivot.id* to self or neighbor's id, *Pivot.ccl* to zero, *Pivot.dis* to zero). Upon receiving a level or Pivot message from a neighbor and only while in active state, a sensor will first execute the cloud formation algorithm (see Fig. 1) and then try to update its Pivot selection (line 4 in Fig. 5). The discussion of the full details of the Pivot selection algorithm is delayed to Section 5.2). If a sensor chooses itself as the new upgraded Pivot the sensor is elected as a *core* member.

Algorithm pivot core election for node i

```
 1. for "Level (lvl)" message from node $j$:
 2.   if($active$)
 3.     execute the cloud formation alg
 4.     $pivot \leftarrow update\_pivot()$
 5.     if ($pivot\ changed$)
 6.       broadcast "Pivot (id, ccl, dis)"
 7.     if ($pivot.id = i$)$and$(!$core$)
 8.       $core \leftarrow true$
 9.       report location ()
10.     else $core \leftarrow false$

11. for "Pivot (id, ccl, dis)" message from node $j$:
12.   $N(j).pivot \leftarrow (id, ccl, dis)$
13.   if ($active$)
14.     $pivot \leftarrow update\_pivot()$
15.     if ($pivot\ changed$)
16.       broadcast "Pivot (id, ccl, dis)"
17.     if ($pivot.id = i$)$and$(!$core$)
18.       $core \leftarrow true$
19.       report location ()
20.     else $core \leftarrow false$
```

Fig. 5. Pivot core election algorithm.

5.2. Pivot ghost information

One of the challenges of the algorithm is the constant changes in the current highest cloud level. As the target moves and the cloud evolves continuously, the number of levels in the cloud changes, e.g., the cloud's highest level can be seven at one point in time and then decrease to six. Let us now consider the Pivot election mechanism. In our algorithm, each sensor attempts to learn the highest Pivot level in the cloud to assess whether its level is the same and hence it is a Pivot. When the cloud highest level increase, it is propagated and the Pivots are updated. However, when the cloud's level decreases, we are haunted by what is termed ghost information, i.e., old information of higher level Pivot that is kept in a the cloud and prevents the new core to be set as Pivot. We term this ghost information, i.e., Pivots that no longer exist, Phantom Pivots. Let us examine the example in Fig. 6, where an arrow indicates the next hop towards the sensor's Pivot. The target is in the middle, and nodes A–H are the elected Pivots at level *l*, pointed by nodes 1–8, correspondingly. After electing the Pivot, each of the nodes 1–8 tells its neighbors about its Pivot. Now suppose, nodes A–H decrease their level to $l - 1$ and thus initiate a Pivot election process and notify their neighbor that they are no longer Pivots. Nodes 1–8 will now look for other Pivots of level *l*, and as indicated

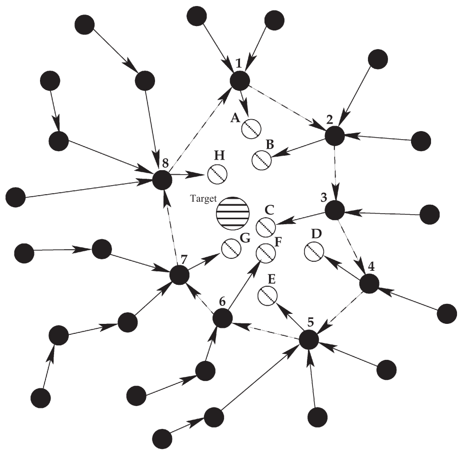


Fig. 6. Phantom Pivots in the sensing cloud.

in the figure by the dashed arrow, can form a loop that signifies the Phantom Pivot phenomenon.

