Noname manuscript No.
(will be inserted by the editor)

Ghost : Voronoi-Based Tracking in Sparse Wireless Networks using Virtual Nodes

Francisco Garcia · Javier Gomez · Marco A. Gonzalez · Miguel Lopez-Guerrero · Victor Rangel

the date of receipt and acceptance should be inserted later

Abstract Conventional tracking techniques for wireless networks locate a target by using at least three non-collinear tracker nodes. However, having such a high density of trackers over the monitored area is not always possible. This paper presents Ghost, a new tracking method based on Voronoi tessellations able to track a target by using less than three tracker nodes in wireless networks. In Ghost, different locations of the target create different Voronoi diagrams of the monitored area by placing *virtual* nodes around tracker nodes. These diagrams are used to estimate the current location of the target by intersecting the previous and current Voronoi diagrams. The target's route is constructed by systematically searching the most likely estimated target's locations over time. Simulation results validate that the proposed method has better tracking accuracy compared with existing proposals. Moreover, our approach is not tied to a specific technology, thus it can be applied in different platforms (e.g., WLAN and WSN).

Keywords Tracking · Voronoi tessellation · Sparse Wireless Networks

Francisco Garcia (✉) · Marco A. Gonzalez
Department of Computer Sciences, IIMAS,
National Autonomous University of Mexico, Mexico City 04510.
E-mail: {lgarciaj,marco}@uxmcc2.iimas.unam.mx

Javier Gomez · Victor Rangel
Department of Telecommunications Engineering,
National Autonomous University of Mexico, Mexico City 04510.
E-mail: {javierg, victor}@fi-b.unam.mx

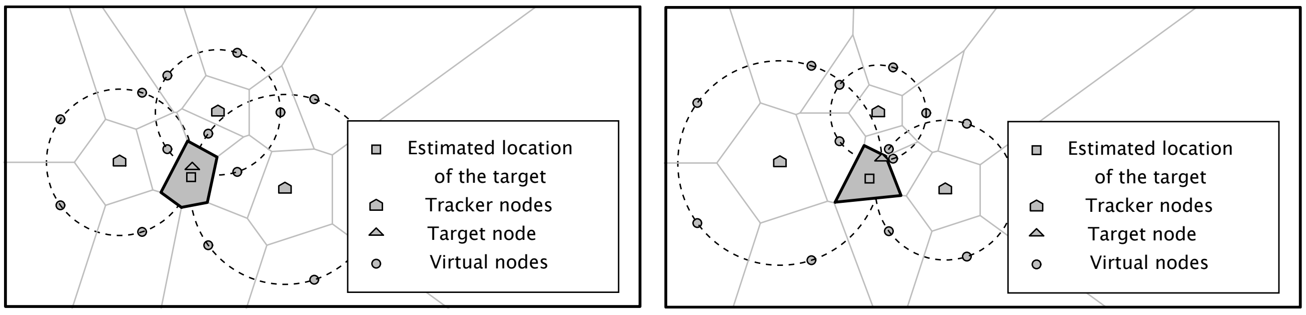
Miguel Lopez-Guerrero
Department of Electrical Engineering,
Metropolitan Autonomous University, Iztapalapa Mexico City 09340.
E-mail: milo@xanum.uam.mx

1 Introduction

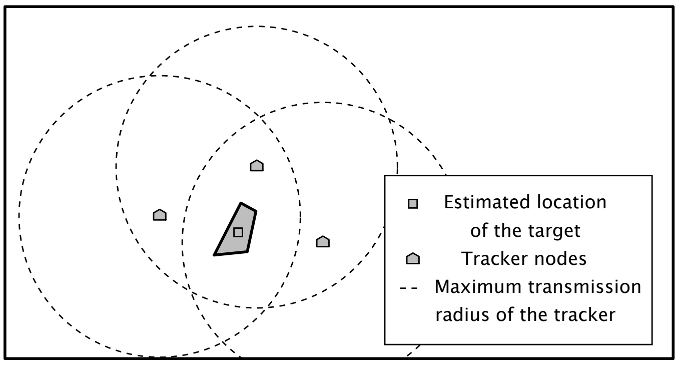
Tracking systems in wireless networks are usually composed by a set of scattered fixed nodes in the monitored area that are able to estimate the target's route over time. Estimating a target's route in wireless networks is a nontrivial task due to the presence of several factors in the monitored area that make network conditions change frequently such as: obstacles, signal interference, irregular areas and power outages. These factors may also cause changes in network topology and may cause loss of connectivity in some areas. On the other hand, tracking systems typically use trilateration methods by using three or more non-collinear nodes to locate a transmitting target [1,2]. The problem with this method is that not all network infrastructures can guarantee the presence of three tracker nodes everywhere in the network. For instance, in home, hotspot and university networks, having connectivity by a single Access Point (AP) is considered satisfactory. We consider that having less than three tracking nodes detecting the presence of a moving node is a common situation.

Over the past years, various proposals focused on increasing tracking accuracy and energy efficiency [3] [4]. However, less attention has been paid in scenarios where less than three fixed tracking nodes are available to estimate the target's location/route. To overcome this limitation we propose Ghost, a new tracking method based on Voronoi tessellations [5] able to track a target by less than three tracker nodes in wireless networks. Ghost is composed by three main elements:

– A target node. It is a mobile node that has a wireless transceiver.
– A tracker node. It is a fixed wireless node within the monitored area that has the abilities of sensing and



(a) Voronoi diagram at time $t-1$ (b) Voronoi diagram at time t


Estimated location of the target
Tracker nodes
-- Maximum transmission radius of the tracker

(c) Intersection between the previous and current chosen polygons

Fig. 1 Ghost Operation: (a) Ghost selects the most probable polygon where the target node is located at time $t-1$ (see polygon filled in gray color). (b) When a new detection is obtained at time t, Ghost selects again the most probable polygon where the target node is located. (c) Ghost intersects the previous and current selected polygons in order to create a smaller polygon enclosing the target's location

computing the distance between the tracker and the target node [6].
- A sink node. It is a processing center that collects information from several tracker nodes and is able to compute the target's route using the Ghost algorithm.

In Ghost, locating a target node involves three phases. First, tracker nodes estimate how far from the target node they are and report the estimated distance to the sink node. Second, the sink node divides the monitored area into small regions by using a Voronoi tessellation algorithm [5] that takes into account two sets of points. The first set is composed by tracker nodes in the monitored area given by their cartesian coordinates, see Figures 1(a) and 1(b). The second set is composed by *virtual* nodes. Virtual nodes are placed along a circle centered on each tracker node having a radius equal to: (*i*) the estimated distance reported to the sink node by trackers detecting the target node (see Figures 1(a) and 1(b)), (*ii*) the maximum transmission range for trackers not detecting the target node. By placing virtual nodes around each tracker node, Ghost can create different Voronoi diagrams for each new location of the target node. Moreover, by increasing the number of virtual nodes, Ghost can narrow down the location of the target node into a smaller Voronoi region (i.e., convex polygon) due to the fact that the number of Voronoi regions increases linearly with the number of virtual nodes [5]. Third and finally, Ghost utilizes the current Voronoi diagram to select the most probable polygon(s) where the target node is located by using a point location algorithm [5, 7].

Since Voronoi diagrams of the monitored area change while the target node moves (see Figures 1(a) and 1(b)), Ghost estimates the target's route by intersecting the current polygon(s) chosen by point location algorithm at time t with selected polygon(s) computed at time $t-1$. The result of each intersection (if it exists) is a smaller polygon(s) enclosing the likely location of the target node, see Figure 1(c). These polygons are used to built a graph G where the center of each polygon is used as a vertex of the target's route (i.e., the estimated location of the target node).

The main contribution of this paper is that whereas most tracking proposals require continuous coverage by three or more tracker nodes to operate properly, we propose a technique able to track a target node by two or even one tracker node. In this way, our approach is more feasible to implement given that it considers more realistic scenarios where only one or two trackers detect the target node; we named these scenarios as {one,two}-coverage detection. Moreover, we introduce a new technique to narrow down the area where the target node is located by placing virtual nodes around real trackers.

The rest of this paper is organized as follows. In Section 2, we present previous works related to localization and tracking for wireless networks. In Section 3, we present the assumptions and computational algorithms used by Ghost. Section 4 defines the concept of maximum speed circle (MSC) that provides additional information for cases when only one or two trackers detect the target node. In Section 5, we present a quantitative evaluation of Ghost in a network simulator written in Python language and we compare our approach with existing tracking techniques. Finally, in Section 6, we present our final remarks.

