Wireless Netw (2012) 18:713–733
DOI 10.1007/s11276-012-0471-y

Energy-efficient information routing in sensor networks for robotic target tracking

Jason M. O'Kane · Wenyuan Xu

Published online: 20 June 2012

Abstract Target tracking problems have been studied for both robots and sensor networks. However, existing robotic target tracking algorithms require the tracker to have access to information-rich sensors, and may have difficulty recovering when the target is out of the tracker's sensing range. In this paper, we present a target tracking algorithm that combines an extremely simple mobile robot with a networked collection of wireless sensor nodes, each of which is equipped with an unreliable, limited-range, boolean sensor for detecting the target. The tracker maintains close proximity to the target using only information sensed by the network, and can effectively recover from temporarily losing track of the target. We present two algorithms that manage message delivery on this network. The first, which is appropriate for memoryless sensor nodes, is based on dynamic adjustments to the time-to-live (TTL) of transmitted messages. The second, for more capable sensor nodes, makes message delivery decisions on-the-fly based on geometric considerations driven by the messages' content. We present an implementation along with simulation results. The results show that our system achieves both good tracking precision and low energy consumption.

Keywords Sensor networks · Routing · Robotic target tracking

J. M. O'Kane · W. Xu (✉)
Department of Computer Science and Engineering, University of South Carolina Columbia, Columbia, SC 29208, USA
e-mail: wyxu@cse.sc.edu

J. M. O'Kane
e-mail: jokane@cse.sc.edu

1 Introduction

Tracking problems for mobile robots have received substantial attention in recent years [1–5]. In these problems, a robot *tracker* seeks to maintain close proximity to an unpredictable *target*. Effective target tracking algorithms have many important applications, including monitoring and security. Another potential application is that the tracker needs to carry cargo for the target and has to 'follow' the target in real time. We note that this type of robotic tracking problems is different from the sensor network target tracking problem in which the main goal is to identify the moving trajectory of the target. In this paper, we focus on the robotic tracking problem. Algorithms have been proposed to solve this problem with mobile robots under various constraints and sensor models [1–5]. However, these existing methods for robotic tracking are hampered by two primary limitations.

1. *Locality:* Existing tracking methods generally rely on sensors onboard the robot, which by nature only provide information about the target's location when the target is nearby. This limitation is particularly problematic in cases where (a) the tracker starts with little or no knowledge of the target's location or (b) the tracker loses contact with the target during its execution. To recover from these situations using only local information is a challenging problem, requiring extensive search in the worst case [6].
2. *Sensing complexity:* Prior work assumes that the robot has access to sensors that are (in spite of their local nature) relatively powerful and information-rich, such as visual or range sensors [1, 4, 5]. Such sensing capabilities add additional cost and complexity to the robot. It is desirable to design and deploy simpler

robots with less sophisticated sensing hardware. Moreover, tracking with limited sensing is of independent interest for cultivating understanding of the information requirements of the target tracking task.

In this paper, we propose a tracking framework that resolves these limitations by allowing the robot to utilize a wireless sensor network to assist in the tracking task. The tracking task can be divided into two parts: sensing the target and following its movements. As such, we decouple these parts and delegate the sensing task to a stationary sensor network. The mobile tracker then follows the target using only the observations made by these sensor nodes. Such an arrangement has several advantages. Firstly, it eliminates the need for complex sensors on the tracker, which relaxes the hardware requirements of the tracker and simplifies the sensing data processing algorithm that is needed to filter out the difference of consecutive readings caused by the tracker's movement [7] (e.g., laser or camera readings). Secondly, it provides a means for delivering nonlocal information to the tracker, which is critical to help the tracker recover when it loses contact with the target, for example.

To further simplify the proposed system, we assume the sensor nodes are equipped with binary proximity sensors that cannot sense the accurate location of the target. Instead, the sensors report only whether the target is within a given sensing range, and these sensors may experience frequent errors, including both false positives and false negatives. A variety of proximity sensors could be used. For instance, it could be an inductive proximity sensor [8] that can detect metal targets, or a photoelectric sensor [9] for non-metal targets.

The coarse and unreliable measurements provided by such sensors, combined with the sensor nodes' resource-constrained characteristics, impose new challenges in designing a real-time tracking algorithm. In particular, the tracking algorithm should address two sub-problems: How to control the tracker utilizing coarse location information provided by sensor nodes? How to deliver sensing data from the sensor nodes to the tracker with high energy-efficiency?

To control the tracker utilizing the coarse location information, our approach makes extensive use of the concept of *information states* [10], which explicitly encode the robot's uncertainty about the target position. Specifically, the tracking robot uses information collected from the network to synthesize a set of *possible states*, then makes greedy motions intended to reduce the size of this set. With regard to designing a lightweight, energy-efficient protocol, we focus on designing cross layer routing protocols that limit the total number of message transmissions in spite of the high mobility of the tracker. In particular, to eliminate the overhead of route discoveries and route updates, our routing protocols leverage flooding, but the scope of flooding is strictly controlled. In summary, the contributions of this study include the following.

- We design a unique architecture for robotic tracking, whereby a robot cooperates with a sensor network with the following features: Nothing is known about the target's motion other than its maximum speed; The tracking robot has no sensors that directly provide information about the target; The sensor nodes detect only when the target is nearby, but do not provide any precise location information, and are subject to frequent, unpredictable failures; and Each sensor node has a limited energy budget for making transmissions. Adopting such an architecture allows us to keep both the robot and the sensor network lightweight and enables us to understand the cooperation between robots and sensor networks with regard to tracking performance.
- We devise two energy-efficient, low-maintenance, and robust routing protocols that can forward information towards a mobile tracker. Both routing protocols leverage cross layer information (i.e., tracking information in the application layer) to reduce routing overheads. Neither protocol requires periodic beacon exchanges between neighbors, and both are robust to a wide range of network dynamics, e.g, sensor nodes may fail or even move. Furthermore, both of our protocols maintain little information to support routing. The first algorithm uses a TCP-style technique of controlled flooding, in which each node maintains a TTL value to be used for its next transmission, and the TTL value roughly estimates the hop count between the tracker and the target. In the second algorithm, each node keeps track of a circular portion of the network field known to contain the target and another circle known to contain the target, and makes routing decisions based on information about the shortest paths between these circles.
- Through our extensive simulation study, we demonstrate that for such systems, both tracking performance and energy-efficient routing protocols can be achieved simultaneously. We also demonstrate that the addition of greater computing power and memory to the sensor nodes enables smarter algorithms that improve the performance even more in certain circumstances.

The remainder of this paper has the following structure. We begin the paper by reviewing related work in Sect. 2. We next formalize the tracking problem in Sect. 3. We present our approach, including the strategy for controlling the tracker (Sect. 4) and the two routing protocols to deliver sensing data to the tracker (Sect. 5). In Sect. 6, we

present our validation effort and discuss experimental results. Extensions that overcome false positive errors in the sensors appear in Sect. 7. We end the paper by providing concluding remarks in Sect. 8. Early versions of this work appeared at ICRA 2009 [11] and at IROS 2010 [12].

2 Related work

2.1 Robotic target tracking

Target tracking problems for mobile robots have been studied for some time. The objective for these problems is generally to maintain visibility between the target and the tracker. Algorithms are known for planning the tracker's motions using dynamic programming [2], sampling-based [5], and reactive approaches [3]. Others have considered the problem of stealth tracking, in which the tracker seeks to maintain visibility of the target, while remaining near the boundary of the target's visibility polygon to avoid possible detection [1]. Still another approach explicitly considers the target's privacy in the formulation [13]. Our work is also related to algorithms for pursuit-evasion, which consider the problem of locating adversarial mobile agents within an environment [14–17].

2.2 Sensor network target tracking

Wireless sensor networks (WSNs) have been deployed to track the positions of humans [18, 19], moving vehicles [20, 21], and other moving targets [22, 23, 24]. Those surveillance systems leverage stationary sensor networks in which each node collects measurements using on-board sensors and reports the measurements to a central data collection unit (that is, the sink) via multi-hop routing. Alternatively, to keep track of locations of targets, sensors are attached to the moving targets, such as zebras [25]. Whether stationary or attached to targets, sensor nodes passively collect measurements and rely on multi-hop communication to deliver data to the central data collection unit for further analysis. As a result, the communication can become expensive when the network size is large. The tracking architecture proposed in this work addresses such communication issues by having a mobile tracker follow a target and collect the information from the target in its vicinity. Utilizing a similar architecture, Tsai et al. [26] has proposed to let the mobile tracker query the target location by flooding the entire networks. Then, a 'near-node' responds after discovering a route to the tracker. Unlike their work, our routing protocols can deliver information to the mobile tracker without route discovery or neighbour awareness.

The use of mobile sensor networks, in which individual nodes have both sensing and motion capability, has also been proposed as a means to track moving targets [27, 28]. The primary concern in this area is to track the targets while maintaining the network connectivity. We propose a different architecture in which the connectivity problem and mobility management issues are decoupled.

The binary proximity sensors we use have been previously studied for other tasks including target counting [29, 30] and pursuit-evasion [17]. Shrivastava, Mudumbia, Madhow, and Suri consider the problem of reconstructing the path of a target moving through a field of such sensors [31]. Our work is complementary in the sense that we are concerned with communication issues they do not address.

2.3 Routing in sensor networks and MANET

Routing has been studied extensively, since it is a basic building block of networking. In the area of sensor networks, spanning tree based routing [32] builds a routing tree rooted at the sink prior to data delivery. Such protocols work well with stationary sinks, but are unsuitable for a mobile receiver. In the area of wireless sensor networks with mobile sinks, Kusy et al. [33] proposed to leverage mobility prediction to compute fresh routes to a mobile sink before old routes become useless. Ye et al. [34] designed a two tier data dissemination model. Their networks are formed in a grid structure and messages are first routed between cells then flooded inside the cell that contains the mobile sink. In our network, sensors are not required to predict the mobility of the sink, nor to form a hierarchical multi-tier structure.

Another category of routing in sensor networks is called data-centric routing, where data is searched or stored based on their name rather than the network addresses of nodes. Essentially, the data-centric routing is a query problem, whereby route discovery is initiated by queries, and then data of interest will be collected via the discovered route. To facilitate data searching, some of those routing algorithms depend on flooding to propagate queries [35], some cleverly store the data in a curve with which the query messages are guaranteed to intersect [36], and some map the keyword of the data to a deterministic location for storing based on geographic hash functions [37]. In comparison, our robotic tracking problem does not involve queries, but focuses on delivering messages to a mobile tracker efficiently.

In the area of mobile ad hoc networks (MANET), Ad hoc On-demand Distance Vector (AODV) [38] and Optimized Link State Routing (OLSR) [39] are two well-known routing protocols. AODV works reactively, and it discovers routes only if they are needed, while OLSR proactively maintains and updates path selection regardless of

being used or not. Those protocols either impose high latency for the initial path setup or lead to a wasteful overhead of routing traffic, and therefore are not suitable for a network with a highly mobile receiver and battery-operated sensors.

Additionally, several routing algorithms exploiting geographic information have been proposed. Those geographic routing algorithms [40, 41] refer to destinations by their location, and forward messages greedily, when possible, towards the destination, e.g., to a node that is closest to the destination. Such routing algorithms cannot be applied to the robotic tracking problem directly, because they require the precise location information of destinations and focus on delivering messages to the destination location. However, the goal of our routing algorithm is to deliver messages to the mobile tracker instead of a specific location.

