A Beacon-Less Location Discovery Scheme for Wireless Sensor Networks

Lei Fang, Wenliang Du
Department of Electrical Engineering
and Computer Science
Syracuse University
Email: {lefang,wedu}@ecs.syr.edu

Peng Ning
Department of Computer Science
North Carolina State University
Email:pning@ncsu.edu

***Abstract*— In wireless sensor networks (WSNs), sensor location plays a critical role in many applications. Having a GPS receiver on every sensor node is costly. In the past, a number of location discovery schemes have been proposed. Most of these schemes share a common feature: they use some special nodes, called beacon nodes, which are assumed to know their own locations (e.g., through GPS receivers or manual configuration). Other sensors discover their locations based on the information provided by these beacon nodes.**

In this paper, we show that efficient location discovery can be achieved in sensor networks without using beacons. We propose a beacon-less location discovery scheme. based on the following observations: in practice, it is quite common that sensors are deployed in groups, i.e., sensors are put into n groups, and sensors in the same group are deployed together at the same deployment point (the deployment point is different from the sensors' final resident location). Sensors from the same group can land in different locations, and those locations usually follow a probability distribution that can be known a priori. With this prior deployment knowledge, we show that sensors can discover their locations by observing the group memberships of its neighbors. We model the location discovery problem as a statistical estimation problem, and we use the Maximum Likelihood Estimation method to estimate the location. We have conducted experiments to evaluate our scheme.

Keyword: System Design.

I. Introduction

Sensor networks have been proposed for various applications. In many of these applications, nodes need to find their locations. For example, in rescue applications, rescue personnel can perform their tasks only if location of the hazardous event (reported by sensors) is known. Location is also important for geographic routing protocols, in which the location information (in the form of coordinates) is used to select the next forwarding host among the sender's neighbors [11]–[13], [20], [23]. Because of the constraints on sensors, finding location for sensors is a challenging problem. The location discovery problem is referred to as *localization* problem in the literature.

The Global Positioning System (GPS) [10] solves the problem of localization in outdoor environments for PC-class nodes. However, due to cost, it is highly undesirable to have a GPS receiver on every sensor node. This creates a demand for efficient and cost-effective location discovery algorithms in sensor networks. In the past several years, a number of location discovery protocols have been proposed to reduce or completely remove the dependence on GPS in wireless sensor networks [3], [6], [9], [15]–[18], [21], [22]. Most of these schemes share a common feature: they use some special nodes, called beacon nodes, which are assumed to know their own locations (e.g., through GPS receivers or manual configuration). Other sensors discover their locations based on the information provided by these beacon nodes.

Although the overall cost of beacon-based location discovery schemes is significantly less than the GPS-like schemes, the cost for each beacon node is still expensive. To have a more robust and accurate positioning system, the number of beacon nodes tend to increase. Therefore, it is appealing to achieve location discovery without using beacon nodes.

In general, a positioning system consists of two components: one is the reference points, whose coordinates are known; the other is the spatial relationship between sensors and the reference points. For example, in Global Positioning System, the satellites are the reference points, and the time of arrival reveals the relationship between a GPS receiver and the satellites. In beacon-based positioning system, beacons are reference points, and relationships between a sensor and the reference points include time of arrival, time difference of arrival, angle of arrival, received signal strength, and hop-based distance, etc. For a positioning system that does not use beacon nodes, we still need to find some type of reference points with which sensors can find their locations.

We have observed that when sensors are deployed, the coordinates of the deployment points are usually known. Let us look at a deployment method that uses an airplane to deploy sensor nodes. The sensors are first pre-arranged in a sequence of smaller groups. These groups are dropped out of the airplane sequentially as the plane flies forward. This is analogous to parachuting troops or dropping cargo in a sequence. The positions where each sensor group are dropped out of the airplane are referred to as *deployment points*; their coordinates can be pre-determined and stored in sensors' memories prior to the deployment. Then during the deployment, using the GPS receivers on the airplane, we can ensure that the actual deployment points are the same as the pre-determined coordinates. We will use these deployment

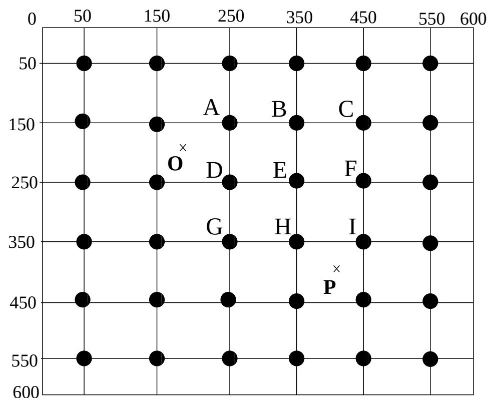


Fig. 1. An Example of Group-based Deployment (each dot represents a deployment point).

points as the reference points.

Next, we need to find a way to allow each sensor to establish a spatial relationship with the reference points, so that sensors can use this relationship (along with the coordinates of the reference points) to find their own locations. We have observed the following facts: after the deployment, sensors usually do not land in locations that are uniformly random across the whole deployment area, they tend to be distributed in areas around their deployment points. Therefore, sensors in different locations will observe different types of neighbors (i.e. neighbors from different groups). For example, assume that the deployment points are arranged in a grid style depicted in Figure 1, and a group of nodes are deployed at each deployment point. After the deployment, a node at location O will find out that more of its neighbors are from group A and D than from group H and I; on the contrary, a node at location P has more neighbors from group H and I than from A and D. This means, knowing how many of its neighbors are from each deployment group, a sensor can derive its spatial relationship with the deployment points.

To derive such a spatial relationship, we need to have a prior knowledge about how sensors from each group are distributed after the deployment, i.e., how likely can they land in a location z meters away from the their deployment points? In practice, given the methods and the conditions of the deployment, such knowledge can usually be modeled using a probability distribution function (pdf).

Based on the prior knowledge about the deployment points and the pdf of the deployment, we propose a beacon-less location discovery scheme, KPS (deployment Knowledge-based Positioning System). In our scheme, each sensor first finds out the number of its neighbors from each group. We call this the *observation* of a sensor. With this observation, a sensor estimates a location based on the principle that the estimated location should maximize the probability of the observation. This is exactly the principle of the *maximum likelihood estimation (MLE)*. Therefore, we use the MLE method to conduct the location estimation. Our results have shown that KPS can achieve a decent accuracy.

The rest of the paper is organized as follows: the next section overviews the existing work on location discovery. Section III presents the modeling of deployment knowledge. Section IV describes our beacon-less scheme. Section V presents the evaluation results. Section VI compares the beaconless scheme with the existing localization schemes. Finally we conclude and lay out some future work in Section VII.

II. Related Work

In the past several years, a number of location discovery protocols have been proposed to reduce or completely remove the dependence on GPS in wireless sensor networks [1], [3], [4], [6], [8], [9], [15]–[19], [21], [22].