2 Related work

Tracking a target is typically based on localization approaches that fall into two categories: range-based approaches and range-free approaches. Range-based approaches use various techniques to estimate the location of a target such as: Time of Arrival (ToA) [8], Time Difference of Arrival (TDoA) [9,10], Angle of Arrival (AoA) [11] and Received Signal Strength (RSS) [12–14]. Numerous range-based approaches use a trilateration technique to enhance the accuracy of target's location. For instance, in [1], the authors proposed a trilateration approach to decrease location errors by applying Kalman filtering techniques over RSSI signals from a WLAN infrastructure. In [2], the authors proposed a trilateration approach to locate APs around a neighborhood by mounting a steerable directional antenna on a set of vehicles driving around the neighborhood. Range-free schemes, on the other hand, are based on proximity and connectivity schemes. In [4], the locations of wireless sensor nodes are found by extracting relative location information from different nodes at different times in order to reconstruct a two-dimensional map of the monitored area. In [15], the authors proposed a localization technique by using a real-valued hop-count instead of a traditional integer-valued hop-count in order to improve the multidimensional scaling method (MDS). APIT algorithm [16], locates wireless sensor nodes by dividing the monitored area into triangular regions in which a sensor node resides. Depending on whether the node is inside or outside the triangle formed by three anchors, this algorithm can narrow down the area where a node is located. For more information on localization approaches the readers is referred to [17] where the authors presented a through survey.

Once the position of the target can be estimated over time, the next challenge is to estimate the trajectory of a target node as it roams in the monitored area. Most tracking approaches based their operation on predictive probabilistic schemes using Kalman filters [18] [19] [20]. For instance, in [20], the authors applied a Kalman filter to a set of ToA measurements to smooth out the estimated target's route.

Nowadays computational geometry emerged with new algorithmic techniques that improve and simplify many of the previous approaches. For example, in [21], the authors proposed a human tracking system for indoor environments by placing sparse infrared and ultrasound sensors. This algorithm can distinguish the identity and location of a person by using learning techniques on human motion and Voronoi graphs. A tracking algorithm based on a face routing technique is proposed in [3], where the authors used two types of sensors named monitor and backup. This algorithm detects target's movements by selecting the most likely monitor sensor associated to a face region in order to form a linked list of monitor and backup sensors as a route for the target node. In [22], the authors proposed the first tracking framework based on a polygonal spatial neighborhood by partitioning the monitored area into polygonal regions using planar graph algorithms. The main idea of this framework is to select the most probable polygonal edges the target node is crossing.

In summary, although tracking a target is extremely difficult to carry out even under simple scenarios, previous proposals do not consider the case where less than three fixed tracking nodes detect the presence of the target node. To the best of our knowledge, the only work dealing with this problem is [23], where the authors proposed two algorithms based on predictive information obtained from a Kalman filter. The first algorithm proposed in [23] is called predictive location tracking (PLT), which uses the predictive information obtained by a Kalman filter to provide signals from missing base stations (BS) in order to let trilateration techniques work properly. The second algorithm called geometric-assisted PLT (GPLT) adjusts the location of missing BSs by using geometric dilution of precision metric (GDOP). Because the two methods reported in [23] represent the closest approaches to our work, we compare their performance with Ghost in Section 5.

3 Ghost System Model

We consider a set of N tracker nodes placed randomly on a two dimensional rectangular space with unique identifiers. For each tracker node located at point p (i.e., (x,y)), we represent the maximum transmission range as a unit disk graph (UDG) $C_u(p,R_c)$ centered at point p having a unit radius R_c. We assume that transmission range is uniform and has constant radius. A tracker node can detect a target node if and only if the target node is within the tracker's UDG coverage. We modeled the estimated route of the target node as a directed graph $G(V,E)$, where the center of a chosen polygon represents a vertex $v \in V$. Two different vertices share a common edge $e(u,v) \in E$ if and only if vertex u is chosen by Ghost at time $t-1$ and v is chosen at time t. We assume that each tracker node knows its own location (e.g., by means of GPS [24,25] or other localization method [15–17]) and this location is sent to the sink node.

3.1 Euclidean space division

In Ghost, whenever the target node is detected by at least one tracker node, Ghost places a circle centered on each tracker having a radius equal to: i) the Euclidean distance between a tracker node and the target node for trackers detecting the target node. ii) the maximum transmission range (R_c) for trackers not detecting the target node. These circles are divided into m equal arcs, where two arcs share a point called *virtual* node as shown in Figures 1(a) and 1(b). These virtual and tracker nodes are used as input to the Voronoi algorithm in order to divide the Euclidean space of the monitored area into smaller regions. The Voronoi algorithm creates $m \times N + N$ convex polygons, where N is the number of tracker nodes (i.e., real nodes in the monitored area) and m is the number of virtual nodes per tracker node. In other words, each tracker node and each virtual node in the monitored area has an associated convex polygon. The time complexity cost of the Voronoi tessellation algorithm is $O(n \; log \; n)$ [5], where n is equal to $m \times N + N$. It is important to mention that the number of virtual nodes creates a tracking accuracy and time complexity cost trade-off that will be discussed in Section 5.

In Ghost, tracking a target involves three main cases named {three,two,one}-coverage detection, which are described in the following subsections. We present the operation of Ghost without considering distance estimation errors and we assume that the Euclidean space is divided by the sink node using the Voronoi algorithm [5,21,26]. Later, we will introduce an error model in the simulation section.

3.2 Three-coverage detection

Ghost uses a trilateration technique [1] whenever the target is detected by three or more tracker nodes. This technique returns the location of the target node as point p. This point strictly lies within a polygon $\mathcal{P}$ in the monitored area. Ghost uses point location algorithm [5][7] to find polygon $\mathcal{P}$. There are several point location algorithms, but Ghost uses slab decomposition [5] that works as follows: the current polygonal subdivision of the monitored area (given by the current Voronoi diagram) is partitioned by drawing vertical lines through each vertex. Two consecutive vertical lines form a slab region in which the x-coordinate of each vertex is sorted in an array. The slab decomposition algorithm uses binary search in $O(log \; n)$ time to find the x-coordinate of point p. A similar process is used for the y-coordinate. Then, the x and y coordinates of point p are linked to a unique polygon $\mathcal{P}$ within the polygonal subdivision. In a real scenario; however, the estimated location of the target node may be different to its real location due to distance estimation errors. Therefore, Ghost uses the center of the chosen polygon $\mathcal{P}$ as the most appropriate estimated location of the target node.

In cases when the target node is located by three trackers at consecutive times, the target's route is constructed as follows: let $\mathcal{P}$ and $\mathcal{Q}$ be two polygons chosen by Ghost at time t and time $t-1$, respectively. Let $\mathcal{Z}$ be the convex polygon resulting by intersecting the polygons $\mathcal{P}$ and $\mathcal{Q}$. Ghost constructs target's route by linking the center of polygon $\mathcal{Q}$ with the center of polygon $\mathcal{Z}$, and the center of polygon $\mathcal{Z}$ with the center of polygon $\mathcal{P}$ (i.e., Ghost links $v_{\mathcal{Q}} \rightarrow v_{\mathcal{Z}} \rightarrow v_{\mathcal{P}}$). In case polygon $\mathcal{Z}$ is $\emptyset$ (i.e., no intersection occurs between polygons $\mathcal{P}$ and $\mathcal{Q}$), Ghost links the center of polygon $\mathcal{P}$ with the center of polygon $\mathcal{Q}$ (i.e., $v_{\mathcal{Q}} \rightarrow v_{\mathcal{P}}$). Ghost computes polygon intersection in $O(m+n)$ time [27], where m and n are the vertices of polygons $\mathcal{Q}$ and $\mathcal{P}$, respectively.

All locations of the target node obtained by at least three tracker nodes are called *hook* points. These points might be used as anchor points in order to reduce the uncertainty of possible locations of the target node when less than three tracker nodes detect the target node, as it is shown in the following subsections.

The operation of Ghost in pseudo code for a three-coverage detection case is shown in Algorithm 1.

3.3 Two-coverage detection

Figure 2(a) shows the case when only two tracker nodes (v_1, v_2) detect the target node, resulting in an overlapping region created by intersecting the coverage area

Algorithm 1 Three-coverage detection.

```
Require: Define $m$ (number of *virtual* nodes per tracker).
 1: if a target node is detected by at least three tracker nodes then
 2:     Send the estimated distances to the sink node.
 3:     Compute Voronoi diagram of the monitored area.
 4:     Select polygon $\mathcal{P}$ where the target node is located.
 5:     Link the center of polygon $\mathcal{P}$ (i.e., $v_\mathcal{P}$) with graph $G$.
 6:     Concatenate $v_\mathcal{P}$ to the set of hook points.
 7: end if
 8: if an intersection takes place between polygons $\mathcal{Q}$ and $\mathcal{P}$ then
 9:     Compute polygon $\mathcal{Z}$.
10:     Link $v_\mathcal{Q} \to v_\mathcal{Z} \to v_\mathcal{P}$.
11: else
12:     Link $v_\mathcal{Q} \to v_\mathcal{P}$.
13: end if
```