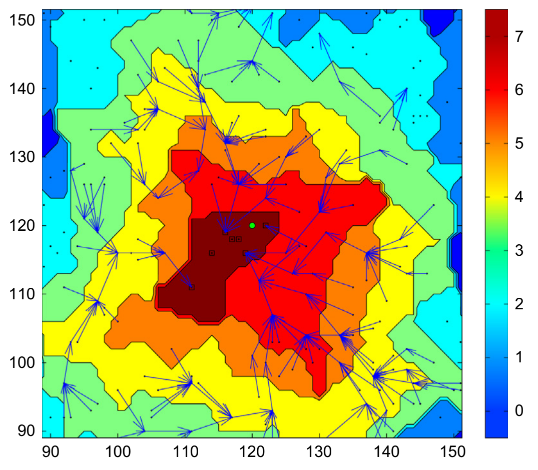
To enable sensors to differentiate between real Pivots and Phantom Pivots, we add constrains to the Pivot upgrading process. A Pivot message now carries also a distance value that is updated at each hop. When a sensor receives a Pivot message, it increases the Pivot distance number by one, and propagates the maximal Pivot value it knows of with the new distance value. In this way, each sensor can determine its distance from the Pivot. If a loop is formed this distance can grow to infinity. To prevents such loops, we limit the allowed distance to the selected Pivot by D^*. Namely, a sensor cannot choose a Pivot too distant, regardless of its level.

To set D^* we use a method we term P_{Th_p} that sets it to the difference between the sensor's and the Pivot's levels plus Th_p, a predefined threshold value.

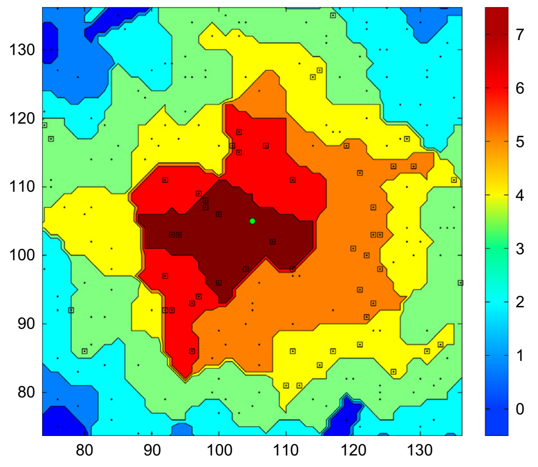
Alternately, in algorithm P_{HIS}, we propose a method to avoid *Phantom Pivots* completely by keeping track of the minimal distance a sensor heard of a specific Pivot. According to the "Source Node Condition" [25] while maintaining shortest path in a graph, if we refrain from selecting a path (or a Pivot offered by the next hop neighbor), which we know was at distance D_1in the past but is now available with distance D_2 while $D_2 > D_1$, no loop can be formed and hence no *Phantom Pivot* can keep circulating in our cloud.

Fig. 7b show a typical core election (elected core members are marked with a square) performed by the basic core election algorithm. A large group of sensors whose level is only locally maximal can be seen elected as core members due to the absence of a higher level neighbor. Fig. 7 shows a core election performed by the Pivot core election algorithm, only maximum level core members can be seen. Fig. 7 shows the Pivot selection of each sensor in the form of trees pointing to the trees Pivot.[3] Each low leveled sensor is supported by a Pivot, the target is marked

[3] The tracking using trees here has very little in common with the algorithm in [7] since here we do not have the distance information that enables an easy selection of a tree root.



(a) Pivot core election algorithm



(b) Basic core election algorithm

Fig. 7. Elected core sensors.

with a green circle. On the cloud's edge we can notice some sensors which have not yet found their supporting Pivot but present no performance degression risk since there level is lower than $L/2$.

6. Simulation results

We simulated a 500 m $\times$ 500 m field with S sensors (see Eq. (2)), uniformly distributed with an average of N neighbors per sensor. Each simulation result is the average of five random sensor placements in order to minimize sensor deployment biasing. Since the standard deviation was negligible in all runs it is omitted from the graphs. Layer 2 inter-sensor communication is not implemented in our simulations, we assume a reliable communication protocol supporting the communications and a bounded messages hop delay. We assume each sensor will wait no more than HD msec to send its pending messages due to capture [26].