2.4 Data aggregation in sensor networks

The goal of in-network data aggregation [42] in WSNs is to reduce expensive transmission. Typically, the data aggregation algorithm, such as TAG [42], routes the aggregated values up towards the root of the routing tree with partial data aggregated at internal tree nodes. That algorithm relies on the routing tree that is established prior to the data aggregation process. The smart network concept presented in this paper is different in the sense that the in-network state computation is used to assist route selection, so that messages are delivered efficiently between the sensors near the target and the ones near the tracker.

2.5 Cooperative robot-network systems

The idea of combining WSNs with mobile robots has been investigated by Sukhatme et al. [7, 43–45]. In particular, mobile robots are used for sensor network deployment with the goal of achieving good sensor coverage [43, 44]. Our work complements theirs in the sense that we focus on the tracking application after the deployment is done. WSNs are also proposed to assist mobile robots to track targets [7], using the sensors that can supply precise location information to the robot. We take a different viewpoint: we design the tracking algorithms by considering issues associated with both sensor networks and mobile robots; and thus, achieve good tracking performance at reasonable operational cost for the network while using simple sensor devices and robots.

3 Problem statement

Before discussing the control algorithms for the robot tracker and routing protocols for the sensor nodes in our tracking framework, this section formalizes our target tracking problem and the performance criteria we use to measure the success of our algorithms.

3.1 System model

A point target moves unpredictably, but with maximum speed s_{tgt}, in a closed, bounded, polygonal, planar environment $E \subset \mathbb{R}^2$. The environment need not be simply connected. Time is modeled as a continuous progression along the interval $[0, T]$. Let $q(t) \in E$ denote the position of the target at time $t \in [0, T]$.

A point robot called the tracker also moves in E. At time t, the position of the tracker is denoted $p(t)$. The tracker can choose its velocity vector $u(t)$, so that $d\ p\ /\ dt = u$. The velocity is constrained by a maximum speed s_{trk}. The tracker knows its position $p(t)$ within E, using either standard sensor-based localization methods [46, 47], or GPS [48]. The tracker has no sensors that directly report on the position of the target; it instead must rely solely on the location information provided by the network, as described below. The *state* $x(t) = (p(t), q(t))$ comprises the tracker and target positions.

To assist the tracker, a network of k stationary wireless sensor nodes is distributed through E at positions $n_1, \ldots, n_k$. The nodes localize themselves using one of the well-known sensor network localization schemes [49–52]. To simplify the notation, we assume that the nodes are identical, with a fixed sensing range r_s and a fixed communication range r_c. Each node n_i can:

1. Possibly **detect the target** whenever $\|q(t) - n_i\| \leq r_s$, where $\|\ *\ \|$ is the Euclidean distance. This sensing is boolean: The node knows only whether or not the target has been detected, but no other information. This detection is also unreliable, in the sense that failing to detect the target does not imply that $\|q(t) - n_i\| > r_s$. Such false negatives, which can occur as a result of unmodeled occlusions in the environment, noise, or other factors, are assumed to be extremely common. To simplify the discussion, we assume for now that false positive errors—in which the sensor reports incorrectly that the target is nearby—do not occur. Section 7 lifts this assumption.
2. **Broadcast a message** to all nodes n_j for which $\|n_i - n_j\| \leq r_c$. We assume that the time required for each transmission is negligible compared to the physical speeds of the robots, and these broadcasts are subject to intermittent communication failures. In our algorithm, the content of these messages is a description of the information the node has accumulated, along with a timestamp indicating when that knowledge was collected. Specifically, each message is an ordered pair

(η, t), in which η is a description of an *information state* (I-state) and $t \in [0, T]$ is a timestamp. Depending on the message delivery scheme, η can be a circle known to contain the target (as described in Sect. 4.1) or a disk-based I-state describing both the tracker and the target locations (Sect. 5.2).

In addition, the tracker is equipped with network communication hardware, so that it can receive messages that are broadcast by nodes within r_c of $p(t)$. The tracker also uses this hardware to transmit a beacon that informs the wireless sensor nodes of its presence. This beacon is detected by the node at n_i whenever $\|p(t) - n_i\| \leq r_c$. As with the target detection sensors, receipt of this beacon signal is subject to frequent communication failures.

3.2 Evaluation criteria

The execution of the proposed tracking framework can be informally divided into two phases: (1) a *startup* phase, during which the tracker works to reduce a relatively large distance between itself and the target, and (2) a *steady state* phase, during which the tracker has moved near the target and works to maintain this proximity. To evaluate the performance in both phases, we define the following criteria.

3.2.1 Tracking performance P

Regardless of whether it is in the startup or steady state phase, the tracker's primary objective is to minimize the average distance between $p(t)$ and $q(t)$ throughout the system's execution:

$$P = \frac{1}{T}\int_0^T ||p(t) - q(t)||dt. \quad (1)$$

Besides the average distance between the target and the tracker, the maximum distance between them over the lifetime of the system's execution is also an important performance metric, since it may be difficult for the tracker to re-acquire the target's location. Thus, we define the worst case performance as

$$P_{max} = \max_{t\in[0,T]} ||p(t) - q(t)||. \quad (2)$$

3.2.2 Tracking startup time S

During the startup phase, the tracking framework naturally performs worse than during the steady phase. The definition of P averages out the performance impact of the startup phase as T increases. Therefore, we also consider the *capture time* S, which measures the length of the startup phase, and is defined by

$$S = \min\{t \in [0, T] \mid ||p(t) - q(t)|| \leq \epsilon\}, \quad (3)$$

in which ϵ is a small positive constant.

3.2.3 Energy cost C

In both phases, because the energy available to each wireless sensor node is generally limited by battery capacity, a secondary objective is to minimize the average number of message broadcasts made in the network per unit time. Let $C(i)$ denote the number of broadcasts made by the node at n_i between $t = 0$ and $t = T$. The system seeks to keep

$$C = \frac{1}{T}\sum_{i=1}^{k} C(i) \quad (4)$$

as small as possible. We note that the communication energy cost includes both transmitting and receiving cost. However, $C(i)$ can be easily scaled up to reflect energy consumed by receiving by multiplying a coefficient, since on average each broadcast is received by a fixed number of nodes that are evenly distributed.

These three performance criteria, P, S, and C, are at least partially in conflict with one another: Intuition suggests—and our experiments confirm—that decreases in P or S generally require corresponding increases in C. This tradeoff motivates our use of Pareto optimality concepts, treating the problem as a multi-objective optimization problem. Section 6 presents and discusses these results.

4 Controlling the tracker

This section describes the algorithms used by the tracker to respond to the data it receives from the network.

Since the current state $x(t)$ is not necessarily known, the tracker faces two challenges: First, it must efficiently represent the limited knowledge it has about the target's position. Second, it must use this representation to plan its motion toward the target. Our approach overcomes these challenges using the the tracker's *information states* (I-states). In this context, the I-state is the set of possible states that are consistent with the information the tracker has received. The tracker computes its information state, then chooses its motions as a function of this current information state.

We first describe how to maintain the information state (Sect. 4.1), then we present strategies used by the tracker to utilize this information (Sect. 4.2). Note that we are concerned here only with the control of the tracker; we defer to Sect. 5 our discussion of approaches to efficiently deliver target location information to that tracker.

4.1 Uncertainty and information states

4.1.1 Modeling the uncertainty

The primary challenge in this target tracking problem is the *uncertainty* throughout the system. The tracker does not know the target's precise location, and instead must rely in the history of messages it has received to direct its motions.

To overcome this uncertainty, we use the concept of *information states* (I-states). Although the information space framework is quite general and flexible,[1] in this paper we use only *nondeterministic* I-states. Informally, a nondeterministic I-state is a set of "possible states" consistent with observations of the tracking robot or a sensor node. Let $\mathcal{I}$ denote the space of all such information states.

4.1.2 Computing the information state

Formally, suppose that tracker has received m messages

$$\{(c_1, t_1), \ldots, (c_m, t_m)\}, \tag{5}$$

as of some time t_f, with each c_i describing a circle known to contain the target at time t_i, and with $0 < t_1 < \ldots < t_m < t_f$. Each c_i is either the sensing disk (in our Dynamic TTL algorithm, Sect. 5.1) or a disk-based I-state (in our Smart Local Routing algorithm, Sect. 5.2) of one sensor node. Then a target position q' is *consistent with* those messages if and only if there exists a continuous trajectory $q{:}[0,t_f] \to E$ such that $\|d\,q\,/\,d\,t\| \leq s_{\text{tgt}}$ for all $t \in (0, t_f)$, and we have $\|q(t_i) - c_i\| < r_s$ for $1 \leq i \leq m$, with $q(t_f) = q'$. Figure 2 illustrates this definition. Note the implicit assumption that the tracker starts with no information about the target's location. We use the notation $Q(t)$ for the set of target positions consistent with the messages received by the tracker up to time t. The tracker always knows its own position, so the information state (that is, the set of possible states) at time t is

$$\eta(t) = \{p(t)\} \times Q(t). \tag{6}$$

Observe that, because of the definitions, we always have $x(t) \in \eta(t)$. However, because the definition of $\eta(t)$ contains an existential quantifier ("there exists") over target trajectories, it is not directly useful for computing the information states. Instead we perform iterative updates, maintaining the current information state and updating it when time passes and when new messages are received. We start with the initial information state $\eta(0) = \{\ p(0)\ \} \times E$. Then two kinds of updates are performed throughout the execution:

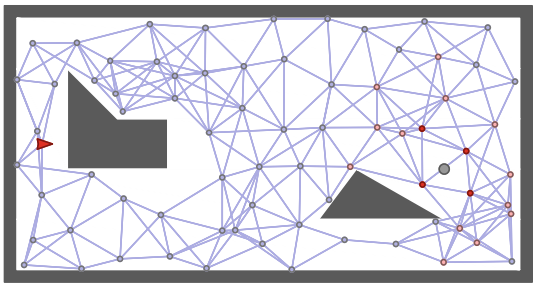


Fig. 1 An example tracking problem, in which a tracker (*triangle, left side*) seeks to find and maintain close proximity to a target (*circle, right side*). A wireless sensor network deployed in the environment provides the tracker with information about the target's whereabouts. Edges connect nodes within communication range of one another

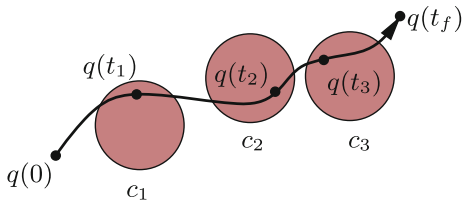


Fig. 2 A target position is consistent with a set of messages if there exists a starting target position and a feasible target trajectory that pass through each message's *circle* at the appropriate times reaching that target position

1. When time from t_1 to t_2 passes without any messages being received, we compute $\eta(t_2)$ from $\eta(t_1)$. To accomplish this we replace $p(t_1)$ with $p(t_2)$, perform a Minkowski sum of $Q(t_1)$ with a disc of radius $(t_2 - t_1)$ s_{tgt}, and intersect the resulting region with E. The resulting region is retained as $Q(t_2)$. Note that this approach may slightly overestimate the information state when $t_2 - t_1$ is large and the boundary of E has sharp non-convex corners. This effect, which is similar to the sampling issues that arise in collision detection for path segments, can be reduced or eliminated by partitioning the time period from t_1 to t_2 into smaller segments.
2. When a message (c, t) is received, the existing information state is updated to the correct $\eta(t)$ by performing an intersection with a disk with center c and radius r_s.

Figure 3 illustrates each of these updates. If the I-state is represented as polygon with n vertices along its boundary and the discs are approximated as regular polygons with m vertices, then each of these operations takes time $O(mn)$ [53, 54]. Our implementation the GPC General Polygon Clipper Library [55], with $m = 125$, to perform both types of updates.