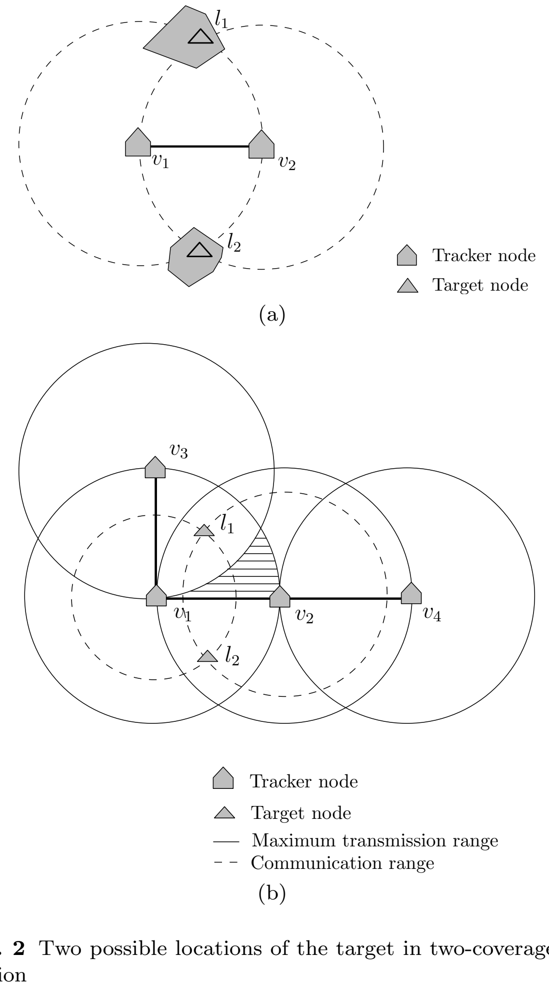


Fig. 2 Two possible locations of the target in two-coverage detection

of each tracking node. This intersection generates two possible locations of the target node (i.e., l_1 and l_2). One of these locations is the real location of the target node and the other is called the mirrored location.

The mirrored location of the target can be discarded in some special situations. We illustrate this case in Figure 2(b), where tracker nodes v_1 and v_2 detect the presence of the target node and send their estimated distances to the sink node. The sink node can eliminate the mirrored location of the target node if and only if the mirrored location lies within the coverage area of the nearest neighbors (e.g., v_3 and v_4 in Figure 2(b)) to the two-coverage detection, since these neighbors do not detect the presence of the target node. In the example of Figure 2(b), the real location of the target is l_2, while l_1 is the mirrored location. Location l_1 lies within v_3's coverage area, thus it can be discarded by the sink node since node v_3 does not detect the target node. In the same example, however, if the mirrored location lies in the shadowed region (see Figure 2(b)), it is not possible to discard the mirrored location since this location lies outside v_3 and v_4 coverage area. In Section 4 we will use the concept of maximum speed circle (MSC) to discard the mirrored location in cases when the previous method does not apply.

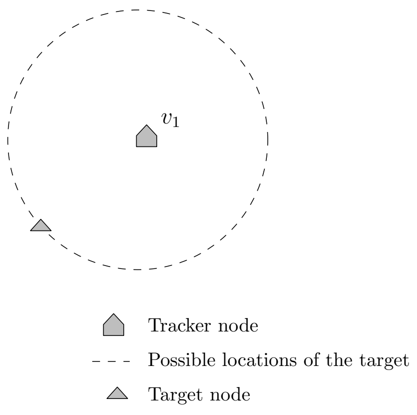
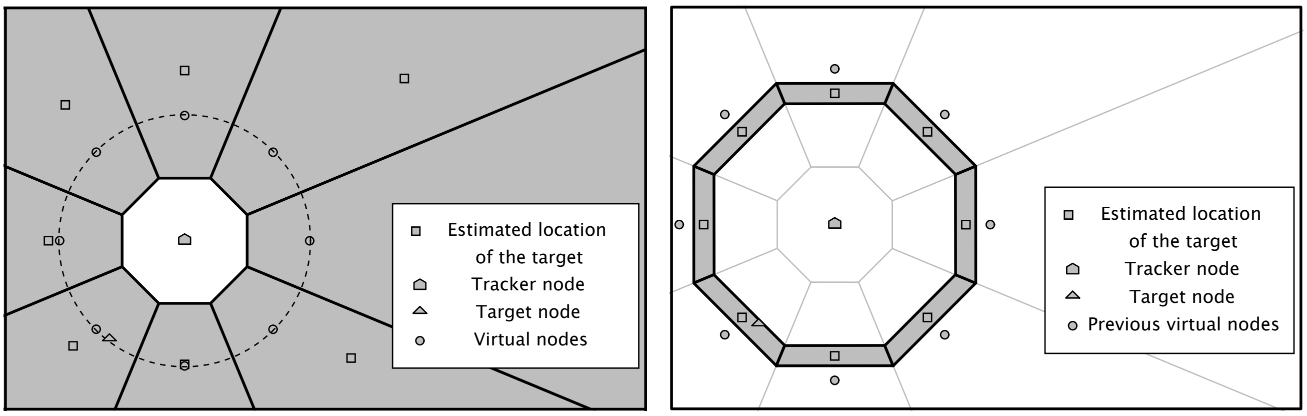


Fig. 3 One-coverage detection

3.4 One-coverage detection

In sparse networks, one-coverage detection is a common situation resulting in an infinite number of possible locations of the target node around the tracker node, see Figure 3 where a target node is detected by one tracker node only. Ghost overcomes this drawback by placing virtual nodes around the tracker node detecting the target. Figure 4(a) shows a Voronoi diagram formed by eight virtual nodes and one tracker node. As we can see in this figure, using the center of the chosen polygons (i.e., polygons filled in gray) as likely location of the target might result in a large location error. This occurs because some of these polygons may have unbounded edge(s). To solve this problem, Ghost limits the size of each chosen polygon in one-coverage detection by intersecting computed polygons at time t with computed polygons at time $t-1$. Figure 4(b) illustrates the result of this process. Note that we use virtual nodes



(a) Voronoi diagram formed by eight virtual nodes and one tracker node

(b) Donut constructed by intersecting polygons at time $t-1$ with polygons at time t

Fig. 4 One-coverage detection using virtual nodes

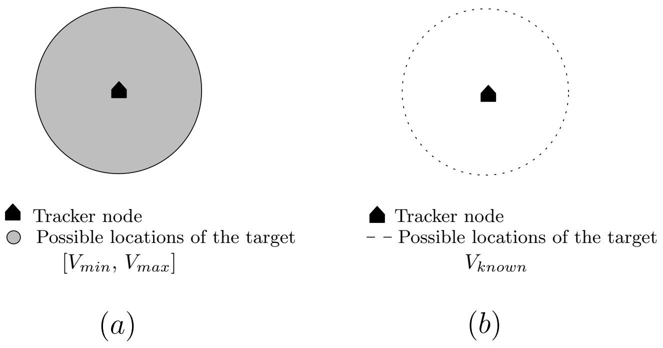
computed at time $t-1$ to compute Voronoi diagram at time t. This process constructs a set of polygons in which each polygon (filled in gray) is bounded by four edges, as shown in Figure 4(b). We called this set of polygons a *donut*.

4 Maximum Speed Circle

This section introduces the concept of maximum speed circle (MSC), which is used to provide additional information when a target node is located within {one,two}-coverage detection. We define the MSC as a circle containing all possible locations of the target between two consecutive target's detections. The MSC can be seen as a circle with zero radius located at the center of previous selected polygon, increasing its radius at a certain speed until a new detection is obtained. For cases where the speed is unknown but bounded in the interval [0, V_{max}], the MSC increases its radius at a maximum target's speed. All points inside the MSC are possible locations of the target node because it can move freely between two consecutive detections (see the gray area in Figure 5(a)). In Ghost, we also consider the case when target's speed is known or it can be estimated (V_{known}). In this case, only points in the perimeter of MSC are valid locations of the target node as long as the target node does not change its trajectory in the interval [$t-1$, t], as shown in Figure 5(b).

4.1 Two-coverage detection using MSC

MSC can be used to discard the mirrored location of the target node in a two-coverage detection as follows: the MSC is initially located at the last known location of the target (e.g., hook point or closest known location(s)



Tracker node
Possible locations of the target
$[V_{min}, V_{max}]$
(a)
Tracker node
Possible locations of the target
V_{known}
(b)

Fig. 5 Maximum Speed Circle Scope

before entering the two-coverage detection) with radius equal to zero. Then, the MSC increases its radius given the speed of the target node (e.g., V_{max} or V_{known}) until a new detection is obtained. Ghost can discard the mirrored location if and only if the mirrored location falls outside the MSC. To illustrate this, Figure 6 shows the last hook point of the target node (shown as □) and the overlapping region created by two tracking nodes (v_1 and v_2). In this example, location l_2 lies outside the MSC, thus it can be discarded. Note that the last hook point necessarily was computed at time $t-1$. If no hook point or known locations exist, then Ghost cannot discard the mirrored location of the target, because no MSC can be placed at any previous known location.