Table 1 summarizes the simulation parameters we varied in the simulation. For each parameter, we selected a default value (The fixed column in Table 1) to be used

Table 1
Parameters used in the simulations.

Simulation parameter	Symbol	Value range	Fixed
Target's influence field	d_s	32 m ... 64 m	48 m
Inter-sensor comm. range	d_c	6 m ... 14 m	8 m
False positive readings	*FP*	0% ... 10%	5%
Target speed (km/h)	*Sp*	15, 30, 45, 60	15
Average number of neighbors	*N*	6 ... 14	10
Sensor activation threshold	Th_a	10% ... 30%	20%
Hop comm. delay (ms)	*HD*	20 ... 100	40

when it is not varied for a certain graph. In Section 6.1 we concentrate on simulating the Pivot history algorithm, P_{HIS}, in the following section we compare several core election algorithms.

6.1. Field parameters influence on performance

We examine the performance of the Pivot history algorithm (of course, in combination with the cloud formation) using several important metrics: The tracking error, the core size (a measure of the reporting cost to the HS), inter-sensor communication cost, and the algorithm success ratio.

Fig. 8 shows the tracking error (in meters) in a 500 m × 500 m field. We used influence fields of 32–64 m thus an error of five meters corresponds to 15%-8%.

Fig. 8 shows that the denser the sensor field, the lower is the tracking error. On the other hand, the larger the target's influence field, the larger is the cloud and its tracking error. Additionally, our simulations show that the algorithm tracks accurately even when up to 10% of the sensors falsely sense the target, and for high speeds of up to 60 kmh.

In some cases, the algorithm may fail to elect even a single core node. As FN and FP readings are changed (or the target moves) the algorithm should be able to elect a core. Our simulations show that for a connectivity level of $N \geqslant 10$ a location estimation is available (a core is elected) in more than 90% of the simulation time. However, when the average number of neighbors per sensor, *N*, is below nine the core election algorithm may succeed in as low as 65% of the time. We can still have good performance in this range if we keep the inter-sensor communication range low ($d_c \leqslant 9$) and thus enlarge sensor density in the field.

The core size is a good indicator for the number of nodes transmitting their location to the HS. We assume that if the HS accepts messages from core nodes with several level values, only the ones with the highest value are used for the location estimation.

In the simulations, we saw an increase in the number of core members with an increase in the target's influence field. The increase is due to the growth of the cloud's size, its number of level rings and hence the number of wrongly elected core member. However, our algorithm is robust, in respect to the core size and hence, the number of location report messages, to the false positive readings. No significant difference observed in the core members group size for the simulated change in the FP detection probability.

In Fig. 9 we notice an increase in core size for a denser field and, to a lesser degree, a higher number of neighbors per sensor. As the field become denser, the number of sensors residing in the highest ring in the cloud grows, and thus more core nodes are elected. The effect of the target speed on the core size is minimal up to a threshold. Moving from 45 kmh to 60 kmh we see a noticeable increase in the core size (for non-dense fields) which is due to the failure of the Pivot messages to propagate fast enough. As a result, more sensors elect themselves as core members, temporarily unaware of Pivots with a higher level.

Fig. 10 shows the average amount of local messages sent by each active sensor per second. The measurements include all active sensors in the field during an entire simulated target trajectory.

We see in Fig. 10 the decrease in the average number of messages sent per sensor with the rise in FP levels, contrary to our expectations. We noticed that the increase in the field's FP level results in an increase in the total number of messages in the field. However, the amount of sensors sharing this total number is much bigger, since FP detection causes the formation of low leveled clouds, the sensors participating in such clouds have a small

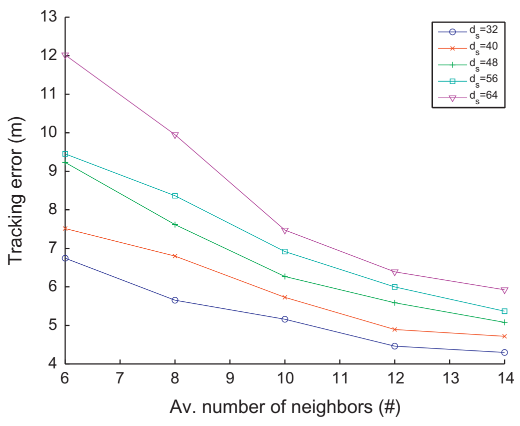


Fig. 8. Target tracking error as a function of the sensors' density for different influence fields.