[1] See, for example, Chapters 11 and 12 of LaValle's book [10].

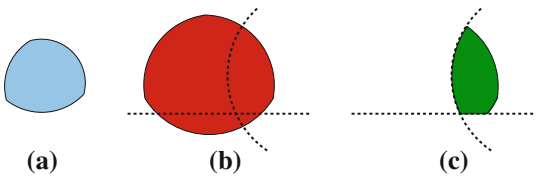


Fig. 3 Computing the information state. **a** An initial information state. **b** Expansion to account for the passage of time, and intersection with received message circles and the environment. **c** The resulting updated information state

4.2 Tracker strategy

We now describe how the tracker moves. Because the information state is a complete picture of the knowledge available to the tracker, we describe the tracker's strategy as a function mapping information states to tracker velocities:

$$\pi : \mathcal{I} \rightarrow \{u \in \mathbb{R}^2 | ||u|| \leq s_{\text{trk}}\}. \quad (7)$$

Therefore, the tracker's chosen velocity $u(t)$ is a function of its information state $\eta(t) = (p(t), Q(t))$. Notice that, aside from knowing its own position and a set of possibilities $Q(t)$ for the target's position, the tracker cannot draw any additional conclusions about the state. Given this uncertainty, the ideal position for that tracker, that minimizes the average distance to the target across all its possible positions is, by definition, the centroid of $Q(t)$. Note, however, that the centroid of $Q(t)$ may not be inside E. Based on these observations, we use the following strategy for the tracker:

> *Move with speed s_{trk} along the shortest path in E from $p(t)$ to the point in E that is closest to the centroid of $Q(t)$.*

Computing this motion takes time linear in the complexity of $Q(t)$, for both the centroid and shortest path elements [56]. Each time the I-state is updated, the tracker's motion plan is also recomputed accordingly.

5 Sensing and data delivery

As discussed in Sect. 3, we consider a network of k nodes spread throughout E. Whenever a node detects the target, it may generate a message and initiate an attempt to deliver the message to the tracker. The design of the message delivery protocol is complicated by two factors. First, the location of the intended receiver, i.e., the tracker, is unknown in general to the network nodes, which makes protocols designed for stationary sensor networks inapplicable. Second, the sensor nodes have extremely constrained energy budget, so MANET routing protocols are not suitable because of their high overhead in route discoveries and route updates. As a result, instead of establishing routes ahead of time, our method leverages the tracking information in the application layer to broadcast messages in a strictly controlled way.

Instead of attempting to design a "one size fits all" protocol, we categorize sensors into two types and devise differing routing algorithms for each type: (1) one for sensors with almost no memory nor computation capability (Sect. 5.1); and (2) one for sensors with memory and moderate computation capability (Sect. 5.2)

5.1 Dynamic-TTL broadcasting for memoryless sensors

The message delivery protocol for memoryless sensors leverages the concept of time-to-live (TTL). Instead of using intermediate sensors to remember the state or to perform substantial computation, routing information is stored in the message header. Each message header contains a *sequence number*, the source node identifier, and a nonnegative integer *time-to-live* (TTL). We use these headers in two ways:

- A node will only forward each message once, using the sequence number and the source node identifier to track which messages it has forwarded already.
- Each time a message is forwarded, its TTL is decreased by 1. Messages with TTL of 0 are discarded. As a result, the initial TTL for a node determines the maximum number of hops the message can travel on the network.

Each node forwards every message it receives as long as neither of these drop conditions is met.

Figure 4 depicts an example of sensor nodes along with the tracker. Suppose that node G detects the presence of the target. Node G generates a message with TTL =2, a new sequence number, and its location, and sends out the

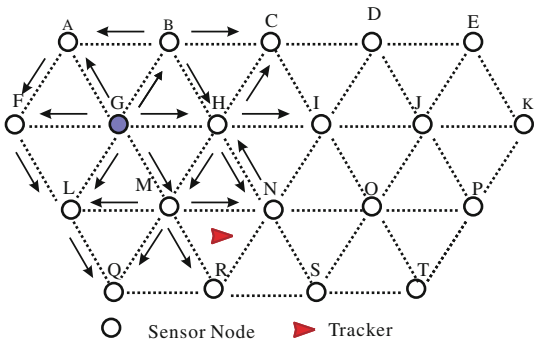


Fig. 4 An illustration of TTL-based broadcast: The source sensor node G (denoted in the *solid circle*) sent a message with TTL = 2 and the message will be forwarded at most 2 hops from node G

message. Its neighbor M receives the message, decreases the TTL by 1 and broadcasts it. Similarly, node N receives the message, decreases TTL, but will not forward the message since TTL = 0. In this example, the tracker is within the radio range of node M, so it successfully receives the message from G.

How can we choose the TTL value for a new message? Ideally, we want to set the TTL to the minimum hop count needed for messages to reach the tracker. Unfortunately, because the precise location of the tracker is unknown to the sensor nodes, this value is not available. Large TTL can guarantee the delivery of the message at the cost of high energy consumption, whereas small TTL requires less energy consumption, at the risk that messages may never reach the tracker. Small TTL values are especially harmful when the tracker starts its execution with no knowledge of the target's position, or when it loses track of the target.

To address the issue of choosing TTL, we propose to dynamically adjust the TTL value. In the steady state, the tracker should be within the vicinity of the target. Thus, we use a small TTL value by default. When the tracker is far from the target, we dynamically adjust the TTL so that the message will reach tracker without flooding the entire network. In order to adjust TTL, we need to know whether the tracker is in the proximity of the target. One observation is that when the tracker follows the target closely, the sensor nodes that sense the target are within the vicinity of the tracker as well. These nodes should receive the beacon signal sent by the tracker. When the tracker is far from the target, these nodes will only observe the target, not the tracker.

Based on these observations, we propose the TTL adjusting algorithm shown in Fig. 5, which is inspired by the TCP congestion control algorithm [57]. Each node executes `Adjust_TTL()` at small, fixed time intervals. If the node senses the target and the tracker, the `MIN_TTL` value is used. Otherwise, the node will double its TTL every time a timeout occurs, e.g., not hearing from the tracker for `TTL_TIMEOUT` number of time slices. We present experiments evaluating various choices for `TTL_TIMEOUT` in Sect. 6.

```
Algorithm:  Adjust_TTL
if trackerSeen() or not targetSeen() then
    messageTTL ← MIN_TTL
    timeoutCount ← 0
else
    timeoutCount ← timeoutCount +1
    if timeoutCount ≥ TTL_TIMEOUT then
        timeoutCount ← 0
        messageTTL ← 2 · messageTTL
    end
end
```

Fig. 5 The TTL adjustment algorithm

The overall effect of this algorithm is that, when a node detects the target for a period of time without also detecting the tracker, the TTL for messages from that node is gradually increased in an attempt to ensure that the tracker receives the messages. When the tracker arrives or the target departs, the TTL is reset.

5.1.1 Comparison to additive TTL increase

The multiplicative increase of the TTL values described in Fig. 5 is motivated by both the time needed for messages to reach the tracker and by the energy required to do so.

Consider an idealized scenario in which the target remains motionless and the tracker is h hops away from the target, with no obstacles along the way. Let ρ denote the average density of the network, measured in nodes per square meter. In this scenario, when a sensor near the target transmits a message with TTL equal to j, the area reached by that message is proportional to πj^2, so the total number of messages transmitted is proportional to $\pi j^2 \rho$.

As a result, a routing algorithm that additively increases the TTL by one at each timeout, the number of messages transmitted is $\sum_{i=1}^{h} \pi i^2 \rho \in O(h^3)$. In contrast, the multiplicative increase we propose would transmit only $\sum_{i=1}^{\lceil \lg h \rceil} \pi (2^i)^2 \rho \in O(h^2)$ messages. Moreover, notice that an algorithm using additive TTL increases would take roughly h timeouts to get a message to the tracker, in contrast to the $\lceil \lg h \rceil$ timeouts required using multiplicative increases.

5.1.2 Examples

Figure 6 illustrates the simulated operation of this algorithm in a rectangular environment with two large obstacles. Several snapshots of the execution are shown, starting from an initial condition in which the tracker and the target are near each other.

Figure 7 shows a slightly more complex situation, in which the tracker and the target are initially separated by a large distance. In this example, the tracker and the target are separated by approximately 60 hops in the network. As a result, no messages reach the tracker until at least one node has experienced 6 timeouts. Note that during this time, in the absence of additional information, the tracker moves toward the centroid of E.

5.2 Smart local routing for smarter sensors

We now consider a network of sensors capable of storing information and performing calculation. The key insight behind these new techniques is the notion of *Smart Local Routing*: We store partial information about the tracker and target positions at each sensor node. Each node determines

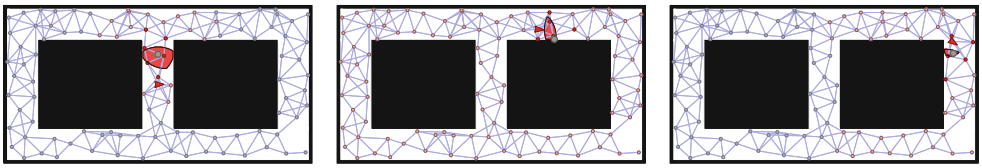


Fig. 6 A sample execution of the dynamic TTL algorithm inside a "cinder block" shaped environment. In all cases, the information state contains the location of the target and the tracker is able to keep close proximity to the target. The information state is *shaded*. The tracker is denoted by a *triangle* and the target is the *grey circle*

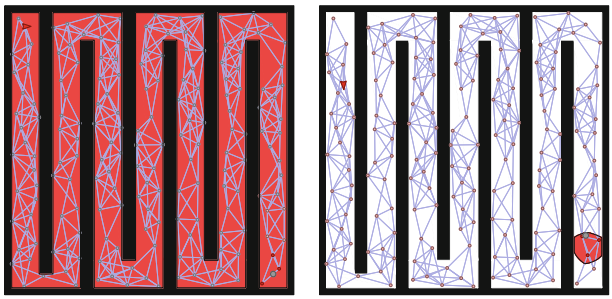


Fig. 7 A sample execution of the Smart Local Routing algorithm inside a "lens-shaped" environment. The information state is shaded *left* An initial condition with large uncertainty for the tracker *right*. The network provides information about the target location from far across the environment

to route messages only if they provide useful information to facilitate the target tracking.

We present the idea of Smart Local Routing with three parts: (1) how to compute the information state at each sensor; (2) when a sensor will initiate a message; and (3) what information shall be sent.

5.2.1 Calculating Node I-States

In this method, each node maintains an I-state similar to that used by the tracker. However, the computation power and memory available to small scale sensors will, in general, be much smaller than that available to a full-fledged mobile robot. Therefore, we reduce the computational load by using a provable overestimate of the precise I-state called the *disk-based I-state*. The disk-based I-state can be stored with a small constant amount of memory and updated in constant time.

In particular, maintaining I-states for the sensor nodes differs in two important ways from the method described in Sect. 4.1 for the tracker's I-state. First, we maintain only a simple approximation of the "true" nondeterministic I-state. Second, because the sensor nodes do not know the tracker's position, but could benefit from this information, we maintain position information about both the tracker and the target.

As time progresses, each sensor node n_i maintains a pair of disks $P_i(t)$ and $Q_i(t)$, with the invariant that

$$p(t) \in P_i(t) \text{and} q(t) \in Q_i(t). \tag{8}$$

These two disks constitute the node's *disk-based I-state*. As initial values, each node assigns both disk centers to its own position, and assigns infinite radii to both disks:

$$P_i(0) = B(n_i, \infty) \tag{9}$$

$$Q_i(0) = B(n_i, \infty) \tag{10}$$

From this initial I-state, four types of updates are required in order to maintain the invariant.