Most solutions for location discovery in sensor networks require a few nodes called beacons (they are also called anchors or reference points), which already know their absolute locations via GPS or manual configuration. The density of the anchors depends on the characteristics and probably the budget of the network since GPS is a costly solution. Anchors are typically equipped with high-power transmitters to broadcast their location beacons. The remainder of the nodes then compute their own locations from the knowledge of the known locations and the communication links. Based on the type of knowledge used in location discovery, localization schemes are divided into two classes: range-based schemes and range-free schemes.

Range-based protocols use absolute point-to-point distance or angle information to calculate location between neighboring sensors. Common techniques for distance/angle estimation include Time of Arrival (TOA) [10], Time Difference of Arrival (TDOA) [1], [8], [19], Angle of Arrival (AOA) [17], and Received Signal Strength (RSS) [1]. While producing fine-grained locations, range-based protocols remain cost-ineffective due to the cost of hardware for radio, sound, or video signals, as well as the strict requirements on time synchronization and energy consumption.

Alternatively, coarse-grained range-free protocols are cost-effective because no distance/angle measurement among nodes is involved. In such schemes, errors can be masked by fault tolerance of the network, redundancy computation, and aggregation [9]. A simple algorithm proposed in [3] and [4] computes location as the centroid of its proximate anchor nodes. It induces low overhead, but high inaccuracy as compared to others. An alternate solution, DV-Hop [18], extends the single-hop broadcast to multiple-hop flooding, so that sensors can find their distance from the anchors in terms of hop counts. Using the information about the average distance per hop, sensors can estimate their distance from the anchors. Amorphous positioning scheme [15] adopts a similar strategy as DV-Hop; the major difference is that Amorphous improves location estimates using offline hop-distance estimations through neighbor information exchange.

Another existing range-free scheme is APIT algorithm [9]. APIT resolves the localization problem by isolating the environment into triangular regions between anchor nodes. A node uses the point-in-triangle test to determine its relative location with triangles formed by anchors and thus narrows down the area in which it probably resides. APIT defines the center of gravity of the intersection of all triangles that a node resides in as the estimated node location.

Our proposed scheme is significantly different from the existing schemes. The major advantage of our scheme is the removal of the dependency on the expensive beacon (or anchor) nodes. However, our scheme does not intend to replace the existing beacon-based schemes, because there are situations when the accurate deployment knowledge is difficult to obtain prior to deployment. If indeed the deployment knowledge can be obtained, our scheme can substantially reduce the cost associated with the expensive beacon nodes.

Rao et al. also proposed a localization scheme without beacons [20]. In this scheme, nodes flood the network to discover the distance (hops) between perimeter nodes. Compared to this flooding scheme, our scheme is more efficient in communications, because in our scheme, nodes only need to communicate with their neighbors once.

III. Modeling of the Deployment Knowledge

We assume that sensor nodes are static once they are deployed. We define *deployment point* as the point location where a sensor is to be deployed. This is not the location where this sensor finally resides. The sensor node can reside at points around this deployment point according to a certain probability distribution. As an example, let us consider the case where sensors are deployed from a helicopter. The deployment point is the location of the helicopter. We also define *resident point* as the point location where a sensor finally resides.

A. Group-based Deployment Model

In practice, it is quite common that nodes are deployed in groups, i.e., a group of sensors are deployed at a single deployment point, and the probability distribution functions of the final resident points of all the sensors from the same group are the same.

In this work, we assume such a group-based deployment, and we model the deployment knowledge in the following (we call this model the *group-based deployment model*):

1) N sensor nodes to be deployed are divided into n equal size groups so that each group, G_i, for $i = 1, \ldots, n$ is deployed from the deployment point with index i. To simplify the notion, we also use G_i to represent the corresponding deployment point, and let (x_i, y_i) represent its coordinates.
2) Locations of the deployment points are pre-determined prior to deployment. Their coordinates are stored in each sensor's memory. The deployment points can form any arbitrary pattern. For example, they can be arranged in a square grid pattern (see Figure 1), a hexagonal grid pattern, or other irregular patterns.
3) During deployment, the resident point of a node k in group G_i follows a probability distribution function $f_k^i(x, y \mid k \in G_i) = f(x - x_i, y - y_i)$. An example of the pdf $f(x, y)$ is a two-dimensional Gaussian distribution.

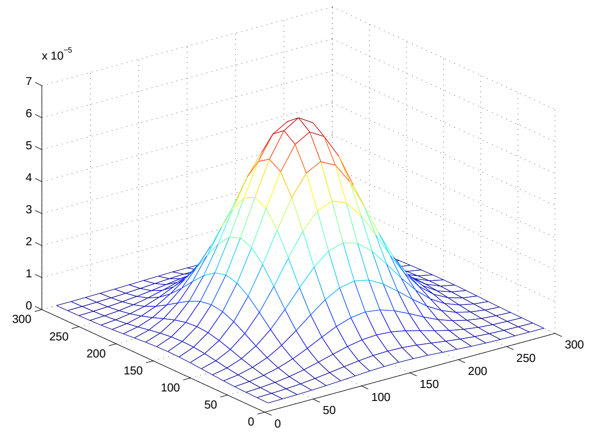


Fig. 2. Deployment distribution for one group.

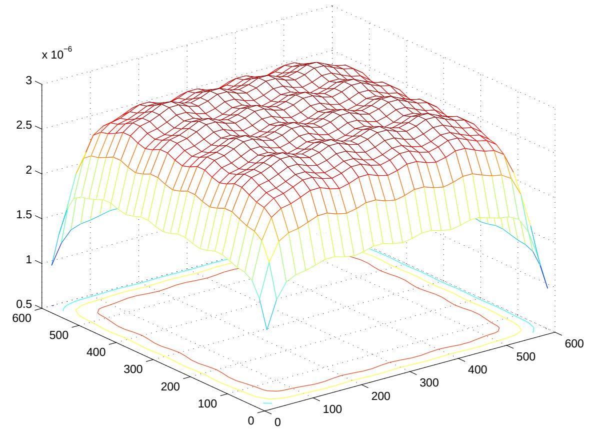


Fig. 3. The overall deployment distribution over the entire region.

Figure 2 shows an example of two-dimensional Gaussian distribution at the deployment point $(150, 150)$.

B. Deployment Distribution

There are many different ways to deploy sensor networks, for example, sensors could be deployed using an airborne vehicle. The actual model for deployment distribution depends on the deployment method.

In this paper, we model the sensor deployment distribution as a Gaussian distribution (also called Normal distribution). Gaussian distribution is widely studied and used in practice. Although we only employ the Gaussian distribution in this paper, our methodology can also be applied to other distributions.