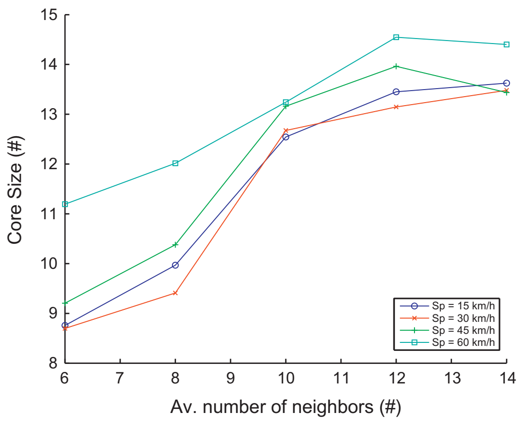


Fig. 9. Core size as a function of the average number of neighbors for different target speeds.

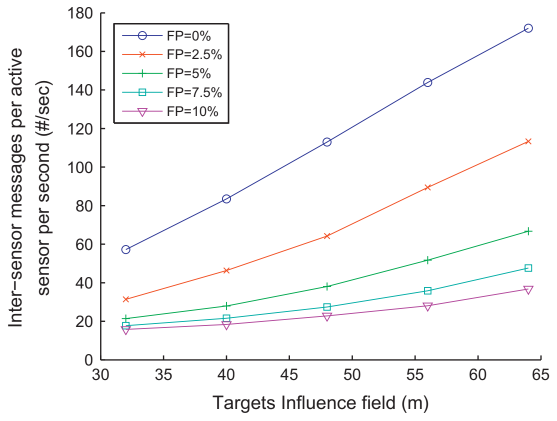


Fig. 10. Inter-sensor messages per active sensors per second as a function of the target's influence field for different false positive readings.

contribution to the messages count average because small clouds need a small number of messages to maintain themselves, thereby, reducing the total average of messages sent.

6.2. Core election algorithm influence on performance

To compare the different types of the core election algorithms, we simulated them over the same fields and compared their performance. We noticed that false positive detection level, inter-sensor message delay, and target speed had little effect on the differences in performance between the different algorithms. Thus, we concentrate here on the fields density and the target influence. Interestingly, both parameter produced showed similar influence on the algorithm performance and as a result produced similar figures, thus, we show here figures controlled only by the target's influence field.

Fig. 11 depicts the core size as a function of the influence field for the different algorithms. The P_{his} algorithm shows a great improvement over *Bsc*, and produce an almost fixed core size, regardless of the cloud size. All other algorithms show an increase in the core size when the cloud size increases. Using algorithms P_{Th_0}, P_{Th_2} and P_{Th_4} is a compromise between using a memory based algorithm, (P_{his}) with a very small core, and using the basic algorithm, (*Bsc*), without the dependency on expensive memory but with a very large core size. We can see that both algorithms P_{Th_4} and P_{Th_2} give core size values which are very close to the values of P_{his}.

Although algorithms *Bsc* and P_{his} produce a converged location estimation on more than 90% of the readings, algorithm P_{Th_4} is only around 65% reliable to produce a reading during the simulation. Algorithm P_{Th_0} is as reliable as *Bsc* but comes up with almost as large core size as does *Bsc* (figure omitted).