1. *Target detection*. When the node detects the target, it knows that the target is within distance r_s of itself. In this case, therefore, $Q_i(t)$ is replaced the smallest disk containing $Q_i(t) \cap B(n_i, r_s)$.
2. *Tracker detection*. When the node receives the tracker's beacon signal, it knows that the tracker is within distance r_c of itself. Similar to the previous case, $P_i(t)$ is replaced the smallest disk containing $P_i(t) \cap B(n_i, r_c)$.[2].
3. *Message receipt*. When the node receives a message from another node n_j, that message will describe the disk-based I-state of n_j. Specifically, this message contains the center coordinates and radius of both $Q_j(t)$ and $P_j(t)$. The node at n_i uses this information to refine its own knowledge, replacing $Q_i(t)$ with the smallest disk enclosing $Q_i(t) \cap Q_j(t)$ and replacing $P_i(t)$ with the smallest disk enclosing $P_i(t) \cap P_j(t)$. In this way, each node can refine its own knowledge based on messages it receives from neighboring nodes. This information sharing is the means by which knowledge is propagated across the network.
4. *Passage of time*. When time Δt passes without any of the previous three events occurring, then new disks are computed with the same centers, and having $\text{radius}(P(t + \Delta t)) = \text{radius}(P(t)) + \Delta t s_{\text{trk}}$ and radius $(Q(t + \Delta t)) = \text{radius}(Q(t)) + \Delta t s_{\text{tgt}}$. As with the tracker's I-state, this expansion of the disks corresponds to the unknown motions that tracker and target might have made during this time.

[2] Note that false positive sensing errors can be problematic for target detection and tracker detection updates, because they may make the I-state inaccurate or even empty. See Sect. 7.

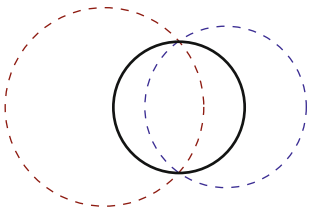


Fig. 8 Computing the smallest disk enclosing the intersection of two disks. The general case, in which the intersection in nonempty, but neither disk fully contains the other

These update operations depend on the ability to compute the smallest disk enclosing the intersection of two other disks. In general, given two disks D_1 and D_2, assume without loss of generality that D_1 is centered at the origin, that D_2 is centered at $(b, 0)$, and that the radii of the disks are r_1 and r_2, respectively. We must consider four cases:

- If $r_2 > r_1 + b$, then D_2 contains D_1. The intersection itself is D_1 itself: $D_1 \cap D_2 = D_1$.
- If $r_1 > r_2 + b$, then D_1 contains D_2, and the intersection itself is a D_2 itself: $D_1 \cap D_2 = D_2$.
- If $r_1 + r_2 < b$, then D_1 and D_2 are disjoint, and there is no solution. This case does not occur in our setting, since $D_1 \cap D_2$ must contain either $p(t)$ or $q(t)$.
- Otherwise, D_1 and D_2 intersect in a "lens-shaped" region, as shown in Fig. 8. Let $c = (r_1^2 + r_2^2 + b^2)/(2b)$. This region has vertical extrema at $(c, +\sqrt{r_1^2 - c^2})$ and $(c, -\sqrt{r_1^2 - c^2})$ and horizontal extrema at $(b - r_2,0)$ and $(r_1,0)$. Any disk enclosing D_1 and D_2 must contain all four of these points. This occurs with minimal radius when, choosing $a = \max(0, \min(b, c))$, the disk is centered at $(a,0)$ and has radius $\sqrt{(a-c)^2 + r_1^2 - c^2}$.

Note that this representation consumes $O(1)$ space, and that each of its updates can be performed in $O(1)$ time. The approach is, therefore, well-suited to simple sensor nodes with limited computation power.

5.2.2 *When to send a message?*

The nodes' strategy is based on the disk-based I-state that each node maintains. Whenever a node receives new information, either by detecting the tracker, detecting the target, or receiving a message from another node, it must decide whether to broadcast a message of its own to propagate this information across the network. The intuition is that a node will broadcast the message only if the message provides useful information to facilitate the target tracking. In particular, the node uses its I-state to identify a number of situations in which it should *not* broadcast such a message to its neighbors:

1. If the new information did not result in any change the node's disk-based I-state, then the node should not broadcast its knowledge. This improves efficiency by filtering redundant messages.
2. If the node received new information about the target's position, but has already broadcast a message triggered by target information in the recent past, it should not broadcast its knowledge. This rule facilitates data aggregation, preventing the node from generating frequent messages, of which each has only limited informative value. The definition of "recent past" is governed by a parameter τ, such that no node will generate two broadcasts triggered by tracker knowledge within time τ of each other.
3. Similarly, if the node received new information about the tracker's position, but has already broadcast a message triggered by tracker information in the last τ units of time, it should not broadcast its knowledge. This rate limiting is governed by the same parameter τ, but is controlled by separate timeout counter. These "dual timeouts" are crucial to facilitate free flow of information in both directions—from the tracker toward the target, and from the target toward the tracker.
4. Finally, if the node can conclude, based on its disk-based I-state, that it is *not near the shortest path between p(t) and q(t)*, then it should not broadcast its knowledge. This condition prevents the wasting of energy by transmitting data to remote portions of the environment that are not active parts of the tracking problem. Details about this condition appear below.

To implement the above constraint (4) in a precise, formal way, the node must compute the *geodesic hull* of its disk-based I-state. We define the geodesic hull of a pair of point sets A and B as the union of all shortest paths in the environment E from a point $a \in A$ to a point $b \in B$. The key insight is that if a node n_i is not near the geodesic hull formed by $A = P_i(t)$ and $B = Q_i(t)$, then n_i knows with certainty that information flowing between the tracker and target need not pass through n_i. The node n_i therefore decides not to broadcast its knowledge. See Fig. 9.

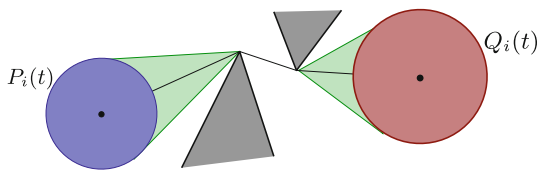


Fig. 9 The geodesic hull of two disks separated by triangular obstacles. If a node is far from the geodesic hull formed by the two disks in its disk-based I-state, it knows that information about the target can reach the tracker without it

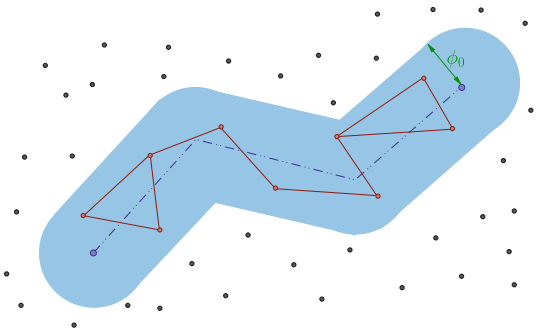


Fig. 10 The *dashed line* represents the shortest path between tracker and target. Nodes located inside the *shaded area* will broadcast messages since they are 'near' the shortest path

So far, the constraint has been expressed in terms of "nearness" to the geodesic hull. Formally, we define a node to be "near" the geodesic hull if its distance to the geodesic hull is smaller than a threshold ϕ_0. The criteria for determining ϕ_0 are to guarantee that the number of nodes near the geodesic hull is small, and those nodes should form a path that can deliver messages from the source to the intended destination, as shown in Fig. 10. Intuitively, if the network density is relatively high, then we expect to be able to use a relatively small threshold ϕ_0 for nearness in condition (4), knowing that other nodes closer to the shortest path are likely to receive the message and propagate the information. Likewise, a relatively sparse network suggests the need for a larger nearness threshold ϕ_0, as a hedge against the possibility of a large empty space in the network preventing information flow. Similarly, a larger r_c maps to a smaller ϕ_0, while a smaller r_c requires larger ϕ_0. In this paper, we select $\phi_0 = r_c$,[3] and we allow nodes to broadcast messages when they, according to their disk-based I-states, could possibly be within distance r_c of the shortest path between $p(t)$ and $q(t)$.

The result of this strategy is that every node broadcasts its knowledge whenever it is within or near the geodesic hull of its disk-based I-state. As a result, information collected by nodes that see the target will propagate to the tracker rapidly. Moreover, nodes that are not on or near the shortest path between target and tracker will, after receiving one of these messages, know that they need not broadcast any messages, and therefore remain silent. In this way, after an initial, one-time flooding, the disk-based I-states stored at each node enable the nodes to propagate messages only in a tight corridor around the relevant parts of the environment.

In practice, the true geodesic hull may be difficult to compute because the points $P_i(t)$ and $Q_i(t)$ may be connected by paths in many different homotopy classes. Therefore, our implementation approximates the geodesic hull by a point set consisting of all points in E that meet at least one of four criteria:

- Points in $P_i(t)$ or $Q_i(t)$.
- Points within distance $r_c - d$ of the shortest path from the center of $P_i(t)$ to the center $Q_i(t)$,
- Points inside the trapezoid formed by the four common tangent points of two particular circles. The first circle is $P_i(t)$. The second circle centered at the second vertex of the shortest path between the disk-based I-state centers and has radius $r_c - d$.
- Points inside the trapezoid formed *mutatis mutandis* from the penultimate vertex of the shortest path between centers and $Q_i(t)$.

This simplification takes advantage of the fact the information need only flow along *some* short path between the tracker and target, and not necessarily the definitive shortest path.

Note that this approach requires each node to have an accurate environment map. Each sensor node must also have sufficient computation power to compute shortest paths in this environment. If the environment is convex (or nearly so) a simpler constant-time approach that tests whether n_i is inside the convex hull of $P_i(t)$ and $Q_i(t)$ can be used instead.

5.2.3 What data to transmit

As mentioned previously, each time a node broadcasts a message, it transmits its entire current disk-based I-state. This design is motivated by a desire to maximize the informative value of each message. This approach stands in contrast to schemes that simply forward messages without modification. The approach proposed here, in particular, indirectly facilitates *data aggregation*, by allowing sensor nodes to accumulate information in their disk-based I-states for a short period of time (via the timeouts described above) before generating a single message encapsulating the combined information from each of the messages it received.

5.2.4 Examples

Figure 11 depicts the algorithm's simulated behavior in a relatively simple environment, illustrating the motions of both the tracker and the target.

In Fig. 12, we show several views of a single time step of the algorithm in a more complex environment. We observe that out of all the nodes that received the message at this time step (Fig. 12(b)), only a portion of them will broadcast the message (Fig. 12(c)), and we mark those "inner" nodes red (the darkest shade). The remaining

[3] We note that the energy consumption C measured in our experiments will be reduced if we selected a tighter bound for ϕ_0.