We assume that the deployment distribution for any node k in group G_i follows a two-dimensional Gaussian distribution, which is centered at the deployment point (x_i, y_i). Namely, the mean of the Gaussian distribution μ equals (x_i, y_i), and the pdf for node k in group G_i is the following [14]:

$$\begin{aligned} f_k^i(x, y \mid k \in G_i) &= \frac{1}{2\pi\sigma^2} e^{-[(x-x_i)^2 + (y-y_i)^2]/2\sigma^2} \\ &= f(x - x_i, y - y_i), \end{aligned}$$

where σ is the standard deviation, and $f(x,y) = \frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/2\sigma^2}$. Without loss of generality, we assume that the pdf for each group is identical, so we use $f_k(x,y \mid k \in G_i)$ instead of $f_k^i(x,y \mid k \in G_i)$ throughout this paper.

Although the distribution function for each single group is not uniform, we still want the sensor nodes to be evenly deployed throughout the entire region. By choosing a proper distance between the neighboring deployment points with respect to the value of σ in the pdf, the probability of finding a node in each small region can be made approximately equal. Assuming that a sensor node is selected to be in a given group with an equal probability, the average deployment distribution (pdf) of any sensor node over the entire region is:

$$f_{overall}(x,y) = \frac{1}{n}\sum_{i=1}^{n} f_k(x,y \mid k \in G_i). \tag{1}$$

To see the overall distribution of sensor nodes over the entire deployment region, we have plotted $f_{overall}$ in Eq. (1) for $6 \times 6 = 36$ groups over a $600m \times 600m$ square region with the deployment points $2\sigma = 100m$ apart (assuming $\sigma = 50$). We use the grid strategy to arrange the deployment points as depicted in Figure 1. Figure 3 shows the overall distribution. From Figure 3, we can see that the distribution is almost flat (i.e. nodes are fairly evenly distributed) in the whole region except near the boundaries.

IV. A Beacon-Less Location Discovery Scheme

After sensors are deployed, each sensor broadcasts its group id to its neighbors, and each sensor can count the number of neighbors from G_i, for $i = 1,\ldots,n$. Assume that a sensor finds out that it has $a_1,\ldots,a_n$ neighbors from group $G_1,\ldots,G_n$, respectively. The question is whether this information, along with the deployment knowledge, can help the sensor estimate its own location.

Intuitively speaking, the observation of the neighbors' group ids is helpful. For example, if a sensor sees many of its neighbors from group G_j but zero neighbors from group G_k, we will know that the sensor is close to the deployment point of G_j, and it is far away from the deployment point of G_k. However, we need a systematic method to use this neighborhood information to calculate the sensor's location directly.

Assume that the location of the sensor of concern is $\theta = (x,y)$. Given the number (m) of nodes deployed in each group and the pdf function of the deployment, we can compute the probability that $a_1,\ldots,a_n$ nodes (from group $G_1,\ldots,G_n$, respectively) can be observed by a node at the location θ. Let X_i be the random variable that represents the number of nodes from group G_i that are neighbors to the node at location θ. Let $\boldsymbol{a} = (a_1,\ldots,a_n)$ be a vector representing the observation. The probability that $\boldsymbol{a}$ is observed by a node at θ is the following:

$$f_n(\boldsymbol{a} \mid \theta) = \Pr\left(X_1 = a_1,\ldots,X_n = a_n \mid \theta\right).$$

Note that, given θ, all X_i are mutually independent. Therefore,

$$\begin{aligned} & f_n(\boldsymbol{a} \mid \theta) \\ = \;& \Pr(X_1 = a_1 \mid \theta)\cdots\Pr(X_n = a_n \mid \theta). \end{aligned} \tag{2}$$

The above probability indicates how likely it is to observe $X_1 = a_1,\ldots,X_n = a_n$ at location θ. The function $f_n(\boldsymbol{a} \mid \theta)$ describes the joint pdf for every observed vector $\boldsymbol{a} = (a_1,\ldots,a_n)$ in the sample. When $f_n(\boldsymbol{a} \mid \theta)$ is regarded as a function of θ for a given vector $\boldsymbol{a}$, in statistics, it is called the *likelihood function*.

The goal of the location discovery now becomes an estimation problem, namely, we need to select the parameter θ from the parameter space Ω. We should certainly not consider any value of $\theta \in \Omega$ for which it would be impossible to obtain the vector $\boldsymbol{a}$ that was actually observed. Instead it would be natural to try to find a value of θ for which the probability density $f_n(\boldsymbol{a} \mid \theta)$ is large, and to use this value as an estimate of θ. For each possible observed vector $\boldsymbol{a}$, we are led by this reasoning to consider a value of θ for which the likelihood function $f_n(\boldsymbol{a} \mid \theta)$ is a maximum and to use this value as an estimate of θ. This is the concept of *maximum likelihood estimation* (abbreviated as MLE).

The method of MLE was introduced by R. A. Fisher in 1912, it is by far the most widely used method of estimation in statistics. The principle of MLE is simple. That is to find the parameter values that make the observed data most likely. In other words, MLE is a method by which the probability distribution that makes the observed data most likely is sought. Details of MLE can be found in most of the statistics textbooks [5].

Let us see how to compute the likelihood function $f_n(\boldsymbol{a} \mid \theta)$ when the vector $\boldsymbol{a}$ is observed. Let $g_i(\theta)$ be the probability that a sensor from group G_i can land within the neighborhood of point θ (we will show how to compute $g_i(\theta)$ later in this section). Therefore, the probability that exactly a_i sensors are within the neighborhood of point θ is the following (where m is the number of sensors deployed at each deployment point):

$$f(X_i = a_i \mid \theta) = \binom{m}{a_i}\left(g_i(\theta)\right)^{a_i}\left(1 - g_i(\theta)\right)^{(m-a_i)}$$

Therefore, according to Equation (2), the likelihood function $f_n(\boldsymbol{a} \mid \theta)$ can be computed using the following equation:

$$\begin{aligned} f_n(\boldsymbol{a} \mid \theta) &= \prod_{i=1}^{n} f(X_i = a_i \mid \theta) \\ &= \prod_{i=1}^{n}\binom{m}{a_i}\left(g_i(\theta)\right)^{a_i}\left(1 - g_i(\theta)\right)^{(m-a_i)}. \end{aligned}$$

The value of θ that maximizes the likelihood function $f_n(G \mid \theta)$ will be the same as the value of θ that maximizes $\log f_n(G \mid \theta)$, because logarithm is an increasing function. Therefore, it will be more convenient to determine the MLE by finding the value of θ that maximizes

$$\begin{aligned} L(\theta) &= \log f_n(G \mid \theta) \\ &= \sum_{i=1}^{n} \log \binom{m}{a_i} + \sum_{i=1}^{n} a_i \log g_i(\theta)) \\ &\quad + \sum_{i=1}^{n} (m - a_i) \log(1 - g_i(\theta)). \end{aligned} \tag{3}$$

There are various ways to find the θ that maximizes $L(\theta)$. When $L(\theta)$ is differentiable and the maximal exists, it must satisfy the following partial differential equations known as the *likelihood equations*:

$$\frac{\partial L(\theta)}{\partial x} = 0 \quad \text{and} \quad \frac{\partial L(\theta)}{\partial y} = 0.$$

This is because the definition of maximum or minimum of a continuous differentiable function implies that its first derivatives vanish at such points.

