Spatial Ordering Derivation for One-dimensional Wireless Sensor Networks

Xiaojun Zhu and Guihai Chen
State Key Laboratory for Novel Software Technology
Nanjing University, Nanjing, P. R. China
Email: gxjzhu@gmail.com, gchen@nju.edu.cn

***Abstract*—Sensors can be deployed along a road or bridge to form a one-dimensional wireless sensor network where nodes have a spatial ordering. This spatial characteristic indicates relative locations for nodes and facilitates functions like object tracking and monitoring in the network. We target at deriving this ordering without attaching extra hardware to sensor nodes. In this work, we propose to exploit RSSI (received signal strength indicator) to generate the spatial ordering. The approach is based on our observation: *the largest RSSI value indicates the shortest distance*. Evaluations on two datasets show that our algorithm generates the spatial ordering at high success rate.**

***Keywords*-wireless sensor networks; one-dimension; linear network; spatial ordering;**

I. Introduction

Localization has long been recognized as an important component in wireless sensor networks, since most of the sensing tasks require the knowledge of position [1], [2]. Though much effort has been devoted to find the locations of sensor nodes in two dimensional scenarios [2], [3], [4], there are one-dimensional networks in reality, such as the Golden Gate Bridge application [5]. Sensors in road networks can be considered as in one dimension too. In these networks, nodes form an ordering spatially. The ordering is important for event detection and object tracking [6]. Deriving the ordering is our target.

The task is not trivial. One approach is to label the nodes while deploying them. In practice, however, nodes might be replaced or reprogrammed in the long run, and it is inefficient to find out the ordering manually every time. We should note that a node is able to change its ID during execution. This operation is supported by TinyOS [7]. Finally, there may be adversaries maliciously exchanging the positions of some nodes. Therefore, we need a method that can automatically find the ordering.

It is inappropriate to use conventional localization methods either. These methods are optimized in terms of absolute error, *i.e.*, the distance between a node's actual position and its estimated position. However, low absolute error does not necessarily guarantee reliable orderings, especially in cases where some nodes are very close. These methods are not designed specifically for deriving orderings.

In this paper, we base our approach on the *best signal observation* (short for BESO hereafter): in a given direction, the node closest to the sender will observe the highest RSSI. This observation is different from the *better signal observation*(short for BETO) in [3], [8], which says that the receiver nearer to the sender will observe higher RSSI. Section III will discuss their differences in detail. In presence of noise, we find that the former is more robust than the latter.

We take the observation as a criterion of correct node ordering. A naive approach is to perform brute-force search over all permutations. With exponential time complexity, this approach is impractical. We show that the search space can be reduced to the set of all nodes.

Our contributions can be enumerated as follows.

- To the best of our knowledge, we are the first to propose and advocate the use of BESO, which tolerates more noise than conventionally used BETO.
- To derive node ordering, we propose an $O(nm)$ algorithm, where n is the number of nodes and m is the number of edges. Instead of accepting all inputs, we allow the algorithm to reject some inputs to obtain high correct ratio. Evaluations on two datasets show that our approach is highly reliable.

The rest of the paper is organized as follows. Section II summarizes related work. Section III presents observation and basic idea. Section IV gives the method. Section V discusses related issues of the method. We evaluate the method in Section VI and conclude the paper in Section VII.

II. Related Work

Our work is closely related to localization in wireless sensor networks. Localization has long been a research issue in ubiquitous computing, see [2] for a survey. The key difference is that we focus on node ordering rather than absolute physical position.

A. One-dimensional Localization

Bischoff *et al.* [9] give a range-free localization method for one-dimensional sensor networks. The novelty of their work is that it has provable performance guarantee. They do not try to give an ordering of sensor nodes. Their method is *connective-based*, which assumes that two nodes are within a given distance if and only if they can hear from each other. Lotker *et al.* [10] give *range free ranking* of nodes in one-dimensional networks. In their approach, the two end nodes are given as anchor nodes. One of them initiates flooding a message and the other is responsible for terminating the flooding. During the flooding, each node computes its *rank* in the spatial ordering. The correctness analysis is based on connectivity assumption. They try to solve the same problem as us, except that our approach is not connectivity-based and we do not need anchor nodes either.

Connectivity information can be easily obtained without extra hardware. However, it can not distinguish between nodes with the same neighbors. Additionally, nodes are assumed to have uniform transmission range, which can be impractical. These methods have inherent limitations to give the spatial ordering of sensor nodes.