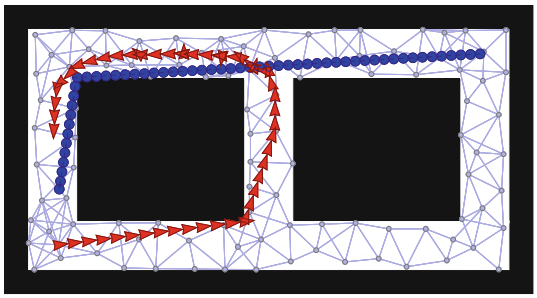


Fig. 11 Time-lapse view of the Smart Local Routing algorithm in a simple environment. The tracker (*triangle*) starts in the lower left corner and the target (*circle*) starts in the upper right corner. This example shows that once the tracker managed to get close to the target, it maintains its proximity to the target

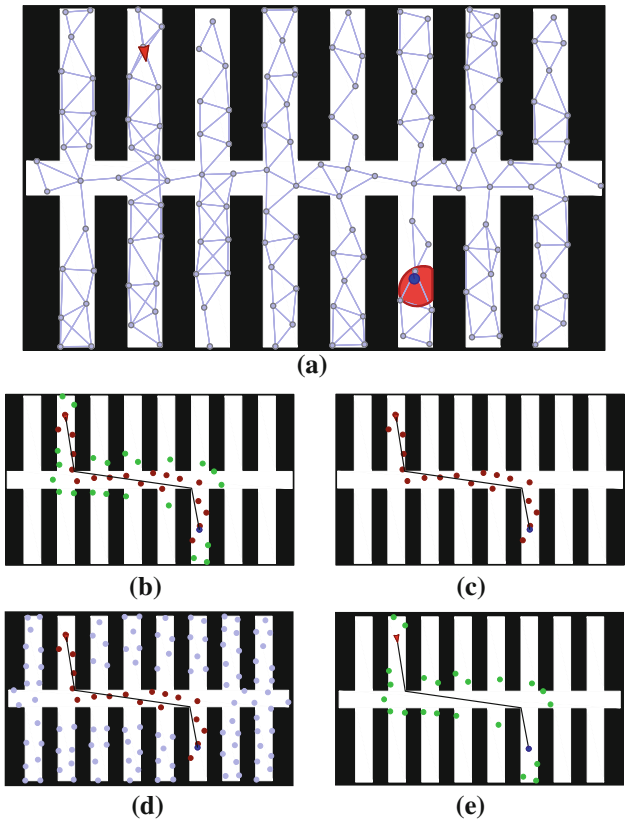


Fig. 12 Execution snapshots the Smart Local Routing algorithm showing the behavior of the network nodes at one time step. **a** The complete network. The tracker's I-state is shaded. **b** Nodes that received a message at this time step. **c** Nodes that sent a message at this time step. **d** Nodes that are near the simplified geodesic hulls of their disk-based I-states. **e** Nodes that are *not* near their simplified geodesic hulls. These nodes know that they are not along the path from tracker to target, and therefore do not spend any energy sending messages

nodes (Fig. 12(e)) serve as "buffer", and decide not to propagate the information any further since they know that they are not sufficiently close to the shortest path. Note especially Fig. 12(d), which shows that a large fraction of the nodes cannot rule out being near the shortest path between tracker and target. For some "inner" nodes (denoted by red or the darkest shade), this is because they are in fact near the shortest path. For other "outer" nodes (denoted by blue or light shade), this is because they are far from the shortest path, and they haven't received any data. As a result, their disk-based I-states have grown into large regions, making them unable to rule out the possibility of being near the shortest path. However, this limitation will not affect the energy efficiency of our algorithm for two reasons. First, a collection of "buffer" nodes (Fig. 12(e) denoted by green or intermediate shade) around the shortest path which do not forward the message acts as "buffer zone." Second, even if such a message did reach a node with very large radii in its disk-based I-state, that node would immediately update its disk-based I-state, and realize that it is far away from the shortest path. This behavior is typical for our algorithm, and shows how our Smart Local Routing prevents messages from spreading to irrelevant parts of the environment.

5.3 Comparison between algorithms

We now compare our two algorithms in terms of "steady state" phase and "startup" phase. In the steady state phase, the tracker and the target are near one another. However, in the startup phase, the tracker and the target are separated by many hops in the network.

5.3.1 Steady state phase

The Dynamic TTL algorithm utilizes a timeout-based approach to limit the flooding region. Its biggest advantage is its simplicity and it works especially well in steady state scenarios: if the target and tracker are near one another, sending messages for a few hops (controlled by a small TTL value) is enough to maintain this proximity. In comparison, although the Smart Local Routing algorithm also works well in this state, it requires each node to maintain the disk-based I-state locally, thus, imposing higher requirements on the sensors' hardware and the software.

5.3.2 Startup phase

The drawback of the Dynamic TTL algorithms is that its performance in the "startup phase" suffers from the following issues.

- *Prolonged delay*. By default, a small TTL value is used to limit the energy spent on message transmission. The TTL value doubles every time a timeout occurs. In the startup phase, it may take many timeouts before TTL grows large enough for a message to reach the tracker,

causing long delay. If the target moves too rapidly, these timeouts may never occur because no single node will be able to observe the target long enough.

- *Unnecessary message delivery*. After information about the target's position has reached the tracker, the network will continue to flood updates with very high TTL values across large portions of the sensor network. These messages are unnecessary because, when the tracker is still far away from the target, slightly updated information about the target's position effects only small changes changes to the tracker's motion.
- *Irrelevant message delivery*. The Dynamic TTL approach wastes significant quantities of energy transmitting messages in portions of the environment that are not helpful for communicating target information to the tracker, as depicted in Fig. 12(a). In that example, only the branches containing the target and the tracker, along with the portion of the center corridor that connects those two branches, are relevant for communicating information about the target to the tracker.

The Smart Local Routing algorithm overcomes these limitations. By storing partial information about the tracker and target positions at each sensor node, each node can decide to route messages only if they provide useful information to facilitate the target tracking.

In summary, the Dynamic TTL algorithm is simple to implement and provides nice tracking performance in a steady state, whereas the smart local algorithm reaches this steady state faster, and works well in complex, highly nonconvex environments.

6 Implementation and evaluation

We have implemented both the dynamic TTL approach and Smart Local Routing in a customized simulator that is programmed in C++. In this section, we first present a quantitative evaluation of their performance in comparison to each other in two representative environments. Then, we compare their performance with other naive routing protocols and studied the impact of the node density on them. In all experiments, we selected $r_s = 2m$ and $r_c = 2m$.

6.1 Performance comparison in two environments

One main factor that affects the performance of the Dynamic TTL approach and Smart Local Routing is the average hop counts between any pair of network nodes. Thus, we examined both approaches in two representative environments with distinguishingly different distributions of hop counts: an open environment and a maze-like environment, as shown in Fig. 13. The open environment

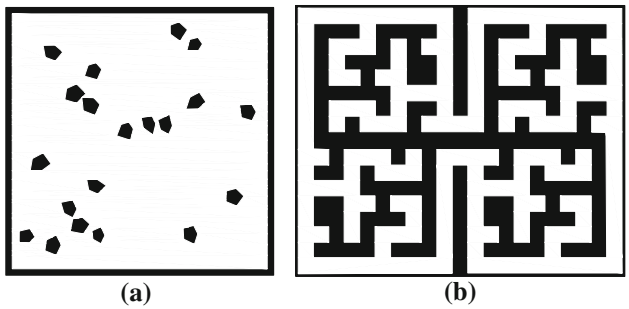


Fig. 13 Experiment environments: **a** An open environment (20 m-by-20 m) used in the experiment of Sect. 6.1.1. The obstacles are generated randomly. **b** A maze-like environment (21 m-by-17 m), used in the experiment of Sect. 6.1.2

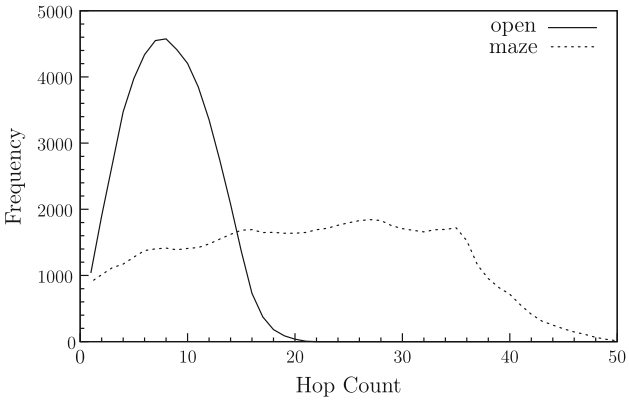


Fig. 14 Histogram of hop counts between all pairs of nodes in two representative environments, which are depicted in Fig. 13. Unsurprisingly, the majority of hop counts in the maze-like environment is larger than the ones in the open environment, which makes the startup phase longer

(Fig. 13(a)) is populated by a collection of randomly-placed obstacles in an open square, where the majority of the hop counts are about 9 hops, as shown in Fig. 14. In comparison, the maze-like environment (Fig. 13(b)) is a larger, more complex environment with several branches, where the hop counts varies from 1 hops up to 50 hops, as shown in Fig. 14.

6.1.1 *In an open environment*

Using the environment shown in Fig. 13(a), we performed a series of 10 trials in which the starting positions of the tracker and target and the placements of the sensor nodes were selected randomly. For each trial, we simulated the startup performance of both tracking algorithms, with 5 different parameter settings each:

- five using the Dynamic TTL approach, with `TTL_TIMEOUT` set to 5,8,11,14, and 17, and
- five using the Smart Local Routing approach, with the minimum time interval between two successive broadcasts τ set to 3,6,9,12 and 15.

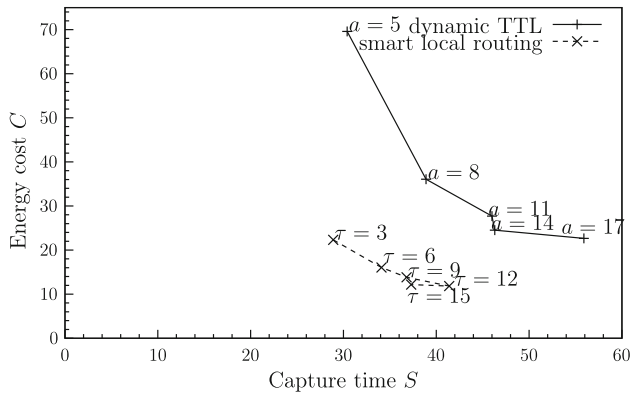


Fig. 15 Start-up performance: Energy cost C versus capture time S for the environment depicted in Fig. 13(a). Smart Local Routing requires less amount of time to start up than the Dynamic TTL algorithm at smaller energy costs. Note a stands for `TTL_TIMEOUT`

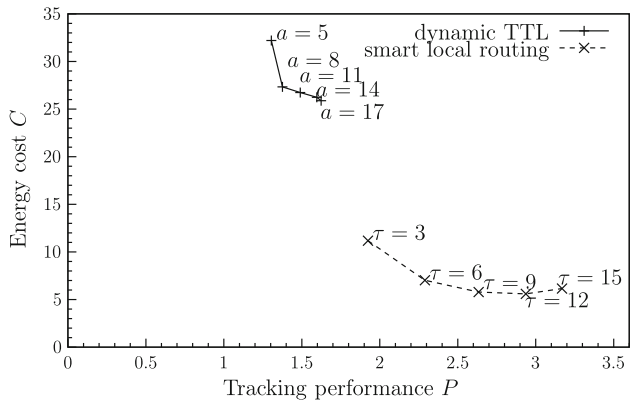


Fig. 16 Steady-state performance for the environment in Fig. 13(a), showing Dynamic TTL method maintained slightly better tracking performance at the cost of higher energy consumption. Note a stands for `TTL_TIMEOUT`

The performance of each algorithm, averaged over all trials, is shown in Fig. 15. Because the start-up phase of our tracking problem has two optimality criteria, each dimension of the plot measures one of the optimality criteria. Note that the origin of the plot represents the (unachievable) ideal of perfect tracking with no energy cost. Each data point, therefore, dominates—in the sense of faring better in both performance criteria—any other data points that are both above it and to its right.