If the first derivative has a simple analytic form, we can solve the above likelihood equations to find the value for $\theta = (x, y)$. However in practice, often we cannot derive an equation with a simple analytic form for its first derivative. This is especially likely if the model is complex and involves many parameters and/or complex probability functions. As we will show later when we describe how to compute $g_i(\theta)$, $L(\theta)$ is indeed very complicated. In such situations, the MLE estimate must be sought numerically. We will describe several numerical methods in the next subsection.

A. Finding Maximum

1) Gradient Descent: Gradient descent [7], also known as the method of steepest descent, is a common method in numerical analysis. The key idea of gradient descent is to find the maximum of a function based on the information of its gradient. Intuitively, we can imagine that a two-dimensional function is represented as a surface in a three-dimensional space, and the maximum point (also called peak) holds a zero gradient. The goal of the gradient descent method is to find a shortest path to reach the peak from a selected starting point. Usually the path consists of many iteration steps, and at each step, the choice of the direction is where the function increases most quickly. The whole process is like hill climbing, and the goal is to reach the top of the hill using the minimal amount of steps. To reduce the computation cost, numerous optimization schemes have been proposed to find a shortest path to the maximum point. One method is the conjugate gradient method [2], which usually converges faster to the maximum point than the gradient descent method. These optimizations are beyond the scope of this paper. In our work, we only focus on the most basic gradient descent method.

The gradient descent method, if used improperly, can be computationally intensive, and thus not suitable for resource-constrained sensor nodes. The cost of the gradient descent method in our scheme can be significantly affected by the selection of the starting point and the computation of the likelihood function $L(\theta)$ and its first derivatives. In section IV-C and IV-D, we will show how to simplify $L(\theta)$ and its first derivatives using approximation and table-lookup approaches.

In the next subsections, we describe two algorithms that achieve a much better efficiency, however, at the cost of the accuracy. These two algorithms can be used as a stand-alone approach to estimate the location when the accuracy requirement is not high. Moreover, they can also be used to find the starting point for the gradient descent method.

2) A Geometric Approach: From a geometric perspective, if a sensor can get its distance from at least three deployment points, it can calculate its position. We will give a much simplified scheme to estimate a sensor's distance from a deployment point. Assume that the sensor has observed a_i neighbors from the deployment group G_i, it can use the MLE to find the distance z, such that the probability to observe a_i neighbors from group G_i is maximized.

We use $L_i(\theta)$ to represent the log likelihood function, where θ represents the location of the sensor. Based on Equation (3), we have

$$\begin{aligned} L_i(\theta) &= \log f(X_i = a_i \mid \theta) \\ &= \log \binom{m}{a_i} + a_i \log g_i(\theta) \\ &\quad + (m - a_i) \log(1 - g_i(\theta)). \end{aligned}$$

Let us use z to represent the distance from θ to the deployment point of G_i. Let $g(z)$ represent the probability that a sensor from group G_i can land within a circle (with radius R), the center of which is z distance from the deployment point of G_i. Because $g(z) = g_i(\theta)$, we can use z to replace θ in the above equation:

$$\begin{aligned} L_i(z) &= \log \binom{m}{a_i} + a_i \log g(z) \\ &\quad + (m - a_i) \log(1 - g(z)). \end{aligned}$$

To find z, such that $L_i(z)$ is maximized, we let the first derivate of $L_i(z)$ be zero:

$$\frac{dL_i(z)}{dz} = 0$$

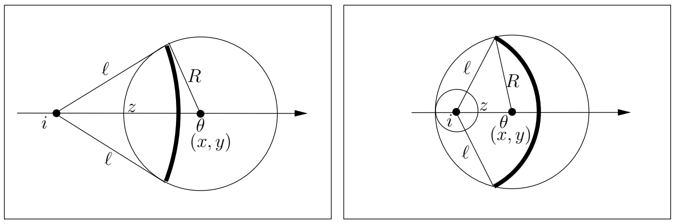
Therefore, we get the following result:

$$g(z) = \frac{a_i}{m}.$$

When $g(z)$ is complicated, we can use the table-lookup approach to find z given a_i and m, namely, we pre-calculate $g(z)$ for various values of z, and store the table of results in sensor's memory. Once $\frac{a_i}{m}$ is known after the deployment, a sensor can find z by looking up the value of z from the table. If the accuracy requirement on z is not so high, the amount of memory needed for such a table is not so large.

As we will see from the next subsection, using regression, we can approximate $g(z)$ using a Gaussian distribution. Therefore, finding z from a_i and m is quite simple.

Once we get the distance of the sensor from three deployment points, we can find the location of the sensor using the coordinates of these deployment points. The computation of this scheme is quite efficient.



(a) i is outside of the circle ($z > R$) (b) i is inside of the circle ($z < R$)

Fig. 4. Probability of nodes residing within a circle.

3) Small area search approach: In the above geometric approach, because we only considered three deployment points, the accuracy might not be desirable, especially when the number of the neighbors from the other deployment points is not negligible. That is, the location we find might not be the maximum point of the likelihood function $L(\theta)$. To improve the accuracy, we use the point (X_0, Y_0) found by the geometric approach as an initial point, and then conduct a search in the nearby locations to find the maximum of $L(\theta)$, i.e., the value of $L(\theta)$ will be computed on the following points:

$$\begin{aligned} X &= X_0 + i * LEN \quad -RG \le i \le RG \\ Y &= Y_0 + j * LEN \quad -RG \le j \le RG, \end{aligned}$$

where LEN is the length of each step (e.g. we can set it to 2 to 5 meters), and RG determines the search range. We use the number of steps along each direction to represent the range.

We will pick the point the has the maximum of $L(\theta)$ as the node's estimated location. The computational cost of the search depends on the number of steps and the step length. This approach will bring better result than the simple geometric scheme at the cost of computations. A performance comparison will be given later in Section V.

B. Computing $g_i(\theta)$

We use z to represent the distance from point θ to the deployment point of group G_i. We define Ψ as the set of all deployment groups in the KPS scheme. We draw two circles. The first circle has a radius ℓ, and is centered at i, the deployment point of group G_i. We call this circle the i-circle. The second circle has a radius R, and is centered at $\theta = (x, y)$. We call this circle the θ-circle. When two circles intersect, we call the i-circle's arc within the θ-circle the L_{arc}, and we use $L_{arc}(\ell, z, R)$ to represent the length of the arc. We now consider an infinitesimal ring area $L_{arc}(\ell, z, R) \cdot d\ell$. The bold areas in Figure 4.a and 4.b show the infinitesimal ring areas.