Fig. 12 shows the cost of this reduction in core size, the high increase in local messages between cloud members. The *Bsc* algorithm uses a minimal number of local messages, such that less than 30 messages per second are sent to maintain the core by each active sensor in the field on average. The advanced algorithms, on the other hand, forces the entire cloud to engage in the Pivots circulation process, and hence the amount of local messages is much higher, and depends heavily on the influence field (the cloud's size). As expected P_{his} produce the greatest average

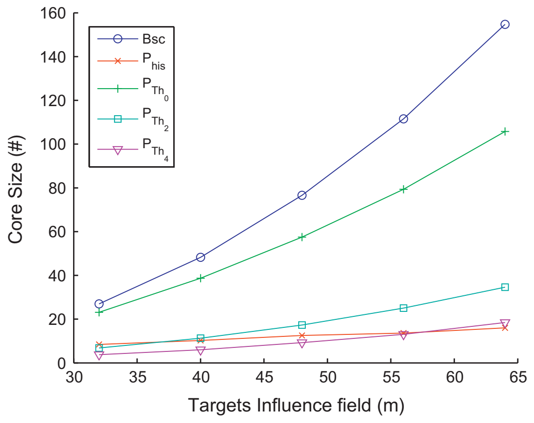


Fig. 11. Core size as a function of the target's influence field for the different election algorithms.

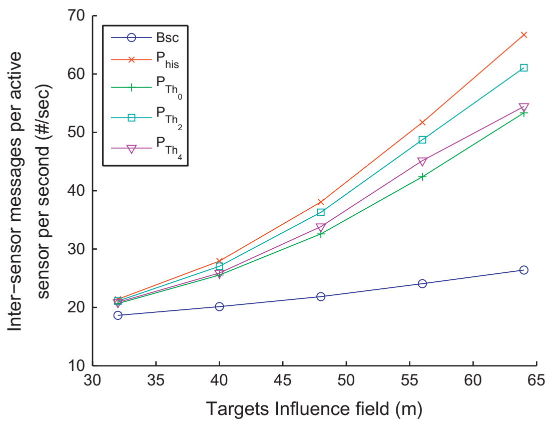


Fig. 12. Inter-sensor messages per active sensors per second as a function of the target's influence field for the different election algorithms.

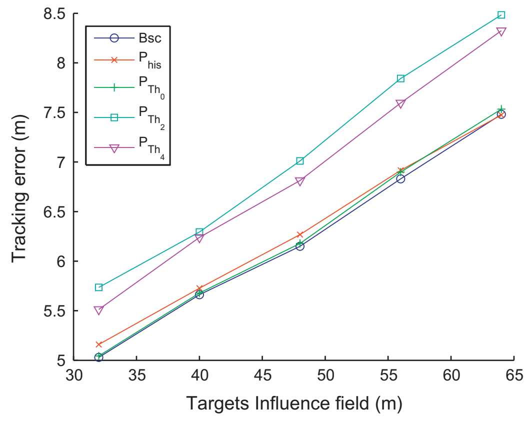


Fig. 13. Target tracking error as a function of the target's influence field for the different election algorithms.

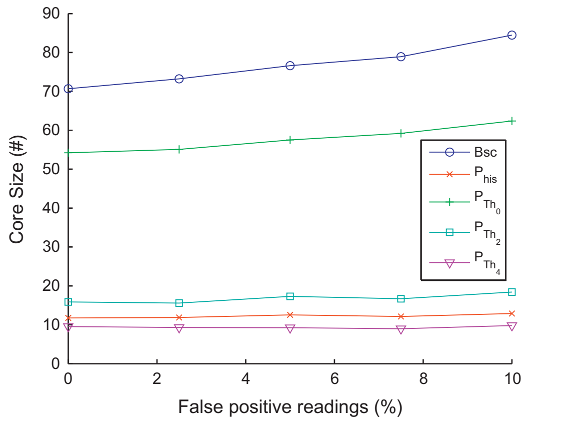


Fig. 14. Core size as a function of the false positive readings for the different election algorithms.

number of messages per sensor per second as it is obligated to update, keep track of, and circulate each Pivot declared in the cloud.