These results confirm our prediction that the Smart Local Routing is superior to the Dynamic TTL algorithm in the startup phase, regardless of the parameter value τ in the range that we tested.

To evaluate the performance of our algorithms in the steady-state phase of tracking, we performed ten additional trials in which the tracker and target are started near one another. We used $T = 1000$, and averaged the results over all ten trials. Figure 16 depicts the results of this experiment, showing that the Dynamic TTL method maintained slightly better tracking performance at the cost of higher energy consumption.

Note that the energy cost of Smart Local Routing algorithm does not necessary decrease when τ increases, while the tracking performance decreases most of the time. We note the point of $\tau = 15$ in Fig. 15 is counterintuitive. Our hypothesis is that a larger τ forces the sensor to send messages at larger intervals, and gives the network less freedom to determine the best time to send messages. Such freedom is beneficial, since it helps to improve the tracking performance most of the time.

6.1.2 *In a maze-like environment*

Using the environment shown in Fig. 13(b), we performed 10 trials with randomly selected initial conditions, with the same 10 tracking schemes as above, for both steady-state and startup scenarios. The results appear in Figs. 17 and 18.

As with the simple environment, these results demonstrate that the Smart Local Routing approach dominates the Dynamic TTL approach in the startup phase, and this superiority is independent of the parameter value τ in the range that we tested. Compared with the results in the simple environment, the improvement of the Smart Local Routing approach over the Dynamic TTL approach is more pronounced. This can be explained by the following: the tracker and the target are separated by a larger number of hops in the maze-like environment; thus, it takes larger number of timeouts before TTL grows large enough to deliver the message, resulting in large capture time. The Smart Local Routing algorithm is not affected, because message delivery to the tracker begins immediately.

Similar to the result in the simple environment, in the steady-state phase of tracking, the Dynamic TTL approach

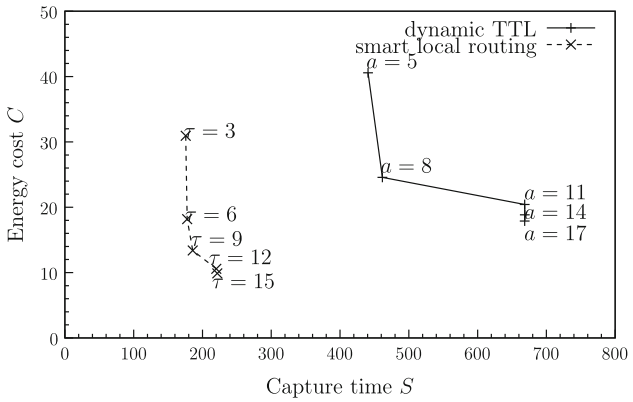


Fig. 17 Start-up performance: Energy cost C versus capture time S for the environment depicted in Fig. 13(b), showing that the Smart Local Routing is superior to the Dynamic TTL algorithm in the startup phase. Note a stands for `TTL_TIMEOUT`

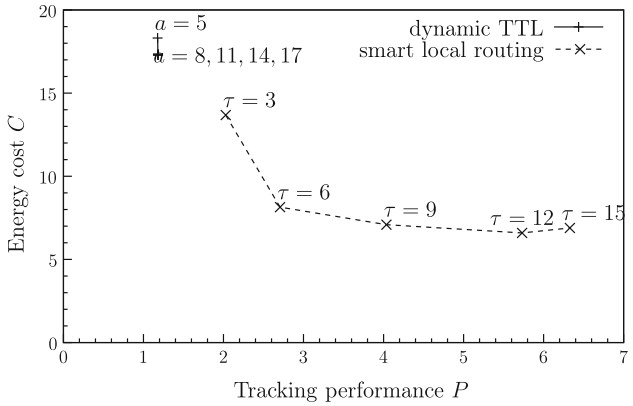


Fig. 18 Steady-state performance for the environment in Fig. 13(b), showing the Dynamic TTL approach can achieve better tracking performance at the cost of extra energy consumption. Note a stands for `TTL_TIMEOUT`

can achieve better tracking performance at the cost of extra energy consumption.

6.1.3 Worst case performance

In both environments we measured the worst case performance P_{max} in the same set of simulation runs. Table 1 summarizes P_{max} for both approaches. For the Dynamic TTL approach, a smaller `TTL_TIMEOUT` results in a smaller P_{max}. Similarly, for Smart Local Routing approach, adopting a smaller τ (the minimum time interval between two successive broadcasts) creates a smaller P_{max}. Even though the P_{max} was several times greater than P for large `TTL_TIMEOUT` or τ, both algorithms always managed to recover from it quickly and to maintain a low P.

Table 1 The maximum distance between the tracker and the target during the steady-state phase. $P_{max} = \max_{t\in[0,T]} ||p(t) - q(t)||$. Both algorithms always managed to recover from it quickly and to maintain a low P

	Dynamic TTL		Smart local routing	
	a	P_{max}	τ	P_{max}
Maze	5	2.06	3	4.17
	8	2.34	6	8.05
	11	3.03	9	14.26
	14	3.03	12	15.94
	17	3.03	15	16.25
Open	5	4.57	3	3.90
	8	6.20	6	4.51
	11	10.31	9	5.14
	14	11.17	12	6.26
	17	11.17	15	6.65

6.2 Impact of node density

To study the impact of node density on our approaches, we evaluated the steady-state performance and the normalized energy costs while varying the node density in the environment depicted in Fig. 11. When the total number of nodes is less than 75, the network is usually not connected, and thus the performance of our algorithms suffers. Once the network is fully connected (more than approximately 75 nodes), the performance becomes promising, as shown in Fig. 19(b). Figure 19(a) shows the average energy cost for each node, e.g., the average number of messages transmitted by each node in a unit time. We observe that the Smart Local Routing scales gracefully as the network density increases. That is, the normalized energy cost for the Smart Local Routing remains constant, while the energy cost for the Dynamic TTL approach scales linearly with the total number of nodes. This is because the Smart Local Routing algorithm aggregates its data and suppresses the redundant messages well. Another nice feature of the Smart Local Routing approach is that it can achieve good performance as long as the network is connected, and in

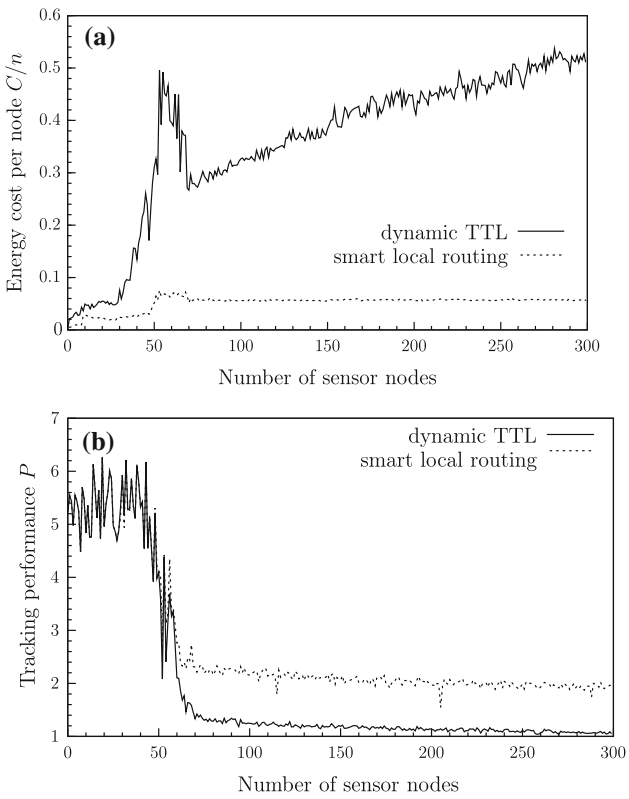


Fig. 19 The impact of node density on the steady-state performance and energy costs for the environment in Fig. 11. We observe that the Smart Local Routing scales gracefully as the network density increases, while the energy cost for the Dynamic TTL approach scales linearly with the total number of nodes. a stands for `TTL_TIMEOUT`. We used `TTL_TIMEOUT` =3 and $\tau = 5$

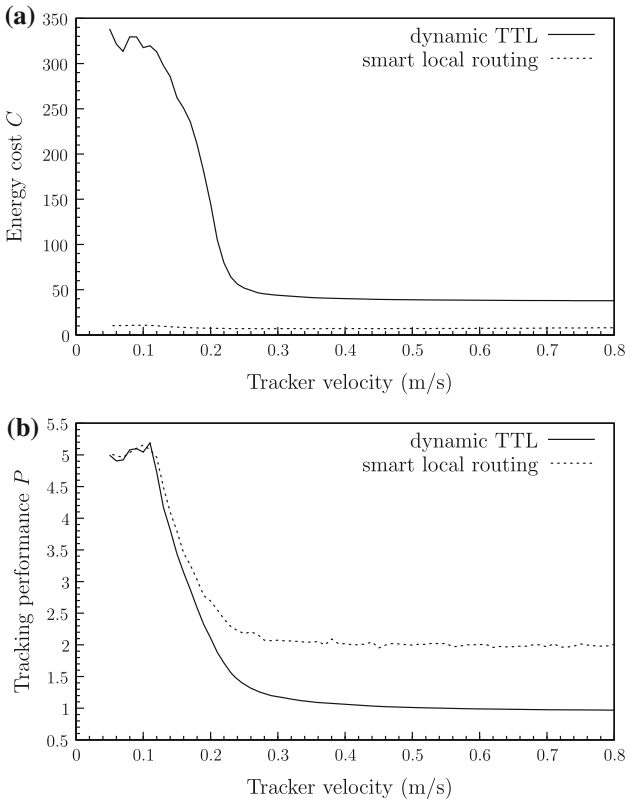


Fig. 20 The impact of tracker speed on the steady-state performance and energy costs for the environment in Fig. 11. We used `TTL_TIMEOUT` = 3 and $\tau = 5$

fact its performance does not benefit significantly from an over-dense network.

6.3 Impact of speed differences

We also performed an experiment to determine the effect of speed differences between the tracker and target on our algorithms' performance. Using the environment depicted in Fig. 11, we held the target speed constant at $s_{\text{tgt}} = 0.2$ *m/s* and varied the tracker speed s_{trk} between 0.01 *m/s* and 0.8 *m/s*, increments of 0.01 *m/s*. For each tracker speed, we performed 10 trials for each algorithm, and measured both the energy cost and steady-state tracking performance. The results appear in Fig. 20. As predicted, when the tracker travels slower than the target, it cannot keep up with the target. When the tracker travels faster than the target, both algorithms suffice to assist the tracker to keep close proximity with the target.

6.4 Comparison to existing algorithms

Finally, to ensure that our approach compares favorably to known methods, we performed a final experiment testing its performance against two basic protocols:

Table 2 Comparison to existing message delivery schemes

	Startup		Steady state	
	S	*C*	*P*	*C*
Flooding	28.1	1,064.0	1.2	1,232.3
Static TTL	59.5	21.7	1.8	25.6
Dynamic TTL	30.4	69.6	1.3	32.2
Smart local routing	36.8	13.8	2.6	5.8

Static TTL used a TTL value of 1. Dynamic TTL used a `TTL_TIMEOUT` of 5. Smart local routing used $\tau = 9$. The results demonstrate that both Dynamic TTL approach and smart routing protocol achieve something close to "the best of both worlds" in balancing tracking precision and energy efficiency

1. *Flooding*, in which every message is forwarded to every node in the network. This approach delivers every message to the tracker and generates very accurate tracking but also very large energy consumption.
2. *Static TTL*, in which every message is broadcast with a fixed TTL of 1. This approach is very energy efficient, but leads to poor tracking performance because messages are forwarded only locally.