Based on the two-dimensional Gaussian distribution, the probability that a node n_i from group $i \in \Psi$ with deployment point (x_i, y_i) resides within this small ring area is

$$\begin{aligned} & \frac{1}{2\pi\sigma^2} e^{-\frac{\ell^2}{2\sigma^2}} \cdot L_{arc}(\ell, z, R) \cdot d\ell \\ = \quad & f_R(\ell \mid n_i \in G_i) \cdot L_{arc}(\ell, z, R) \cdot d\ell, \end{aligned}$$

where $f_R(\ell \mid n_i \in G_i)$ is defined as the following Gaussian distribution:

$$f_R(\ell \mid n_i \in G_i) = \frac{1}{2\pi\sigma^2} e^{-\frac{\ell^2}{2\sigma^2}}.$$

Using geometry knowledge, it is not difficult to derive the following equation for $L_{arc}(\ell, z, R)$:

$$L_{arc}(\ell, z, R) = 2\ell \cos^{-1}\left(\frac{\ell^2 + z^2 - R^2}{2\ell z}\right).$$

We define $g(z \mid n_i \in G_i)$ as the probability that the sensor node n_i from group i resides within the θ-circle, where z is the distance between θ and the deployment point of group G_i.

To calculate $g_i(z \mid n_i \in G_i)$, we integrate the probabilities over all the ring areas (for different ℓ) within the θ-circle. Therefore, when $z > R$ (as shown in Figure 4.a),

$$\begin{aligned} & g(z \mid n_i \in G_i) \\ = & \int_{z-R}^{z+R} f_R(\ell \mid n_i \in G_i) \cdot L_{arc}(\ell, z, R)\, d\ell. \end{aligned}$$

When $z < R$ (as shown in Figure 4.b),

$$\begin{aligned} & g(z \mid n_i \in G_i) \\ = & \int_0^{R-z} \ell \cdot 2\pi f_R(\ell)\, d\ell \\ + & \int_{R-z}^{z+R} f_R(\ell \mid n_i \in G_i) \cdot L_{arc}(\ell, z, R)\, d\ell. \end{aligned}$$

Putting both $z > R$ and $z < R$ cases together, we have the following:

$$\begin{aligned} & g(z \mid n_i \in G_i) \\ = \quad & \mathbf{1}\{z < R\} \left[1 - e^{-\frac{(R-z)^2}{2\sigma^2}}\right] \\ + & \int_{|z-R|}^{z+R} f_R(\ell \mid n_i \in G_i) \cdot L_{arc}(\ell, z, R)\, d\ell, \end{aligned} \tag{4}$$

where $\mathbf{1}\{\cdot\}$ is the set indicator function[1].

Therefore, $g_i(\theta)$, the probability that a node from the deployment group G_i can land within the neighborhood of point θ, can be computed in the following:

$$g_i(\theta) = g(\sqrt{(x - x_i)^2 + (y - y_i)^2} \mid n_i \in G_i).$$

For the sake of simplicity, we use $g(z)$ to represent $g(z \mid n_i \in G_i)$ in the rest of this paper, when it is obvious to see from the context that we are referring to the nodes in group G_i.

The formula for $g(z)$ is quite complicated, and we cannot afford to compute it using Equation (4) in sensor networks. Simplifying the analytical representation of $g(z)$, if possible, is difficult and beyond the scope of this paper. In this paper, we propose two approaches to improve the computations. The first is the table-lookup approach, and the second is the regression approach.

[1]The value of $\mathbf{1}\{\cdot\}$ is 1 when the evaluated condition is true, 0 otherwise.

C. *Simplifying $g(z)$: Table-lookup Approach*

Since $g(z)$ only depends on R and σ, which are known prior to the deployment, we can pre-calculate $g(z)$ offline for each z value, and store the results as a table in sensor's memories. When a sensor needs the result for a specific value, e.g., z_0, it can use z_0 as the index to look up the value of $g(z_0)$ from the table. The computation takes only constant time.

Although the range of z is from 0 to $+\infty$, the values of $g(z)$ beyond certain range is negligible (our analysis shows that $g(z)$ is an exponentially decreasing function). Let α represent the size of the range, in which $g(z)$ has non-negligible values. We divide this range into ω equal-size sub-ranges, and store the $\omega + 1$ dividing points into a table. When a sensor needs to compute $g(z_0)$, it first finds the sub-range that contains z_0 by looking up the table; then it treats the two end-points of the sub-range as the two ends of a straight line, and finds the value corresponding to z_0 on that line. The sensor uses this value for $g(z_0)$.

As we can see that the precision of this approach depends on the size of the sub-range, the smaller the size is, the better. However, smaller sub-range also means more memory is needed for the whole table. Assume each value of $g(z)$ can be represented by two bytes, then we need 2000 bytes of memory to store the table if we divide the range into 1000 pieces. In fact, in our experiments, when the range is divided into 200 pieces (i.e., using 400 bytes of memory), the accuracy is almost not affected.

Note that $g(z)$ does not depend on the deployment points; therefore, as long as the deployment follows the same p.d.f., the same $g(z)$ table can be used, regardless of how the deployment points are arranged.

D. *Simplifying $g(z)$: Regression Approach*

In the regression approach, we want to find a much simple representation for $g(z)$. Such representation does not need to produce the exact same values as the original $g(z)$, as long as it is a reasonable approximation.

After plotting $g(z)$, we have observed that the shape of $g(z)$ is very much like a Gaussian distribution with mean zero. Therefore we use the following Gaussian distribution to conduct the regression (the Guaissian distribution is adjusted by multiplying πR^2):

$$g(z) = (\frac{1}{2\pi\Omega^2}e^{-z^2/2\Omega^2}) \cdot \pi R^2.$$

The goal of the regression is to find out the standard deviation Ω of the regressed Gaussian distribution, such that the error between $g(z)$ and the regressed distribution function is minimized. We get the following relationships:

$$\Omega = 6.328\frac{R^2}{\sigma^2} + \sigma$$

For example, when $R = 40$, $\sigma = 50$, the value of $\Omega = 54.05$. We plot both $g(z)$ and our regression result in Figure 5. The results show that the regression is very accurate for $R = 40$ and $\sigma = 50$. We also plot the mean difference between the original $g(z)$ values and the regression results for various values of R (Figure 6). The figure shows that when R is not too large, the regression is quite accurate.