In Fig. 13 we can see that the tracking error is the smallest for algorithms *Bsc*, P_{his} and P_{Th_0} although P_{Th_2} and P_{Th_4} have only a slightly larger tracking error. An interesting advantage of the advanced election algorithms is their ability to mitigate even further the tracking sensitivity to FP readings of sensors. Fig. 14 shows that the core size remains steady (and small) for false readings of up to 10% while the core size for the *Bsc* algorithm increases by 15% for 10% of false positive detection.

7. Conclusions

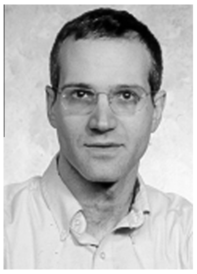
We presented, for the first time, an algorithm that enables true inexpensive binary sensors to track a target in a scenario where they are densely scattered in a field with no location knowledge. The sensors can be very noisy and use only very-low energy local communication. This paper is only a first step in this direction. We plan to further use history for tracking multiple targets whose clouds may be joint at times.

References

[1] B. Rippin, Pearls of Wisdom wireless networks of miniaturized sensors, in: Proc. SPIE 8388, Unattended Ground, Sea, and Air Sensor Technologies and Applications XIV, Baltimore, MD, USA, 2012.
[2] J. Kahn, R. Katz, K. Pister, Next century challenges: mobile networking for Smart Dust, in: The 5th annual ACM/IEEE International Conference on Mobile Computing and Networking, 1999, pp. 271–278.
[3] F. Ye, G. Zhong, S. Lu, L. Zhang, Gradient broadcast: a robust data delivery protocol for large scale sensor networks, Wireless Networks 11 (3) (2005) 285–298.
[4] Y. Sankarasubramaniam, Ö. Akan, I. Akyildiz, ESRT: event-to-sink reliable transport in wireless sensor networks, in: The 4th ACM International Symposium on Mobile Ad hoc Networking & Computing, 2003, pp. 177–188.
[5] S. Arnon, S. Dolev, R. Kat, D. Kedar, Searching for a lion in the desert: optics-based acquisition algorithms for wireless sensor networks, ACM SIGMOBILE Mobile Computing and Communications Review 12 (4) (2008) 32–42.
[6] H. Kung, D. Vlah, Efficient location tracking using sensor networks, in: IEEE Wireless Communications and Networking Conference (WCNC), 2003.
[7] W. Zhang, G. Cao, DCTC: dynamic convoy tree-based collaboration for target tracking in sensor networks, IEEE Transactions on Wireless Communications 3 (5) (2004) 1689.
[8] S. Martínez, F. Bullo, Optimal sensor placement and motion coordination for target tracking, Automatica 42 (4) (2006) 661–668.
[9] C. Lin, W. Peng, Y. Tseng, Efficient in-network moving object tracking in wireless sensor networks, IEEE Transactions on Mobile Computing 5 (8) (2006) 1044–1056.
[10] Y. Cai, W. Lou, M. Li, and X. Li, Target-oriented scheduling in directional sensor networks, in: IEEE INFOCOM'07, 2007, pp. 1550–1558.
[11] L. Song, D. Hatzinakos, A cross-layer architecture of wireless sensor networks for target tracking, IEEE/ACM Transactions on Networking 15 (1) (2007) 145–158.
[12] J. Xu, X. Tang, W. Lee, A new storage scheme for approximate location queries in object-tracking sensor networks, IEEE Transactions on Parallel and Distributed Systems 19 (2) (2008) 262–275.
[13] K. Shih, S. Wang, H. Chen, P. Yang, CollECT: collaborative event detection and tracking in wireless heterogeneous sensor networks, Computer Communications 31 (14) (2008) 3124–3136.
[14] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, D. Rus, Tracking a moving object with a binary sensor network, in: The 1st International Conference on Embedded Networked Sensor Systems, 2003, pp. 150–161.
[15] N. Shrivastava, R. Madhow, S. Suri, Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms, in: The 4th International Conference on Embedded Networked Sensor Systems, 2006, pp. 251–264.
[16] K. Mechitov, S. Sundresh, Y. Kwon, G. Agha, Cooperative tracking with binary-detection sensor networks (Poster abstract), in: The 1st International Conference on Embedded Networked Sensor Systems, ACM, 2003, p. 333.
[17] Z. Wang, E. Bulut, B. Szymanski, A distributed cooperative target tracking with binary sensor networks, in: ICC Workshops, 2008, pp. 306–310.
[18] Z. Wang, E. Bulut, B. Szymanski, Distributed target tracking using binary sensors with imprecise range measurements, in: The Second Annual Conference of the International Technology Alliance (ACITA), 2008.
[19] J. Liu, J. Liu, J. Reich, P. Cheung, F. Zhao, Distributed group management for track initiation and maintenance in target localization applications, in: IPSN, lNCS 2634, 2003, pp. 113–128.
[20] J. Lee, K. Cho, S. Lee, T. Kwon, Y. Choi, Distributed and energy-efficient target localization and tracking in wireless sensor networks, Computer Communications 29 (13–14) (2006) 2494–2505.
[21] L. Xiangqian, Z. Gang, M. Xiaoli, Target localization and tracking in noisy binary sensor networks with known spatial topology, Wireless Communications and Mobile Computing 9 (8) (2009) 1028–1039.
[22] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, et al., A line in the sand: a wireless sensor network for target detection, classification, and tracking, Computer Networks 46 (5) (2004) 605–634.
[23] P. Varshney, C. Burrus, Distributed Detection and Data Fusion, Springer Verlag, 1997.
[24] A. Segall, Distributed network protocols, IEEE Transaction on Information Theory IT-29 (1) (1983) 23–35.
[25] J.J. Garcia-Lunes-Aceves, Loop-free routing using diffusing computations, IEEE/ACM Transactions on Networking 1 (1) (1993) 130–141.
[26] R. Rom, M. Sidi, Multiple Access Protocols: Performance and Analysis, Springer-Verlag, 1990.