We selected these two baseline algorithms because the represent two extremes in the tradeoff between tracking performance and energy efficiency. This experiment used the environment shown in Fig. 13(a). The final results, which are averaged across ten trials for startup performance and ten trials for steady-state performance, appear in Table 2. The results demonstrate that both Dynamic TTL approach and smart routing protocol achieve something close to "the best of both worlds" in balancing tracking precision and energy efficiency.

7 Overcoming sensing errors

A malfunction sensor node can either produce false negative errors or false positive errors. False negative errors occur when a node fails to detect the tracker (or target) when the tracker (or target) is indeed within its sensing range. False positive errors happen when a node mistakenly claims to detect the tracker (or target) when the tracker (or target) is *not* present within its sensing range.

So far our algorithm are robust against false negative errors, but the preceding discussion assumed that the sensor nodes do not experience false positive errors. In this section, we describe modifications to our basic approach that allow a relaxation of this assumption, i.e. false positive errors are possible.

Dealing with false positive errors is challenging for at least two reasons. First, we do not have any predictive

model for the target's motion. As a result, traditional probabilistic inference methods such as Hidden Markov Models do not apply, because the transition matrix is unknown. Second, because of the limitations of our energy budget, filtering decisions must be made locally, and cannot benefit from a globally consistent view.

To overcome these challenges, two basic changes are needed. First, we must ensure that both the tracker and the sensor nodes have nonempty I-states at all times. To accomplish this, we modify the I-state update process to *discard any observation or message that would lead to an empty I-state*. Note that this change implies that, depending on the order in which messages arrive, the information state may not necessarily contain the true state. Second, we require a means to *detect and discard false positive errors at their source*, to prevent those errors from degrading the quality of the I-states used throughout the system. The remainder of this section is concerned with filtering algorithms that implement this second change. Without loss of generality, we describe these modifications in terms of target detections. In practice, each node maintains two parallel copies of the filter: one for the target and one for the tracker.

We consider two distinct sources of false positive errors: Short-term errors resulting from sensor noise, and persistent errors resulting from hardware failures or misclassification of environment features. For the former case, we use a temporal filtering algorithm in which each node maintains a short-term history of its own observations (Sect. 7.1). For the latter case, we use a spatial filtering approach, in which each node compares its observation to those of its immediate neighbors (Sect. 7.2)

7.1 Temporal filtering

For false positive errors attributable to sensor noise, we use a *temporal filtering* scheme. Each node remembers a short-term history of its own observations, and uses this information in an attempt to detect and eliminate the false positives.

We choose a time interval Δt, representing the length of history considered by the filter. For a fixed time t, let $y_1, \ldots, y_k$ denote the sensor readings obtained in the preceding Δt units of time, so that $y_i = 1$ if the sensor detected the target at time $t - i$ and $y_i = 0$ otherwise. Let R denote the disk in the plane within which the sensor can detect the target.

We assume that the sensors exhibit errors according to known a false positive rate

$$P_{FP} = P(y_i = 1 \mid q(t_i) \notin R) \tag{11}$$

and a known false negative rate

$$P_{FN} = P(y_i = 0 \mid q(t_i) \in R), \tag{12}$$

and that each sensor reading is independent of the others. Moreover, we make the simplifying assumption that the target neither enters nor leaves the region sensed by the node between time $t - \Delta t$ and t. Although, strictly speaking, the target's motion may violate this assumption, it can be a reasonable simplification when Δt is small compared to s_{tgt}.

Under these assumptions, we can construct a straightforward Bayesian filter to determine the probability that the target is within the sensor's range, given the node's recent history of observations. Using Bayes' rule and assuming uniform priors, we can express the probability that the target is within the sensor's range in terms of P_{FP} and P_{FN} as

$$\begin{aligned} &P(q(t_i) \in R \mid y_1, y_2, \ldots, y_k) \\ &\quad = \frac{P(y_1 \ldots y_k | q(t_i) \in R) P(q(t_i) \in R)}{P(y_1, \ldots, y_k)} \\ &\quad = \frac{\prod_{i=1,\ldots,k} P(y_i | q(t_i) \in R)}{2P(y_1, \ldots, y_k)} \\ &\quad = \frac{P_{FN}^{\#\{y_i=0\}} (1 - P_{FP})^{\#\{y_i=1\}}}{2P(y_1, \ldots, y_k)} \\ &\quad = \alpha P_{FN}^{\#\{y_i=0\}} (1 - P_{FP})^{\#\{y_i=1\}}, \end{aligned}$$

in which $\alpha = 1/(2P(y_1, \ldots, y_k))$ is a scaling factor, and the exponents are the numbers of observations in the filter's history with the corresponding value for y_i. Similarly, we can express the probability that the target is *not* within the sensor's range as

$$\begin{aligned} &P(q(t_i) \notin R \mid y_1, y_2, \ldots, y_k) \\ &\quad = \alpha (1 - P_{FP})^{\#\{y_i=0\}} P_{FP}^{\#\{y_i=1\}}. \end{aligned}$$

Taken in combination, these two equations enable us to determine the value of α for which the total probability sums to 1. A more detailed exposition of this kind of Bayesian filtering appears in [58].

Finally, the sensor treats a positive sensor reading as accurate if $P(q(t_i) \in R | y_1, y_2, \ldots, y_k)$ exceeds some threshold $\beta \in (0, 1)$. Although 0.5, which is equivalent to a direct comparison of the two probabilities derived above, might appear to be a natural choice for β, in fact this parameter encodes a tradeoff. If β is large, the filtering is more strict, ensuring that fewer false positives are incorrectly accepted. If β is smaller, the filtering is more relaxed, reducing the number of true positives incorrectly rejected.

We tested the effectiveness of this filter for varying values of k and β with in the environment of Fig. 6. As a representative example of our tracking algorithms, we used the Dynamic TTL method with `TTL_TIMEOUT` = 5. For

Table 3 Performance results for temporal filtering. The results show that the filter makes a substantial improvement in all three performance criteria

	Startup		Steady state	
	S	*C*	*P*	*C*
$\beta = 0.5$				
No filtering	383.8	74.2	6.9	73.4
$k = 3$	63.5	28.1	1.6	22.6
$k = 5$	59.7	24.5	1.7	21.6
$k = 7$	53.8	21.1	1.9	20.9
$k = 9$	61.7	18.0	2.0	20.1
No errors	40.4	60.2	1.3	26.3
$\beta = 0.95$				
No filtering	383.8	74.2	6.9	73.4
$k = 3$	52.9	17.9	1.7	15.9
$k = 5$	51.4	18.8	1.7	17.8
$k = 7$	56.0	16.6	1.8	18.0
$k = 9$	54.1	14.7	2.0	17.4
No errors	40.4	60.2	1.3	26.3

these simulation runs, we used $P_{FP} = P_{FN} = 0.05$. We conducted ten trials for each run. Table 3 shows the results.

For comparison, we also show the performance of our filterless algorithm running the same trials in the absence of sensor errors. Notice that, according to the evaluation criteria set forth in Sect. 3.2, filter makes a substantial improvement in all three performance criteria, in a way that is not substantially sensitive to our choice of parameters k and β.

7.2 Spatial filtering

Another possibility is that the sensor nodes may experience errors that persist over longer periods of time. Such errors may arise, for example, from temporary or permanent hardware failures. In this case, the temporal filter described above is clearly not helpful, because the erroneous sensor readings will repeat, potentially for much longer than the time interval considered by the filter.

For this case, we instead use a spatial filtering approach. Each node compares its own sensor reading to those of its neighbors. If few or none of those neighboring nodes have detected the target, the sensor node rejects its own sensor reading as erroneous. Specifically, we chose a tunable parameter $\gamma \in (0, 1)$ and reject the sensor reading if the fraction of the node's immediate neighbors that also detect the tracker is less than γ. This approach is based on the idea that, as long as the network is sufficiently dense, the sensing regions of neighboring nodes will have some overlap. Therefore, if the target is truly within sensing range, we can expect one or more of the neighboring nodes to be able to corroborate this fact. This method introduces some additional energy cost, because the local voting process requires each node to transmit a beacon message each time it detects the target. However, our experiments suggest that this extra energy cost is effectively offset by the benefits of rejecting the false positive errors.

Table 4 Performance results for spatial filtering, which shows that the filter is effective at mitigating performance drops resulting from false positive sensor errors, especially when γ is small

	Startup		Steady state	
	S	*C*	*P*	*C*
No filtering	125.6	177.9	4.2	253.5
$\gamma = 0.1$	55.8	51.5	1.8	41.1
$\gamma = 0.2$	58.4	28.8	2.4	29.8
$\gamma = 0.3$	67.7	20.4	4.2	23.2
$\gamma = 0.4$	94.7	16.5	5.1	20.0
$\gamma = 0.5$	96.1	15.4	5.2	18.9
No errors	40.4	60.2	1.3	26.3

To determine the effectiveness of the filter, we performed a series of tests, again using the straightforward environment shown in Fig. 6. For each node, we simulated persistent errors that last between 25 and 50 time steps. On average, each node was in a failure state approximately 8% of the time. Several values for γ were tested, and the results were averaged across 10 trials. The network used the Dynamic TTL protocol with `TTL_TIMEOUT` = 5.

Table 4 shows the results of this simulation. The results show that the filter is effective at mitigating performance drops resulting from false positive sensor errors, especially when γ is small. These results are consistent with the intuition that (unless false positive errors are extremely common), if both a node and one or two of its neighbors independently detect the target, those mutually-reinforcing readings are likely to be correct. More generally, it is likely that γ will require tuning based on the density, communication range, sensing range, and error rates of an individual system.

8 Conclusion

We presented a target tracking algorithm that uses a collaboration between a sensorless robot and a network of unreliable sensor nodes. Simulations demonstrate that this algorithm has good performance in balancing energy efficiency with tracking accuracy, even in the presence of false positive sensor errors. However, a number of interesting questions remain unanswered.

One important avenue for future research is to consider how our system can be used to allow multiple robots to

track multiple targets. In scenarios in which the sensors can distinguish targets from one another, the individual tracking problems are orthogonal and we can run several instances of our algorithms in parallel. The more general case, in which the targets are not distinguishable, presents difficult challenges in data association and task assignment [59].

Another important avenue for further research is to design filtering algorithms that are effective at identifying sensor errors that are both persistent in time and widespread in a local region of space. Such errors are challenging because, from an individual node's local perspective, they are indistinguishable from true positive sensor readings. We plan to explore means by which the nodes can resolve these errors by constructing a global viewpoint as needed, without sacrificing the overall energy efficiency of the system.

Acknowledgments O'Kane is supported by NSF (CAREER award IIS-0953503) and DARPA (N10AP20015). Xu is supported by NSF (CAREER award 0845671).