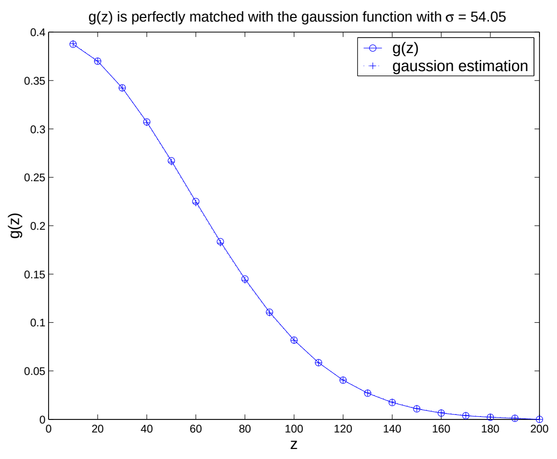


Fig. 5. Gaussian function with Ω=54.05 really matches the g(z)

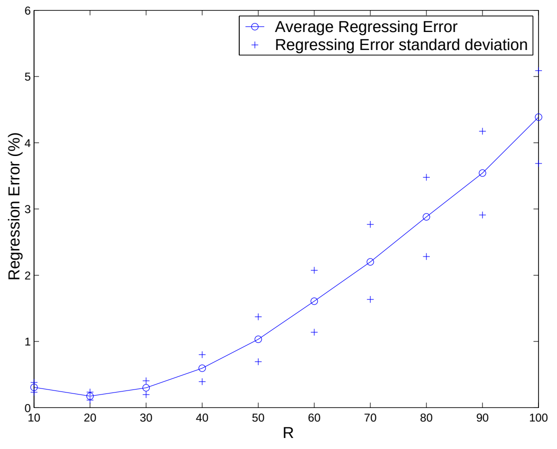


Fig. 6. Regression Errors

The above simplification can significantly reduce the costs for computing $L(\theta)$; however, being able to compute $L(\theta)$ efficiently is not sufficient. If the gradient descent method is to be used to find the maximum of $L(\theta)$, we should also be able to compute the first derivative of $L(\theta)$ efficiently.[2] Let $z^2 = (x - x_i)^2 + (y - y_i)^2$, where (x_i, y_i) is the deployment point of group G_i. The first derivative on x, $\frac{\partial L(\theta)}{\partial x}$, can be derived in the following (the first derivative on y can be similarly derived):

$$\frac{\partial L(\theta)}{\partial x} = \sum_{i=1}^{n} \frac{a_i \frac{\partial g_i(\theta)}{\partial x}}{g_i(\theta)} - \sum_{i=1}^{n} \frac{(m - a_i)\frac{\partial g_i(\theta)}{\partial x}}{1 - g_i(\theta)},$$

[2]Although we can approximately calculate the first derivative of $g(z)$ at point z_0 by using $\frac{g(z_1)-g(z_0)}{z_1-z_0}$, where z_1 is another point close to z_0, the computation is less accurate than the direct calculation.

where $\frac{\partial g_i(\theta)}{\partial x}$ can be calculated in the following:

$$\begin{aligned}\frac{\partial g_i(\theta)}{\partial x} &= \frac{R^2}{-2\Omega^4}e^{-((x-x_i)^2+(y-y_i)^2)/2\Omega^2}(x-x_i)\\ &= g_i(\theta)\frac{-1}{\Omega^2}(x-x_i).\end{aligned}$$

Combining the above two equations together (and also applying the similar method to y), we get the following:

$$\begin{aligned}\frac{\partial L(\theta)}{\partial x} &= \frac{-1}{\Omega^2}\sum_{i=1}^{n}\frac{a_i - mg_i(\theta)}{1-g_i(\theta)}(x-x_i),\\ \frac{\partial L(\theta)}{\partial y} &= \frac{-1}{\Omega^2}\sum_{i=1}^{n}\frac{a_i - mg_i(\theta)}{1-g_i(\theta)}(y-y_i),\end{aligned}$$

Therefore, once we know how to compute $g(z)$, we can also compute the first derivative of $L(\theta)$. To further improve the performance, we can use the table-lookup approach to store the table of the Gaussian distribution into sensor's memory. However, our experiments show only 10% of the performance improvement. This is because the computation on $g(z)$ is not the major cost.

V. Evaluation

This section provides a detailed quantitative analysis evaluating the performance of our beacon-less location discovery scheme. The obvious metric for the evaluation is the location estimation error. We have conducted a variety of experiments to cover different system configurations including varying the node density and varying the transmission range. We have also investigated how the boundary effects affect the accuracy of the location estimation. Moreover, we have compared the performance of the three approaches described in Section V-D.

In our experiments, the deployment area is a square plane of 1000 meters by 1000 meters. In this paper, we only use the square grid pattern for our deployment: namely, the plane is divided into 10×10 grids of size $100m \times 100m$; centers of these grids are chosen as deployment points. Figure 1 shows our deployment strategy. Similar experiments can be conducted for other deployment patterns.

We still use m to represent the number of nodes in each group, R to represent the transmission range. We set the σ of the Gaussian distribution to 50 in all of the experiments. We then randomly generate the sensor networks based on the deployment model.

In the experiments we calculate ΔZ, the average distance between a node's actual position and estimated position. We use ΔZ as the average estimation error of KPS. In our simulation, we estimate the locations for all the nodes in the plane, and then we calculate the average errors. The numerical approach used in all the experiments is gradient descent unless it says otherwise.

A. Estimation Error when Varying Node Density

Because KPS is based on statistical methods, the sample size is critical to the accuracy of the estimation. In KPS, the sample size is decided by the density of the network, which is equivalent to m, the number of nodes deployed in each group (because we have fixed the deployment area and the number of deployment groups). Therefore, in this experiment, we investigate how the estimation error changes when m changes.

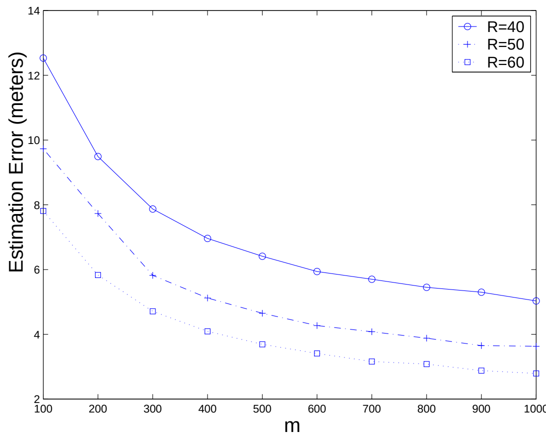


Fig. 7. Estimation errors vs. m ($R = 40$, 50, and 60).

For each experiment, we fix $\sigma = 50$, and then change m from 100 to 1000. We repeat the same experiment for $R = 40$, 50, and 60. The simulation results are depicted in Figure 7. The figure shows that our estimation is quite accurate. For example, when $m = 400$ and $R = 40$, the estimation error is only 8 meters, which equals $0.2R$. The figure also shows that the accuracy of the location estimation becomes better when m increases, i.e., each sensor can observe more nodes in its neighborhood.

In practice, if we do not have enough sensors to deploy to reach the desired node density, we can still achieve the desired density by deploying dummy nodes along with the sensor nodes. A dummy node is a low-cost node, whose only functionality is to broadcast its group identity to its neighbors. A dummy node does not need to find its own location, nor does it need to carry out sensing or computing tasks. Its only goal is to increase the sample size, such that the sensors in its neighborhood can estimate their location more accurately. Therefore, the cost of a dummy node can be much lower than a sensor node.

B. Estimation Error when Varying Transmission Range

Another way to increase the sample size is to increase the transmission range R. When R increases, the number of neighbors for each sensor will increase. In this experiment, we investigate how R affects the estimation accuracy. We fix $m = 400$, and vary R from 40 meters to 120 meters. The simulation results are depicted in Figure 8.