Target's route within two-coverage detection is built by connecting the center of the current MSC with the center of the polygon enclosing the chosen target's location (see Figure 7(a)). In case the mirrored location cannot be discarded (i.e., both, real and mirrored location are inside the current MSC), Ghost links the center of the current MSC with the center of the polygon enclosing the real and mirrored locations (see Figure 7(b)). Then, a new MSC with radius equal to zero is

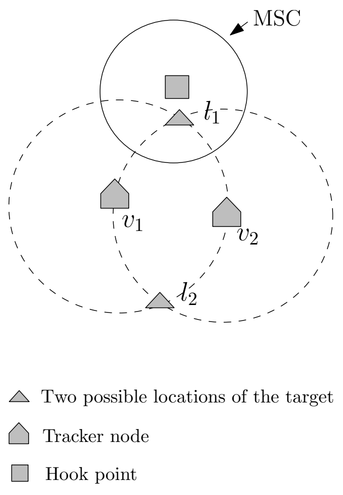


Fig. 6 Two tracker nodes using MSC

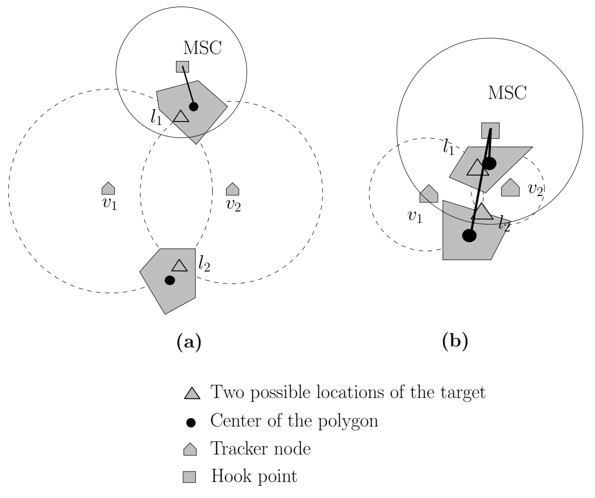


Fig. 7 Building target's route in two-coverage detection using MSC

Algorithm 2 Two-coverage detection.

```
Require: Define $m$ (number of virtual nodes per tracker).
 1: if a target node is detected by two tracker nodes then
 2:     Send the estimated distances to the sink node.
 3:     Compute Voronoi diagram of the monitored area.
 4:     if mirrored location is discarded by the current MSC then
 5:         Select polygon $\mathcal{P}$ where the target node is located.
 6:         Link the center of the current MSC to the center of
            polygon $\mathcal{P}$.
 7:         Place a new MSC at the center of the polygon $\mathcal{P}$ with
            radius equal to zero.
 8:     else
 9:         Select the two polygons where the target node is located
            (i.e., real and mirrored location).
10:         Link the center of the current MSC to the center of the
            two chosen polygons.
11:         Place a new MSC at the center of each chosen polygon
            (i.e., $v_{real}$ and $v_{mirrored}$) with radius equal to zero.
12:     end if
13: end if
14: Go to 1 until the target node leaves the two-coverage detec-
    tion.
```

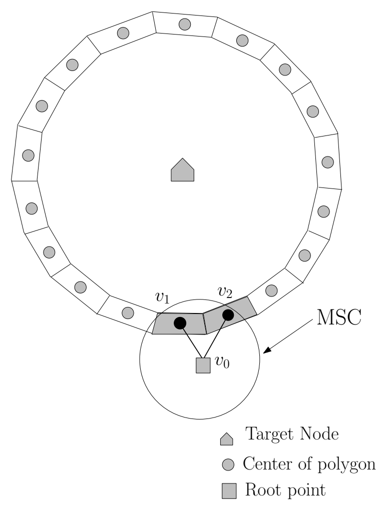


Fig. 8 Building graph G within one coverage detection

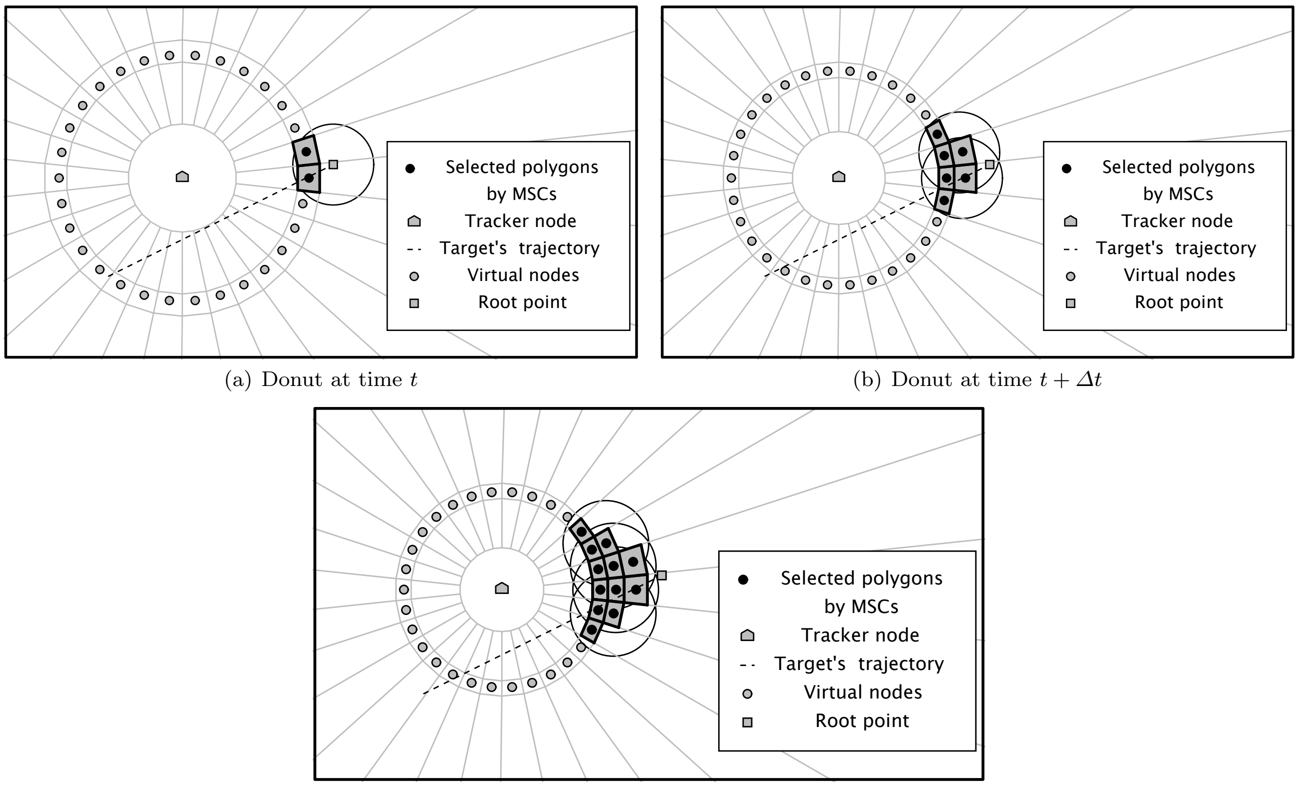
placed at the center of each chosen polygon (i.e., the new hook point) in order to repeat the process until the target node leaves the two-coverage detection.

In cases where the target node is located by two trackers at consecutive times, the target's route is constructed as follows: let $\mathcal{P}$ and $\mathcal{Q}$ be two polygons chosen by Ghost at time t and time $t-1$, respectively. Let $\mathcal{Z}$ be the convex polygon obtained by intersecting polygons $\mathcal{P}$ and $\mathcal{Q}$. Ghost constructs target's route by linking the center of polygon $\mathcal{Q}$ with the center of polygon $\mathcal{Z}$, and the center of polygon $\mathcal{Z}$ with the center of polygon $\mathcal{P}$ (i.e., Ghost links $v_{\mathcal{Q}} \rightarrow v_{\mathcal{Z}} \rightarrow v_{\mathcal{P}}$). In case polygon $\mathcal{Z}$ is $\emptyset$, Ghost links the center of polygon $\mathcal{P}$ with the center of polygon $\mathcal{Q}$ (i.e., $v_{\mathcal{Q}} \rightarrow v_{\mathcal{P}}$).

The operation of Ghost in pseudo code for a two-coverage detection is shown in Algorithm 2.

4.2 One-coverage detection using MSC

Now let us explain how Ghost uses MSC to construct the target's route in one-coverage detection. For instance, consider the last location of the target is known before entering the one-coverage detection (e.g., hook point). This point represents the root point in one-coverage detection where a MSC with zero radius is located. Then, the MSC's radius increases until a new detection is obtained. This detection necessarily lies into one-coverage detection where a new donut is constructed by using virtual nodes at time t and virtual nodes at time $t-1$. In case no virtual nodes at time $t-1$ are available, Ghost computes virtual nodes using a circle centered on the tracker node having a radius equal to the Euclidean distance between the tracker node and the hook point. To illustrate this, Figure 8 shows the



(a) Donut at time t

(b) Donut at time $t + \varDelta t$

(c) Donut at time $t + 2\varDelta t$

Fig. 9 Polygons selected by Ghost as time passes

current MSC centered at point v_0 (i.e., the root point) and the current donut. Ghost constructs target's route by connecting the center of the current MSC with the center of polygons located inside the current MSC. In this example, two edges are formed. The first edge connects nodes v_0 and v_1, while the second edge connects nodes v_0 and v_2. Then, a new MSC with radio equal to zero is placed on nodes v_1 and v_2 in order to repeat the process until the target node leaves the one-coverage detection. Figure 9 shows this process as time passes.