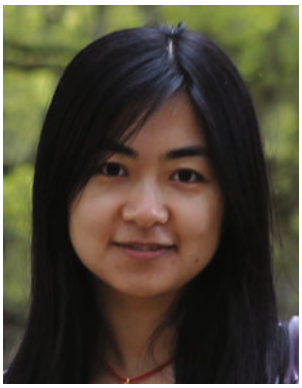
References

1. Bandyopadhyay, T., Li, Y. P., Ang, M. H., & Hsu, D. (2004). Stealth tracking of an unpredictable target among obstacles. In *Proceedings of workshop on the algorithmic foundations of robotics* (pp. 43–58).
2. LaValle, S. M., González-Baños, H. H., Becker, C., & Latombe, J.-C. (1997). Motion strategies for maintaining visibility of a moving target. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 731–736).
3. Murrieta, R., Sarmiento, A., Bhattacharya, S., & Hutchinson, S. A. (2004). Maintaining visibility of a moving target at a fixed distance: The case of observer bounded speed. In *Proceedings of the IEEE international conference on robotics and automation*.
4. Murrieta-Cid, R., Gonzalez-Banos, H. H., & Tovar, B. (2002). A reactive motion planner to maintain visibility of unpredictable targets. In *Proceedings of the IEEE international conference on robotics and automation*.
5. Murrieta-Cid, R., Tovar, B., & Hutchinson, S. (2005). A sampling-based motion planning approach to maintain visibility of unpredictable targets. *Autonomous Robots, 19*(3), 285–300.
6. Guibas, L. J., Latombe, J.-C., LaValle, S. M., Lin, D., & Motwani, R. (1999). Visibility-based pursuit-evasion in a polygonal environment. *International Journal on Computational Geometry and Applications, 9*(5), 471–494.
7. Jung, B., & Sukhatme, G. S. (2006). Cooperative multi-robot target tracking. In *Proceedings of the international symposium on distributed autonomous robotic systems* (pp. 81–90). Minneapolis, Minnesota.
8. Balluff. Inductive proximity sensors. http://www.balluff.com/.
9. Fargo Controls. Photoelectric proximity sensors. http://www.fargocontrols.com/sensors/photoelectric.html.
10. LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press, Cambridge. Available at http://planning.cs.uiuc.edu/.
11. O'Kane, J. M., & Xu, W. (2009). Energy-efficient target tracking with a sensorless robot and a network of unreliable one-bit proximity sensors. In *Proceedings of the IEEE international conference on robotics and automation*.
12. O'Kane, J. M., & Xu, W. (2010). Network-assisted target tracking via smart local routing. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems*.
13. O'Kane, J. M. (2008). On the value of ignorance: Balancing tracking and privacy using a two-bit sensor. In *Proceedings of the workshop on the algorithmic foundations of robotics*.
14. Guilamo, L., Tovar, B., & LaValle, S. M. (2004). Pursuit-evasion in an unkown environment using gap navigation trees. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems*.
15. Isler, V., Sun, D., & Sastry, S. (2005). Roadmap based pursuit-evasion and collision avoidance. In *Proceedings of the robotics: Science and systems*.
16. Vidal, R., Shakernia, O., Kim, H. J., Shim, D. H., & Sastry, S. (2002). Probabilistic pursuit-evasion games: Theory, implementation, and experimental evaluation. *IEEE Transactions on Robotics and Automation*.
17. Vieira, M. A., Govindan, R., & Sukhatme, G. S. (2009). Scalable and practical pursuit-evasion with networked robots. *Intelligent Service Robotics, 2*(4), 247–263.
18. Chen, P., Oh, S., Manzo, M., Sinopoli, B., Sharp, C., & Whitehouse, K., et al. (2006). Instrumenting wireless sensor networks for real-time surveillance. In *Proceedings of IEEE international conference on robotics and automation* (pp. 3128–3133).
19. Simon, G., Maróti, M., Lédeczi, Á., Balogh, G., Kusy, B., & Nádas, A., et al. (2004). Sensor network-based countersniper system. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems* (pp. 1–12).
20. He, T., Krishnamurthy, S., Stankovic, J., Abdelzaher, T., Luo, L., & Stoleru, R., et al. (2004). Jonathan Hui, and Bruce Krogh. Energy-efficient surveillance system using wireless sensor networks. In *MobiSys '04: Proceedings of the 2nd international conference on mobile systems, applications, and services* (pp. 270–283).
21. Zhao, F., Shin, J., & Reich, J. (2002). Information-driven dynamic sensor collaboration. *IEEE Signal Processing Magazine*, 19(2):61–72.
22. Aslam, J., Butler, Z., Constantin, F., Crespi, V., Cybenko, G., & Rus D. (2003). Tracking a moving object with a binary sensor network. In *SenSys '03: Proceedings of the 1st international conference on embedded networked sensor systems* (pp 150–161). ACM: New York, NY
23. Gui, C. & Mohapatra, P. (2004). Power conservation and quality of surveillance in target tracking sensor networks. In *MobiCom '04: Proceedings of the 10th annual international conference on mobile computing and networking* (pp. 129–143).
24. Shrivastava, N., Mudumbai, R., Madhow, U., Suri, S. (2006). Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms. In *Proceedings of the International conference on embedded networked sensor systems* (pp. 251–264).
25. Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L., Rubenstein D. (2002). Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, (pp. 96–107).
26. Chu, C. P., Tsai, H. W., Chen, T. -S. (2007). Mobile object tracking in wireless sensor networks. *Computer Communications 30*, 1811–1825.
27. Olfati-Saber, R. (2007). Distributed tracking for mobile sensor networks with information-driven mobility. In *American control conference* (pp. 4606–4612).
28. Zou Y., & Chakrabarty K. (2007) Distributed mobility management for target tracking in mobile sensor networks. *IEEE Transactions on Mobile Computing, 6*(8), 872–887.

29. Baryshnikov, Y., & Ghrist R. (2008). Target enumeration via integration over planar sensor networks. In *Proceedings of robotics: Science and systems*.
30. Singh, J., Madhow, U., Kumar, R., Suri, S., & Cagley, R. (2007). Tracking multiple targets using binary proximity sensors. In *Proceedings of the information processing in sensor networks* (pp. 529–538).
31. Shrivastava, N., Mudumbai, R., Madhow, U., & Suri, S. (2006). Target tracking with binary proximity sensors: Fundamental limits, minimal descriptions, and algorithms. In *Proceedings of the international conference on embedded networked sensor systems* (pp. 251–264).
32. Woo, A., Tong, T., & Culler, D. (2003). Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on embedded networked sensor systems* (pp. 14–27).
33. Kusy, B., Lee, H. J., Wicke, M., Milosavljevic, N., & Guibas, L. (2009). Predictive qos routing to mobile sinks in wireless sensor networks. In *Proceedings of the 2009 international conference on information processing in sensor networks, IPSN '09* (pp. 109–120). Washington, DC. IEEE Computer Society.
34. Luo, H., Ye, F., Cheng, J., Lu, S, & Zhang, L. (2003). Ttdd: A two-tier data dissemination model for large-scale wireless sensor networks. In *Proceedings of international conference on mobile computing and networking (MobiCom*.
35. Intanagonwiwat, C., Govindan, R., & Estrin, D. (2000). Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the international conference on mobile computing and networking* (pp. 56–67).
36. Sarkar, R., Zhu, X., & Gao, J. (2006). Double rulings for information brokerage in sensor networks. In *Proceedings of the conference on mobile computing and networking, MobiCom '06* (pp. 286–297).
37. Ratnasamy, S., Karp, B., Yin, L., Yu, F., Estrin, D., Govindan, R., et al. (2002). GHT: a geographic hash table for data-centric storage. In *Proceedings of the workshop on Wireless sensor networks and applications* (pp. 78–87).
38. Perkins, C., & Royer, E. (1997) Ad-hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE workshop on mobile computing systems and applications* (pp. 90–100).
39. Clausen, T., & Jacquet, P. (2003). *RFC-3626 optimized link state routing protocol (OLSR)*.
40. Karp, B. & Kung, H. T. (2000). Gpsr: greedy perimeter stateless routing for wireless networks. In *Proceedings of conference on mobile computing and networking, MobiCom '00* (pp. 243–254).
41. Li, J., Jannotti, J., De Couto, D. S. J., Karger, D. R., Morris, R. (2000). A scalable location service for geographic ad hoc routing. In *Proceedings of conference on mobile computing and networking, MobiCom '00* (pp. 120–130).
42. Madden, S., Franklin, M. J., Hellerstein, J. M., & Hong, W. (2002). Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Operating System Review 36*(SI), 131–146.
43. Batalin, M., & Sukhatme, G. S. (2002). Sensor coverage using mobile robots and stationary nodes. In *SPIE conference on scalability and traffic control in IP networks II (Disaster Recovery Networks)* (pp. 269–276).
44. Batalin, M. & Sukhatme, G. S. (2005). The analysis of an efficient algorithm for robot coverage and exploration based on sensor network deployment. In *IEEE international conference on robotics and automation* (pp. 3489–3496). Barcelona, Spain.
45. Dantu, K., & Sukhatme, G. S. (2009). Connectivity vs. control: Using directional and positional cues to stabilize routing in robot networks. In *International conference on robot communication and coordination*.
46. Dellaert, F., Fox, D., Burgard, W., & Thrun, S. (1999). Monte Carlo localization for mobile robots. In *Proceedings of IEEE international conference on robotics and automation*.
47. Fox, D., Thrun, S., Burgard, W., & Dallaert, F. (2001) Particle filters for mobile robot localization. In A. Doucet, N. de Freitas, N. Gordon (Eds.), Sequential Monte Carlo methods in practice (pp 401–428). Berlin: Springer.
48. Reina, G., Vargas, A., Nagatani, K., & Yoshida, K. (2007). Adaptive kalman filtering for gps-based mobile robot localization. In *Proceedings of the IEEE international workshop on safety, security and rescue robotics*.
49. Bahl, P., & Padmanabhan, V. N. (2000). RADAR: An in-building RF-based user location and tracking system. In *Proceedings of IEEE INFOCOM'00*.
50. Bulusu, N., Heidemann, J., & Estrin, D. (2000). Gps-less low-cost outdoor localization for very small devices. *IEEE Personal Communications Magazine, 7*, 28–34.
51. He, T., Huang, C., Blum, B. M., Stankovic, J. A., & Abdelzaher, T. (2003). Range-free localization schemes for large scale sensor networks. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking* (pp. 81–95). ACM: New York.
52. Priyantha, N. B., Chakraborty, A., & Balakrishnan, H. (2000). The cricket location-support system. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking* (pp. 32–43).
53. Greiner, G., & Hormann, K. (1998). Efcient clipping of arbitrary polygons. *ACM Transactions on Graphics 17*(2), 71–83.
54. Wein, R. (2006). Exact and efficient construction of planar minkowski sums using the convolution method. In *European symposium on algorithms*.
55. Vatti, B. R. (1992). A generic solution to polygon clipping. *Communications of the ACM, 35*(7), 56–63.
56. Guibas, L. J., & Hershberger, J. (1989). Optimal shortest path queries in a simple polygon. *Journal of Computer and Systems Sciences 39*(2), 126–152.
57. Allman, M., Paxson, V., & Stevens, W. (1999). Tcp congestion control.
58. Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic Robotics*. MIT Press:Cambridge, MA.
59. O'Kane, J. M. (2011). Decentralized tracking of indistinguishable targets using low-resolution sensors. In *Proceedings of the IEEE international conference on robotics and automation*.

Author Biographies

Jason M. O'Kane earned the B.S. degree in computer science from Taylor University in Upland, Indiana in 2001, and M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign in 2005 and 2007, respectively. He is currently an Assistant Professor in the Department of Computer Science and Engineering at the University of South Carolina. His research interests include algorithms for robotics, planning under uncertainty, artificial intelligence, computational geometry, and motion planning.



Wenyuan Xu received her Ph.D. degree in electrical and computer engineering from Rutgers University in 2007. She is currently an assistant professor in the Department of Computer Science and Engineering, University of South Carolina. Her research interests include wireless networking, network security and privacy. Dr. Xu is a co-author of the book Securing Emerging Wireless Systems: Lower-layer Approaches, Springer, 2009. She received the NSF Career Award in 2009.