The figure shows an interesting trend: when R increases from 40 to 90, the estimation error decreases without a surprise. However, starting from $R = 90$, the estimation error increases. This can be intuitively explained using an extreme-case example: assume that $R = \infty$, which means that all the sensor nodes can observe exactly the same set of

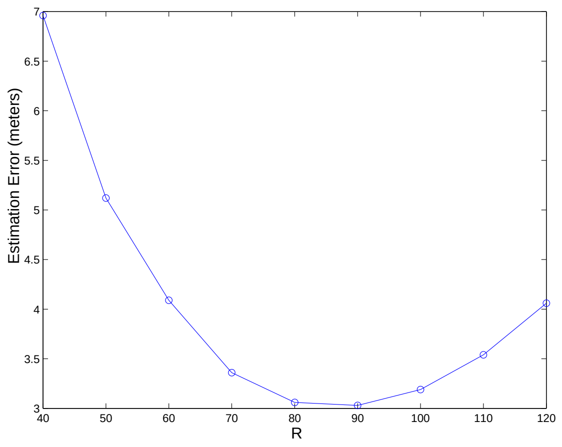


Fig. 8. Estimation errors vs. R ($m = 400$).

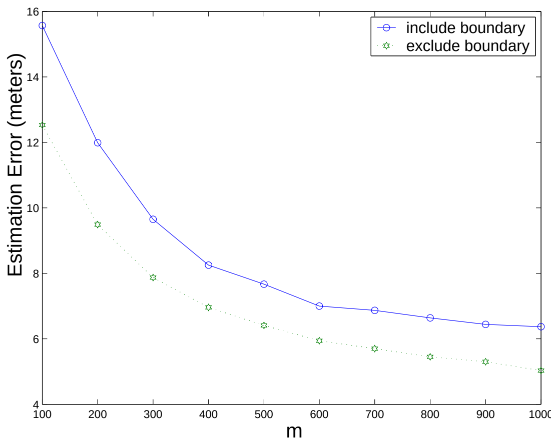


Fig. 9. Boundary ($R = 40$ and $\sigma = 50$).

neighbors (i.e., all the other nodes in the network). Therefore, the estimated location for all the sensors will be the same (actually it will be the center of the deployment area).

The extreme-case example indicates that when R increases to infinite, the estimation error will increase, and eventually will converge to a constant, when the center of the deployment area is selected as every sensor's estimated location. This is largely due to the boundary effects that we are going to discuss in the next experiment. Namely, when R increases, more and more nodes will be affected by the boundary effects because their neighboring areas cover the areas outside of the deployment area, where the node density is close to zero. Therefore, for those nodes, the difference of their observations becomes smaller and smaller while the difference of their locations is still constant. We will further investigate the boundary effects in our experiments.

The fact that Figure 8 has a minimum point tells us we should choose the proper R in practice; just increasing R won't always give us better results.

C. *Estimation Error vs. Boundary Effects*

Boundary is also a factor we must consider. Because there are less nodes on the boundaries, the variance of a node's neighbors is large compared to the nodes near the center. So it's less accurate for nodes to determine their positions with their observations.

In the experiment, we calculate the errors in two different ways: one includes the nodes on the boundary, and the other does not. The boundary nodes are defined as those that are within 50 meters of any of the four borders (50 is chosen because $\sigma = 50$). The results are shown in Figure 9. It is clear that nodes deployed near the boundary will make the estimation error larger.

D. *Comparison of Three Find-Maximum Methods*

As we have discussed in Section IV-A, we propose to use three different approaches to find the maximum point of the likelihood function. Among these three approaches, gradient descent can provide the best accuracy, but its computation cost is most expensive. The geometric approach is the least expensive one, but it produces the worst estimation error. The small-area-search approach is in the middle. In this experiment, we quantitatively compare the computation cost and the accuracy of these three approaches.

1) Computation Cost: The algorithms for the three approaches are tested on a PC with Intel P4 2.8G hz CPU and 1G memory. We set $R = 40$ and $m = 100$. We measure the average time for a sensor to find its location. Their performance comparisons are shown in Table I.

TABLE I
COMPARISON OF THE THREE NUMERICAL APPROACHES.

Algorithm	Computation Expense
Geometric Method	0.02ms
Small Area Search (1 step)	0.05ms
Small Area Search (2 steps)	0.14ms
Small Area Search (3 steps)	0.26ms
Small Area Search (4 steps)	0.32ms
Gradient Decent	0.68ms

The relative comparison among these algorithms is more important than their absolute values. We can find that the computational cost of the gradient descent is 34 times more expensive than the geometric method. Given the fact that the geometric method is very simple (its cost is almost negligible), and the location discovery is only conducted once, the gradient descent method is also affordable for sensor networks. Also as we mentioned before, implementing the optimization technologies such as table-lookup in the sensor system will make it more realistic to use the gradient descent algorithm.

2) Estimation Accuracy: From Figure 10, we see that the gradient descent approach supplies the best results and the geometric approach produces the worst results. The small-area-seach scheme becomes more and more accurate when the number of steps increases. The figure shows that the accuracy of the 2-step method is already close to the gradient decent method.

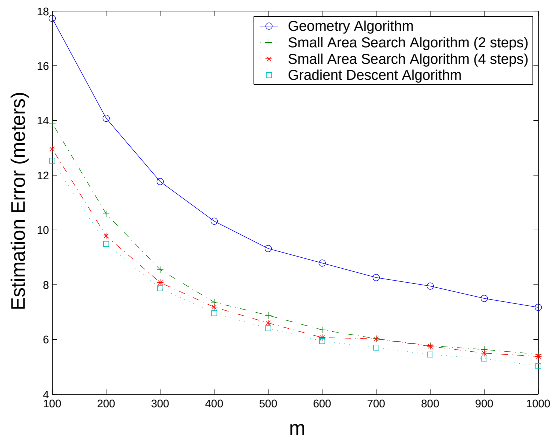


Fig. 10. Comparison of different numerical approach ($R = 40$).

VI. Comparisons with existing schemes

In this section, we compare our KPS scheme with the existing beacon-based location schemes. Because KPS and its assumptions are significantly different from the existing beacon-based schemes, comparing the localization accuracy is not much meaningful. Therefore our comparisons mainly focus on cost, robustness, security, and mobility.

A. Cost analysis

1) Communication Cost: Communication cost is a major concern in sensor networks. For the beacon-based localization schemes, communication cost is very low, because sensor nodes only need to receive signals from the beacon nodes; there is no interaction between sensor nodes. The KPS scheme depends on the knowledge of neighbors, it requires each node to broadcast (only one-hop broadcasting) a message to its neighbors. However, this broadcasting is necessary for neighbor discovering that is required by other functionalities of sensor networks, such as routing. Therefore, the KPS localization scheme does not introduce extra communication cost.