It is important to mention that before constructing the current donut, Ghost determines the scope of one-coverage detection. Figure 10 shows v_1's coverage area overlapping with two other trackers (v_2 and v_3). In this figure we can see that only points around the one-coverage detection are chosen as virtual nodes. In order to determine the scope of one-coverage detection, Ghost uses *kd-tree* algorithm [28] to find the nearest neighbors in $O(n \; log \; n)$ time.

The operation of Ghost in pseudo code for one-coverage detection case is shown in Algorithm 3.

Algorithm 3 One-coverage detection.

```
Require: Define $m$ (number of virtual nodes per tracker).
1: if a target node is detected by one tracker node then
2:     Send the estimated distance to the sink node.
3:     Compute current donut.
4:     Link the center of the current MSC(s) with the center of
       the polygon(s) lying inside the current MSC(s).
5:     Place new MSC with radio equal to zero on each selected
       location.
6: end if
7: Go to 1 until the target node leaves the one-coverage detection
```

In one-coverage detection, the larger the number of virtual nodes, the larger the number of possible paths created while tracking a target. In order to seek an optimal path solution Ghost uses the backtracking algorithm (BT) [29]. But before we approach the optimal path solution, it is important to explain how Ghost ends the paths within the one-coverage detection (i.e., endpoints in graph G for one-coverage detection). Figure 11 illustrates a target node (shown as $\triangle$) near the boundary of one-coverage detection. Points v_j, v_k, v_m, v_n, v_l are the center of each current MSC, respectively. Only the maximum speed circles reaching the next known location of the target (v_z) are chosen as endpoints of the graph. Note that v_z is necessarily located outside the one-coverage detection. In this example, points v_k and v_m are chosen as endpoints of possible paths in one-coverage detection.

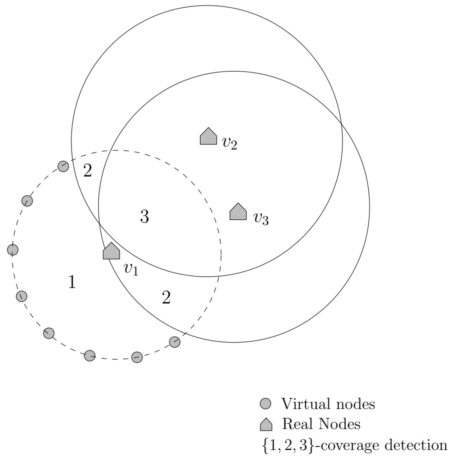


Fig. 10 Scope of one-coverage detection

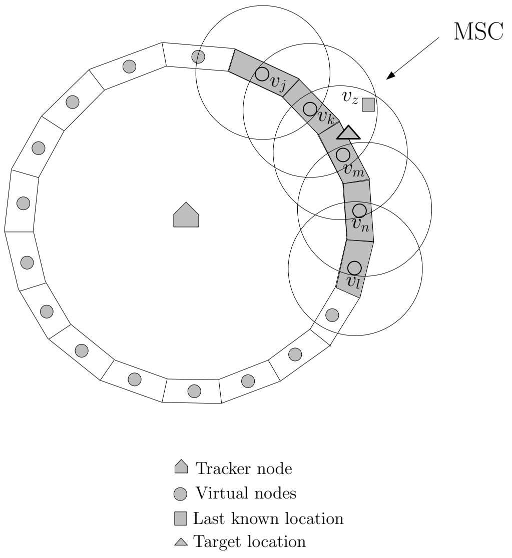


Fig. 11 Endpoints in one-coverage detection

In the event that no hook points exist before entering the one-coverage detection, Ghost can use any previous location(s) discovered by two-coverage detection. This includes the real location, mirrored location or both as well as a set of endpoints discovered in a previous one-coverage detection. In case no points exist before entering the one-coverage detection, Ghost cannot place any MSC in order to reduce the uncertainty. However, this is an unlikely event as Ghost eventually will find points in a two-coverage or tree-coverage detection as time passes which can be used as hook points before entering a one-coverage detection.

Once the endpoints are obtained, Ghost seeks the optimal path solution in one-coverage detection by using the backtracking algorithm which is a simple recursive technique to optimize searches. This algorithm seeks the best partial solution systematically by satisfying a constraint function in order to obtain a global optimal solution. In Ghost, these constraints are the set of endpoints and the number of detections during target's movements within the one-coverage detection. These constraints imply that only paths whose lengths are equal to the number of detections in one-coverage detection are valid paths. The backtracking algorithm allows Ghost to find: *i*) all paths between the root point and all endpoints, *ii*) the shortest path between the root point and a chosen endpoint, *iii*) a random path. Because the backtracking algorithm accounts for most of the processing time in Ghost, we analyze the BT worst-case running time below.

A constraint satisfaction problem (CSP) consists of a set of variables, $X = \{x_1, ..., x_n\}$, where each variable $x_i \in X$ has an associated finite domain $D = \{D(x_1), ..., D(x_n)\}$, and a set of constraints C, where each C_i is in the form of $C_i = \{X_{i1}, X_{i2}, ..., X_{ij}\}$ [30]. A solution in a constraint problem is an assignment of a value $D(x_i)$ that satisfies one or all constraints. If no solution exists, the CSP is inconsistent. For any partial solution $(v_1, v_2, ..., v_i)$, the BT algorithm tries to assign the next value to the next variable $(v_1, v_2, ..., v_i, v_{i+1})$. If this partial solution satisfies the constraints $C_{j(i+1)} \in C$, then the BT algorithm tries to assign a new value to the next variable. In case the partial solution is inconsistent, then the BT algorithm tries to assign another value from $D(x_{i+1})$. If a value cannot be assigned, the BT backtracks to x_i and tries another value $D(x_i)$. The search space to assign a consistent value is at most $\sum_{i=0}^{n} d^i = \frac{d^{n+1}-1}{d-1}$, where n is the number of variables X (in Ghost this is the number of polygons selected by MSC) and d is the size of largest domain. Therefore, the time complexity for every assignment $D(x_i)$ is at most $O(d^n)$. The BT searches in n vertices, thus, the total running time for the BT algorithm is at most $O(nd^n)$. A filtering technique can be applied to optimize the search space in CSP. For example, when a value $D(x_i)$ is inconsistent with the $C_{j(i)}$, we can remove or prune the search space for futures variables. In Ghost, when a path is larger than the number of detections in one-coverage detection, we can prune the tree to reduce the size of current domain. In [30] the authors analyzed the cut-set decomposition algorithm whose time complexity cost by using a filtering technique is $O(nd^2)$.

On the other hand, it is easy to see that one-coverage algorithm is the main algorithm in Ghost, while the two-coverage and three-coverage algorithms are a particular case of one-coverage algorithm. Indeed, the BT algorithm can be used in two-coverage detection to discard the mirrored location(s) by seeking the optimal path solution using the available endpoints and the number of detections of the target in two-coverage detection as constraints.

5 Simulation and results

In this section, we asses the location errors of our tracking method. First, we study tracking accuracy as a function of the number of virtual nodes by isolating {three,two,one}-coverage detection cases. Then, we study tracking performance on a larger network having many tracker nodes. Finally, in Section 5.4, we compare Ghost's performance versus PLT and GPLT algorithms [23].

5.1 Noise Model

As mentioned before, in {three,two,one}-coverage detection, the target's locations are associated to the center of chosen polygons by Ghost. Thus, let us define the mean tracking error as the average Euclidean distance between the center of the polygon enclosing the estimated location of the target by Ghost and the real target's location for all points in the trace. For trackers detecting the presence of the target node, Ghost estimates the distance between a tracker node and the target node as:

$$r_{i,k} = d_{i,k} + n_{i,k} + e_{i,k} \qquad i = 1, 2, 3, , N \tag{1}$$

where $r_{i,k}$ denotes the estimated distance between the ith tracker and the target node at time k. $d_{i,k}$ is the real Euclidean distance between the ith tracker node and the target node at time k. $n_{i,k}$ is the added noise, which is assumed to follow a Gaussian distribution with zero mean and three meters of standard deviation. Finally, $e_{i,k}$ denotes the non-line-of-sight (NLOS) error [31], which is modeled using an exponential distribution as follows.

$$e_{i,k}(v) = \begin{cases} \frac{1}{\lambda_{i,k}} exp(-\frac{v}{\lambda_{i,k}}) & v > 0 \\ 0 & otherwise \end{cases} \tag{2}$$

where $\lambda_{i,k} = c \cdot \tau_m (d_{i,k})^{\epsilon} \rho$. Parameter c is the speed of light, τ_m is the median value of the RMS transmission delay between the ith tracker and the target, which is selected as $0.1 \mu s$ in the simulation. ϵ is the path loss exponent which is selected as 0.5. Finally, ρ is the shadow fading factor which is a log-normal random variable with zero mean and a standard deviation of 4dB. Values used in this noise model were extracted from [23].