Tal Marian received his B.Sc. in Electrical Engineering from Tel-Aviv University, in 2009 (cum laude), and his M.Sc. from Tel-Aviv University, in 2010 (cum laude). His research focused on sensor networks, object tracking and communication algorithms. He currently serves in a managerial position in the start-up industry.

Osnat (Ossi) Mokryn Received the B.Sc. in Computer Sceince and M.Sc. in Electrical Engineering from the Technion, Israel Institute of Technology, Haifa, Israel, in 1993 and 1998, respectively. She received her Ph.D. in Computer Science from the Hebrew University, Jerusalem, Israel, in 2004. Ossi has an extensive high tech experience from leading companies such as Intel and IBM research labs in Haifa, Israel. She was a posDoctorate fellow in the electrical Engineering Departments at Tel-Aviv University, Israel and also at the Technion, Haifa, Israel. Starting 2008 She is a faculty member at the School of Computer Science in Tel Aviv Jaffa Academic College, Tel Aviv, Israel. Her recent research focuses on network protocols, P2P streaming, ad hoc and sensor networks and web data mining.



Yuval Shavitt received the B.Sc. degree in computer engineering (cum laude), the M.Sc. degree in electrical engineering, and the D.Sc. degree from the Technion, Israel Institute of Technology, Haifa, in 1986, 1992, and 1996, respectively. From 1986 to 1991, he served in the Israel Defense Forces first as a System Engineer and the last two years as a Software Engineering Team Leader. After graduation, he spent a year as a Postdoctoral Fellow at the Department of Computer Science at Johns Hopkins University, Baltimore, MD. Between 1997 and 2001, he was a Member of Technical Staff at the Networking Research Laboratory at Bell Labs, Lucent Technologies, Holmdel, NJ. Since October 2000, he has been a Faculty Member in the School of Electrical Engineering at Tel-Aviv University, Tel-Aviv, Israel. He was an Editor of Computer Networks from 2003 to 2004, and served as a Guest Editor for JSAC and JWWW. His recent research focuses on Internet measurement, mapping, and characterization; ad hoc networks, and datamining social networks.