2) Computation and Storage Cost: Compared with the beacon-based location scheme, the calculation of KPS is more complex, so the computation cost of KPS is much higher than most of the beacon-based localization schemes. Most computation burden comes from the Find-Maximum methods. However, our simulation results have shown that the computation cost is still realistic.

As we mentioned in IV-C, to reduce the computation cost, we can store some pre-calculated table in sensor's memory. The size of the table can be limited to several kilobytes.

3) Device Cost: The cost of device on beacon-based schemes is much higher than the KPS scheme. KPS is sensitive to node density. If the node density of the sensor networks is high enough, no extra device is needed; if the node density is too low, cheap dummy nodes can be deployed to help achieve acceptable localization accuracy. However, beacon-based schemes must depend on special beacon nodes, which are much more expensive than normal sensor nodes.

B. Robustness and Security

In the beacon-based schemes, the localization accuracy largely depends on a small number of beacon nodes. When some of these nodes fail to function or when they are tampered with by adversaries (for example, some compromised beacon nodes might report false positions), a significant number of sensors can be affected, i.e., their derived locations can be much far away from their actual locations.

In contrast, the beacon-less KPS scheme is much more robust and secure. In KPS, each sensor depends on its neighbors to find its own location. When one or a few neighbors fail, the localization results will not be affected much. When some compromised neighbors intentionally send out false group memberships, their lies cannot be arbitrary, because a lie that deviates too much from the deployment knowledge can reveal anomalies. Therefore, the KPS scheme can even tolerate node compromise to certain degree. Further analysis regarding this property is undergoing.

C. Limitation of the KPS Scheme

Although KPS achieves localization without using expensive beacon nodes, it does have its limitations. First, Beacon-based schemes support mobile sensor networks. Namely nodes can obtain their locations even if they are mobile. However, KPS depends on the distribution of the node deployment; once a node moves, the distribution cannot be maintained. Therefore, KPS can only be used in a static sensor networks. Second, locations of deployment points are critical. They must be estimated with high accuracy. Although this can be easily achieved for an airborne deployment because GPS can be used on an airplane, the goal is hard to achieve for other types of deployment. In addition, KPS also requires an accurate modeling of deployment knowledge. In our future work, we will study the accuracy of localization if the actual deployment deviates from the model. Due to these limitations, we do not claim that KPS can replace the existing beacon-based localization schemes in all applications. Our KPS scheme provides an less-expensive alternative in those applications that satisfy our assumptions. We believe such assumptions are reasonable in many sensor network applications.

D. Summaries

The comparisons of the KPS scheme and beacon-based localization schemes are summarized in Table II

TABLE II
Comparison of KPS and beacon-based schemes.

	KPS	beacon-based
Communication overhead	Low	Low
Computation cost	High	Low
Device cost	Low	High
Robustness/Security	High	Low
Mobility	None	Good

VII. Conclusion and Future Work

In sensor networks, traditional localization schemes use beacons as the reference points to help sensors find their locations. We present KPS, a beacon-less localization scheme, in which sensors use the deployment distribution and the position of deployment points to find the locations. The major advantage of the KPS scheme is that we do not need the expensive beacon nodes, while achieving comparable location discovery results. We have conducted extensive evaluation. Our results show that when the node density is high, the location estimation error achieved by KPS can be less than a few meters. These results show that the accuracy provided by KPS is sufficient to support various applications in sensor networks. In our future work, we plan to study how the inaccuracy of the deployment knowledge can affect the accuracy of the location discovery. Our motivation is that in practice, the deployment knowledge that we know prior to the deployment might not be quite accurate. It will be interesting to know how KPS is affected by that. We also plan to provide more analytical evaluation results on KPS.

VIII. Acknowledgment

The authors acknowledge supports from the United States National Science Foundation IIS-0219560, IIS-0312366, and CNS-0430252. The authors would also like to thank all the anonymous reviewers for their valuable comments.

References

[1] P. Bahl and V. N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *Proceedings of the IEEE INFOCOM*, pages 775–784, March 2000.

[2] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[3] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low cost outdoor localization for very small devices. In *IEEE Personal Communications Magazine*, pages 28–34, October 2000.

[4] N. Bulusu, J. Heidemann, and D. Estrin. Density adaptive algorithms for beacon placement, April 2001.

[5] M. H. DeGroot and M. J. Schervish. *Probability and Statistics*. Addison Wesley, 3rd edition, 2002. Chapter 6.

[6] L. Doherty, K. S. Pister, and L. E. Ghaoui. Convex optimization methods for sensor node position estimation. In *Proceedings of INFOCOM'01*, 2001.

[7] R. Hamming. *Numerical Methods for Scientists and Engineers*. Dover Pubns, 2nd edition, 1987.

[8] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. In *Proceedings of MOBICOM'99*, Seattle, Washington, 1999.

[9] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. F. Abdelzaher. Range-free localization schemes in large scale sensor networks. In *Proceedings of the Ninth Annual International Conference on Mobile Computing and Networking (MobiCom '03)*, 2003.

[10] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins. *Global Positioning System: Theory and Practice*. Springer Verlag, 4th ed., 1997.

[11] X. Hong, K. Xu, and M. Gerla. Scalable routing protocols for mobile ad hoc networks. *IEEE Network magazine*, (4), 2002.

[12] B. Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of ACM MobiCom 2000*, 2000.

[13] Y. B. Ko and N.H. Vaidya. Location-aided routing (lar) in mobile ad hoc networks. In *Proceedings ACM/IEEE MOBICOM 98)*, pages 66–75, October 1998.

[14] A. Leon-Garcia. *Probability and Random Processes for Electrical Engineering*. Reading, MA: Addison-Wesley Publishing Company, Inc., second edition, 1994.

[15] R. Nagpal, H. Shrobe, and J. Bachrach. Organizing a global coordinate system from local information on an ad hoc sensor network. In *IPSN'03*, 2003.

[16] A. Nasipuri and K. Li. A directionality based location discovery scheme for wireless sensor networks. In *Proceedings of ACM WSNA'02*, September 2002.

[17] D. Niculescu and B. Nath. Ad hoc positioning system (APS) using AoA. In *Proceedings of IEEE INFOCOM 2003*, pages 1734–1743, April 2003.

[18] D. Niculescu and B. Nath. Dv based positioning in ad hoc networks. In *Journal of Telecommunication Systems*, 2003.

[19] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proceedings of MOBICOM*, Seattle, Washington'00, August 2000.

[20] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *Proceedings of ACM MOBICOM 2003*, pages 96–108, September 2003.

[21] A. Savvides, C. Han, and M. Srivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Proceedings of ACM MobiCom '01*, pages 166–179, July 2001.

[22] A. Savvides, H. Park, and M. Srivastava. The bits and flops of the n-hop multilateration primitive for node localization problems. In *Proceedings of ACM WSNA '02*, September 2002.

[23] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of ACM MobiCom 2000*, Rome, Italy, July 2001.