Parameter	Value
Transmission range	100m
Monitored area length	1900m
Monitored area width	1100m
Origin of coordinates	left upper corner
Standard deviation of Gaussian distribution	$\mathcal{N}(0,9)$
Speed Interval	[1 - 5]m/s
Time interval between two consecutive detections	1s
Mobility Model	RWP

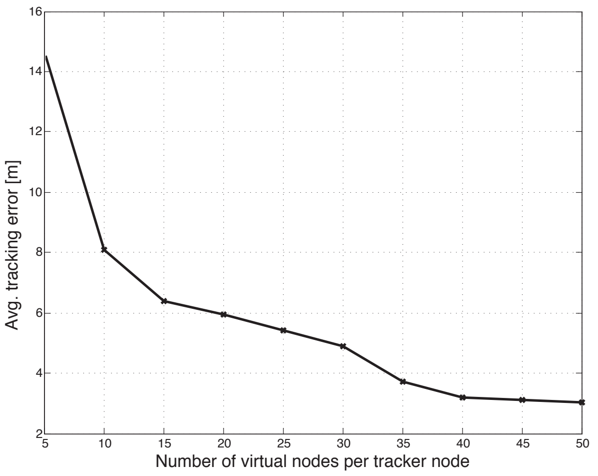
Table 1 Simulation parameters

In the following subsection, we evaluated our system by running simulations on a custom simulator written in Python language. First, we evaluated simulations for {three,two,one}-coverage detection in isolation. Nodes move according to the Random Way Point model (RWP) [32] and we ran each experiment 50 times to get average values. The parameters used for these experiments are shown in Table 1.

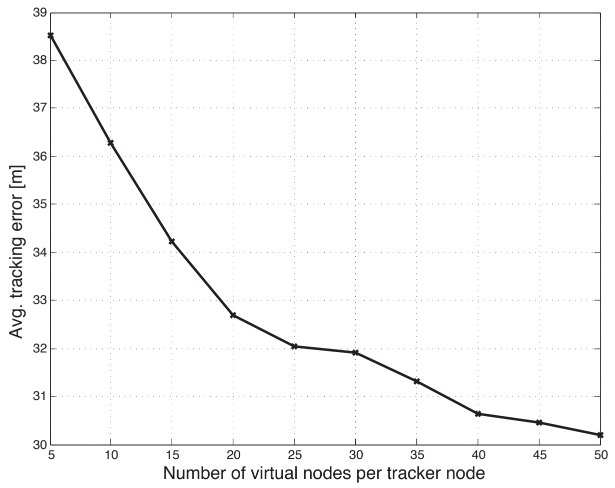
5.2 Experiments in isolated {three-two-one}-coverage detection

For the three-coverage detection experiments, tracker$_1$ is located at coordinates (558, 287), tracker$_2$ is located at (498, 285) and tracker$_3$ is located at (530, 333), all coordinates are in meters. Target's trajectory uses a RWP model whose endpoints are located in (501, 339) and (553, 259) in meters. The trajectory is chosen within the overlapping region created by the three tracker nodes. We evaluate tracking errors using a different number of virtual nodes per tracker ranging from 5 to 50 in steps of 5. Figure 12(a) shows the mean tracking error of isolated three-coverage detection vs. number of virtual nodes per tracker node using the coordinates mentioned above. This figure shows how tracking errors decrease as the number of virtual nodes increases, which is the result of having smaller polygonal regions enclosing the target node's location.

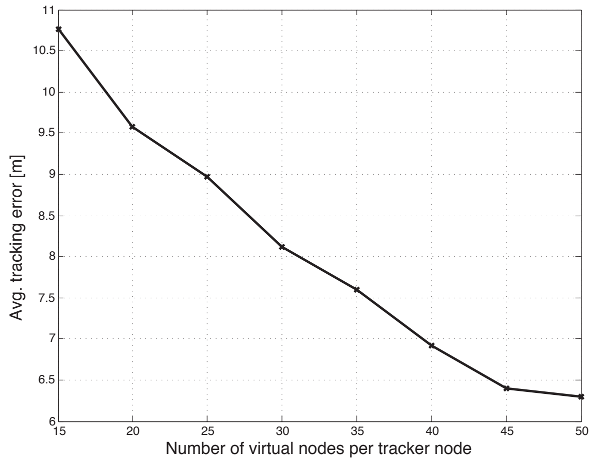
For the two-coverage detection experiments, tracker$_1$ is located at (521, 300) and tracker$_2$ is located at (597, 300) in meters. Target's trajectory uses a RWP model whose endpoints are located in (529, 356) and (588, 247) in meters. As we mentioned before, when two trackers detect the target, two possible paths are formed along the target's trajectory. We report the mean tracking error by estimating the distance between the centers of both associated polygons with respect to the real target's location. In these experiments no MSC was used, thus no mirrored locations were discarded. Figure 12(b) shows how tracking errors decrease as the number of



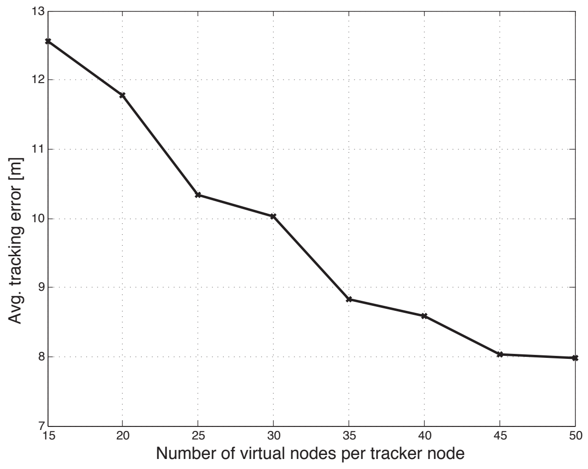
(a) Average tracking error in three-coverage detection



(b) Average tracking error in two-coverage detection



(c) Average tracking error in one-coverage detection using V_{known}

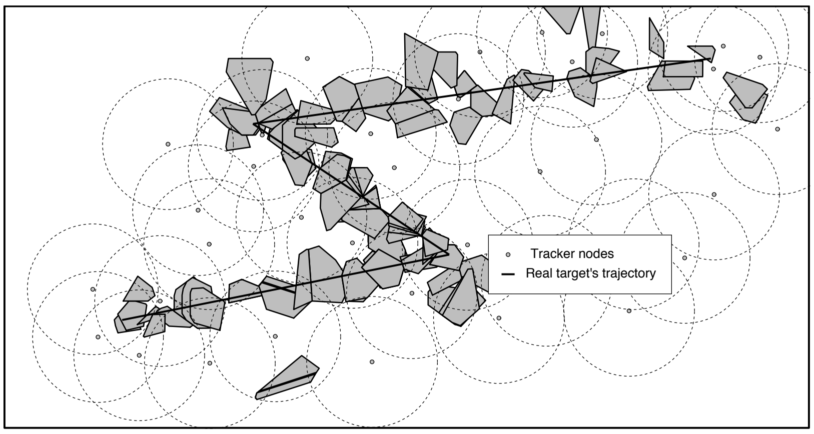


(d) Average tracking error in one-coverage detection using speed interval $[V_{min}, V_{max}]$

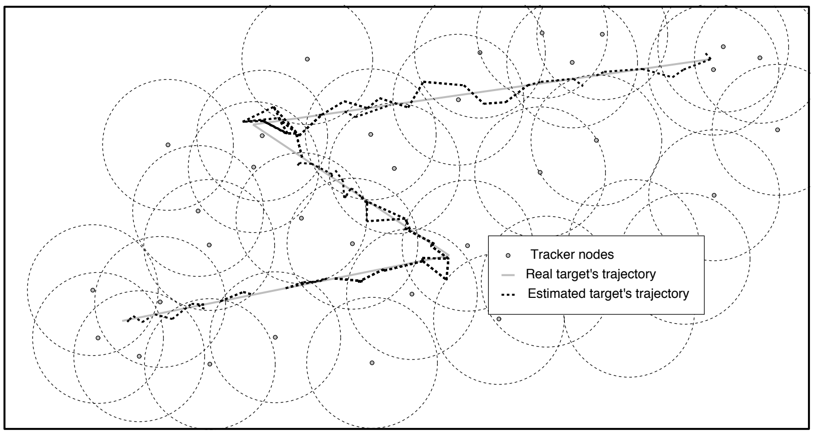
Fig. 12 Average tracking error for isolated {three,two,one}-coverage detection

virtual nodes increases in isolated two-coverage detection, which again is the result of enclosing the target node into smaller polygons.

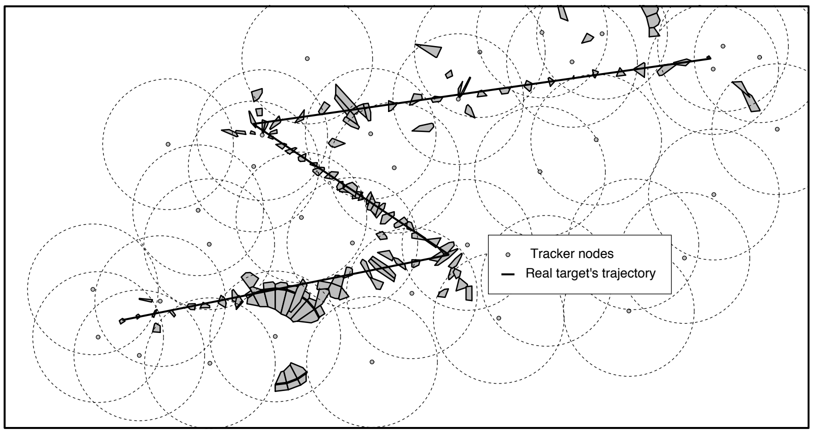
For one-coverage detection simulations we assume to know both the entry point (root point) and the exit point of graph G. Tracking errors in one-coverage detection depend on the number of virtual nodes and the knowledge of the target's speed. For cases when target's speed is known, the backtracking algorithm seeks the closest weighted edge to the target's speed for each detection along the one-coverage detection. On the other hand, when target's speed is unknown but bounded in the interval $[0, V_{max}]$, the backtracking algorithm selects all edges inside the current MSC(s) along the one-coverage detection. In both cases, we compute the mean tracking error by estimating the average Euclidean distance between the real location of the target and the center of selected polygon(s) for every detection inside the one-coverage detection. For these experiments, the tracker node is located at (603, 325) in meters, and target's speed is considered as $4m/s$. Target's trajectory uses RWP model whose entry point is located at (523, 394) and exit point is located at (673, 404), in meters. Figure 12(c) shows again how tracking errors decrease as the number of virtual nodes increases in one-coverage detection. Our simulations show that having more than 15 virtual nodes decrease tracking errors monotonically. However, the cost of computing the optimal path increases exponentially as the number of virtual nodes increases (as we showed in Section 4.2). It is important to mention that the cases with less than 15 virtual nodes per tracker node are not reported in Figure 12(c) and 12(d). This situation arises because the distance between two consecutive virtual nodes in the donut was



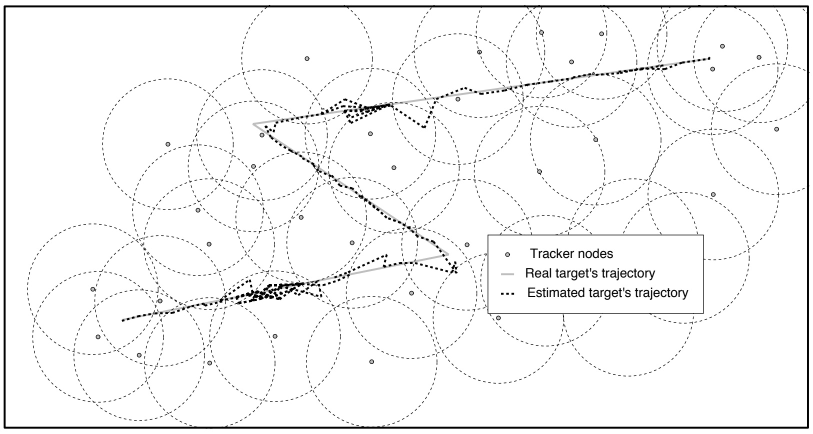
(a) Large network using 5 virtual nodes per tracker



(b) Large network using 5 virtual nodes per tracker



(c) Large network using 25 virtual nodes per tracker



(d) Large network using 25 virtual nodes per tracker

Fig. 13 Tracking in a large network scenario

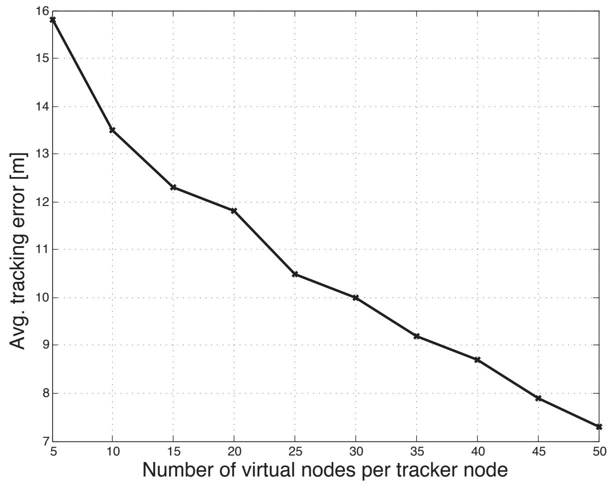


Fig. 14 Tracking error for large network scenario

greater than the MSC's diameter. As a result the MSC did not reach the center of polygons inside the *donut*, thus edges in graph G could not be connected.

Figure 12(d) shows the results of our simulations for the case when target's speed is chosen randomly in the interval $[1,\ 5] m/s$. In this figure, we can observe that having less than 15 virtual nodes per tracker node results in a disconnected graph, similar as when the target's speed is known. We can also observe in this figure that having more than 15 virtual nodes per tracker node decreases tracking error monotonically.

5.3 Tracking in a large network

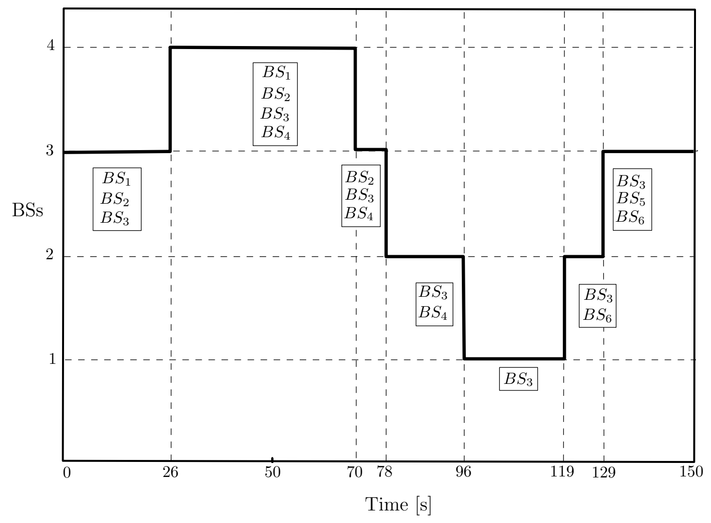
We also performed experiments on a large network with 36 tracking nodes located randomly in the monitored area using the same parameters shown in Table 1. Figure 13(a) illustrates the real target's route and computed polygons by Ghost using five virtual nodes per tracker node, while Figure 13(b) shows the real target's trajectory (solid line) vs. computed trajectory by Ghost (dashed line). In Figure 13(b) we can see graph G is disconnected in some places because too few virtual nodes were used (particularly in one-coverage detection). Figure 13(d) shows the same network scenario but now considering 25 virtual nodes per tracker node. In this figure we observe that the estimated route of the target is not disconnected, and it is similar to real target node's trajectory. In Figure 13(c) we can see, as expected, that the size of chosen polygons by Ghost are smaller compared with selected polygons in Figure 13(a). Figure 14 illustrates how tracking errors decrease as the number of virtual nodes increases on a large network scenario. It is important to mention that MSC for {one,two}-coverage detection were used in these experiments.

5.4 Ghost vs. PLT and GPLT

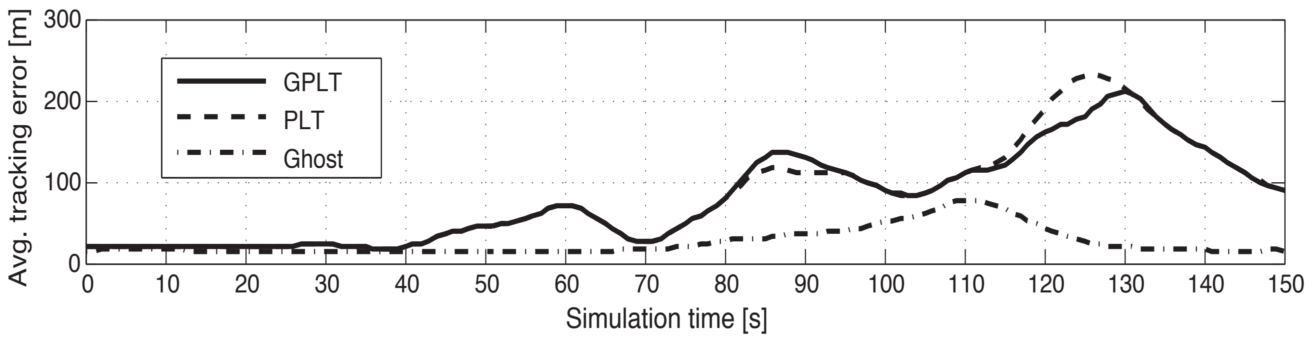
Because Ghost, GPLT and PLT algorithms [23] are focused in tracking a moving target by less than three trackers (i.e., {one,two}-coverage detection). We compared Ghost's performance with GPLT and PTL algorithms [23]. In [23] the authors proposed a scenario where the number of base stations detecting the target node changes over time. Figure 15(a) illustrates this arrangement, where BS_i denotes the *ith* base station detecting the target node at specific times. We used the same noise model parameters ($\mathcal{N}(0,100)$), the same dimensions of the monitored area, a similar target's trajectory, the same velocity model and the same transmission range as proposed in [23] to compare the mean average tracking error over time. In this experiment we used 30 virtual nodes per tracker node because our isolated simulations showed this setting provides a satisfactory performance for most scenarios. Figure 15(b) shows the mean tracking error vs. simulation time. In this figure we can see Ghost outperforms GPLT and PLT algorithms in most of the trace. This is especially clear in the interval $[78 - 129]$ seconds where only one or two BSs detect the target node. The fact Ghost optimizes searches by using the backtracking algorithm whose criterion function is the number of tracking detections during one-coverage detection results in a smaller set of possible paths. In contrast, PLT and GPLT algorithms do not apply a criterion function to reduce possible paths in {one,two}-coverage detection resulting in larger tracking errors. Moreover, as the number of virtual nodes increases, the area per polygon narrows down in Ghost, resulting in a smaller Euclidean distance between the center of selected polygons and the real location of the target.

6 Conclusions

In this paper, we propose a model to track a target node on sparse wireless networks. The proposed method is especially suited for cases when less than three tracker nodes detect the target node. We demonstrate that even for cases when only one tracker detects the target node, a set of possible paths could be constructed. Simulation results show that using dynamic Voronoi diagrams of a monitored area enhances tracking accuracy compared with other proposals found in the literature. Finally, evaluation results demonstrate that tracking a target using virtual nodes on a dynamic Voronoi framework



(a) Simulation time



(b) Tracking Performance

Fig. 15 Ghost vs. PLT and GPLT algorithms

enhances tracking accuracy by reducing the size of regions where the target node is likely located, especially for {one,two}-coverage detection.

References

1. N. Kumar. A weighted center of mass based trilateration approach for locating wireless devices in indoor environment. In Proc. of the Forth ACM International Workshop on Mobility Management & Wireless Access, Terromolinos, Spain, October 2. ACM, 2006.
2. A.P. Subramanian, P. Deshpande, J. Gaojgao, and Das S.R. Drive-by localization of roadside WiFi networks. In Proc. of INFOCOM, pages 718–725, Apr 2008.
3. M.Z.A. Bhuiyan, G. Wang, and J. Wu. Target tracking with monitor and backup sensors in wireless sensor networks. In Proc. of International Conference on Computer Communications and Networks, pages 1–6, 2009.
4. Z. Zhong and T. He. MSP: Multi-sequence positioning of wireless sensor nodes. In Proc. of International Conference on Embedded Networked Sensor Systems, Sydney, Australia, November 6-9, pages 15–28, 2007.
5. T. M. Chan. Point location in O(log n) time, Voronoi diagrams in O(n log n) time, and other transdichotomous results in computational geometry. In Proc. of Foundations of Computer Science, pages 333–344, Oct 2006.
6. P. Moravek, D. Komosny, M. Simek, M. Jelinek, D. Girbau, and A. Lazaro. Investigation of radio channel uncertainty in distance estimation in wireless sensor networks. Telecommunication Systems, pages 1–10, 2011.
7. A.Z.B. Haji Talib, M. Chen, and P. Townsend. Three major extensions to Kirkpatrick's point location algorithm. In Proc. of Computer Graphics International, pages 112–121, Jun 1996.
8. X. Wang, Z. Wang, and B. O'Dea. A TOA-based location algorithm reducing the errors due to non-line-of-sight (NLOS) propagation. IEEE Transactions on Vehicular Technology, 52(1):112–116, Jan 2003.
9. R. Yamasaki, A. Ogino, T. Tamaki, T. Uta, N. Matsuzawa, and T. Kato. TDOA location system for IEEE 802.11b WLAN. In Proc. of Wireless Communications and Networking Conference, IEEE, volume 4, pages 2338–2343, Mar 2005.
10. F. Gustafsson and F. Gunnarsson. Positioning using time-difference of arrival measurements. In Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing, Apr 2003.
11. D. Niculescu and Badri Nath. Ad hoc positioning system (APS) using AOA. In Proc. of Conference of the IEEE Computer and Communications. IEEE Societies, volume 3, pages 1734–1743, Mar 2003.

12. E. Elnahrawy, Li. Xiaoyan, and R.P. Martin. The limits of localization using signal strength: a comparative study. In Proc. of Sensor and Ad Hoc Communications and Networks, First Annual IEEE Communications Society Conference on, pages 406–414, Oct 2004.
13. T. Kitasuka, K. Hisazumi, T. Nakanishi, and A. Fukuda. Positioning technique of wireless LAN terminals using RSSI between terminals. In Proc. of the International Conference on Pervasive Systems and Computing, Las Vegas, Nevada, June 27-30, pages 47–53, 2005.
14. T. Stoyanova, F. Kerasiotis, A. S. Prayati, and G. D. Papadopoulos. Evaluation of impact factors on RSS accuracy for localization and tracking applications in sensor networks. Telecommunication Systems, 42(3-4):235–248, 2009.
15. D. Ma, M.J. Er, B. Wang, and H. B. Lim. Range-free localization based on hop-count quantization in wireless sensor networks. Telecommunication systems, pages 1–6, Jan 2009.
16. T. He, C. Huang, B.M. Blum, J.A. Stankovic, and T. Abdelzaher. Range-free localization schemes in large-scale sensor networks. In Proc. of Mobi-Com, 2003.
17. G. Han, H. Xu, T.Q. Duong, J. Jiang, and T. Hara. Localization algorithms of wireless sensor networks: a survey. Telecommunication Systems, 2011.
18. I. Guvenc, C.T. Abdallah, R. Jordan, and O. Dedeoglu. Enhancements to RSS based indoor tracking systems using Kalman filters. In Proc. of GSPx and International Signal Processing Conference, 2003.
19. E.L. Souza, E.F. Nakamura, and H.A.B. F. de Oliveira. On the performance of target tracking algorithms using actual localization systems for wireless sensor networks. In Proc. of International Symposium o Modeling Analysis and Simulation of Wireless and Mobile Systems, Tenerife, Canary Islands, Spain, October 26-19, pages 418–423, 2009.
20. C. Chao-Lin and F. Kai-Ten. Hybrid location estimation and tracking system for mobile devices. In Proc. of the Vehicular Technology Conference, IEEE 61st, volume 4, pages 2648–2652, 2005.
21. L. Liao, D. Fox, J. Hightower, H. Kautz, and D. Schulz. Voronoi tracking: Location estimation using sparse and noisy sensor data. In Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003.
22. M. Z. A. Bhuiyan, G. Wang, and J. Wu. Polygon-based tracking framework in surveillance wireless sensor networks. In Proc. of Parallel and Distributed Systems, pages 174–181, 2009.
23. T. Po-Hsuan, F. Kai-Ten, L. Yu-Chiun, and Chao-Lin C. Wireless location tracking algorithms for environments with insufficient signal sources. IEEE Transactions on Mobile Computing, 8(12):1676–1689, 2009.
24. N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low-cost outdoor localization for very small devices. IEEE Personal Communications, 7(5):28–34, Oct 2000.
25. D. Niculescu and B. Nath. DV based positioning in ad hoc networks. Telecommunication Systems, 22:267–280, 2003.
26. W. Alsalih, K. Islam, Y. Nunez-Rodriguez, and H. Xiao. Distributed Voronoi diagram computation in wireless sensor networks. In Proc. of Annual ACM Symposium on Parallel Algorithms and Architectures, Jun 2008.
27. C. Yang, P. Shi, W. Zao, L. Wang, X. Meng, and Wang J. New intersection algorithm of convex polygons based on Voronoi diagrams. In Proc. of International Symposium on Voronoi Diagrams in Science and Engineering, pages 224–231, Jul 2006.
28. A. Yershova and S.M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. IEEE Transactions on Robotics, 23(1):151–157, 2007.
29. L. Hong-Bo, L. Zhan-Shan, A. Yang, and D. Hui-Ying. On research of optimization strategy for dynamic backtracking. In Proc. of International Conference on Machine Learning and Cybernetics, volume 1, pages 266–271, Jul 2009.
30. R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. SciencieDirect, Artificial Intelligence, 41:273–312, 1990.
31. F. Benedetto, G. Giunta, A. Toscano, and L. Vegni. Dynamic LOS/NLOS statistical discrimination of wireless mobile channels. In Proc. of IEEE Vehicular Technology Conference, pages 3071–3075, Apr 2007.
32. W. Navidi and T. Camp. Stationary distributions for the random waypoint mobility model. IEEE Transactions on Mobile Computing, 3(1):99–108, 2003.