B. *Relative RSSI and Near-far Relationship*

We have to use more than connectivity information to get node ordering, and we find RSSI, which can be obtained from most platforms without extra hardware. Traditionally, absolute values are used to estimate physical distance, which has poor accuracy. Recently, some researchers use RSSI to indicate near-far relationship, and find it works well for localization problems [3], [4], [8]. These methods actually use BETO. In [4], each node compares its RSSI value with other anchors' to determine which one is nearer to the sender. Based on the information, the node can be located in intersection of different rings. Zhong *et al.* [8] propose to use *signature distance*. Each node has a signature, a neighbor list ordered by RSSI. The distance between two nodes are estimated as the distance between their signatures. Guo *et al.* [3] use a mobile beacon to locate sensor nodes. When the beacon travels, it broadcasts messages containing its current location. Each node listens, and keeps recording the message together with the RSSI. It will observe a maxima in the RSSI curve when the beacon is at the nearest position. Due to elementary geometry, the node must be in the line through the position and perpendicular to the beacon's trajectory. The assumption used is actually BETO.

Inspired by their work, we empirically studied BETO in [11]. The conclusion we made is that BETO does hold among near neighbors, but does not among far-away neighbors. However, it is difficult to classify neighbors into "near" and "far" before the ordering is known. We finally drop the BETO approach, and extract a more reliable observation: BESO, which is reported in this paper. It states less about signal propagation, can tolerate more noise, and is easy to use. The differences between BESO and BETO will be further discussed in the next section.

III. Observation and Basic Idea

For convenience, we use the terms "ordering", "sequence" and "list" interchangeably in the rest of the paper.

A. *RSSI Ordering vs. Distance Ordering*

We use a dataset from literature [8], where 54 nodes are placed in a line. For each node, when it broadcasts a message, its neighbors record the RSSI values during receiving. We divide its neighbors into two groups according to their physical locations: left side neighbors and right side neighbors. For each group, we sort the nodes by RSSI values. The sorted list is named *RSSI ordering*. For a network of 54 nodes, we have $2 + 2 \times 52 = 106$ different such orderings. Then, we check each ordering to see whether it reveals the true distance relationship.

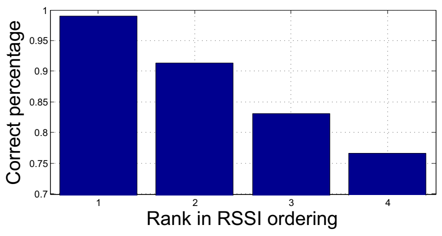


Figure 1. Compare RSSI orderings with distance orderings

Figure 1 shows that over 99% RSSI orderings find the nearest neighbor (rank 1) correctly. The percentage decreases as the increase of rank. Less than 80% RSSI orderings tell correctly about the 4-th nearest node to the sender.

In other words, it is highly probable that in a certain direction from the sender, the nearest neighbor observes the best signal, which we term as BESO. On the other hand, when nodes are far away from the sender, it is less probable to infer their near-far relationship by comparing RSSI values, which shows that BETO is less probable to hold.

B. *BESO vs. BETO*

In fact, the key difference is that the BESO states a partial order of neighbors' RSSI, while BETO states a total order. Consider the linear network 1-2-3-4-5 and suppose nodes can hear from each other directly. Let $r(a, b)$ be the RSSI of signals sent from a to b. When node 2 is a sender, BESO only states that $r(2, 3) > r(2, 4)$ and $r(2, 3) > r(2, 5)$. BETO, on the other hand, will require a total order among $r(2, 1)$, $r(2, 3)$, $r(2, 4)$ and $r(2, 5)$ to reflect the distance relationship.

One can find that the correctness of BESO is *logically implied* by the correctness of BETO: if BETO is true, then BESO is. The reverse is not necessarily true. It is easy to incorporate BETO into our approach, but we find that it does not work well because of noise.

C. *Basic Idea and Challenges*

We can easily collect RSSI information by any data collection protocol, such as CTP [12]. The specific process is out of our concern. After data are collected, we sort node i's neighbors in descending order of their RSSI of signals sent from i. The sorted list is called the *neighbor list* of the sender, denoted as α_i . We use *ordering graph* to denote the set of neighbor lists. Figure 2 shows a one-hop linear network and one possible ordering graph. Ordering graph is the input of our problem.

Note that neighbor list contains all neighbors of a node, different from the RSSI orderings we mentioned above, where each RSSI ordering only contains neighbors on the same direction(there are two directions). Since we do not have information about nodes' relative locations, which is exactly what we want to derive, we cannot split a neighbor list into two sublists.

However, given a node ordering, we can check whether BESO holds, *i.e.*, we can *verify* whether an ordering of nodes is *consistent* with the ordering graph under BESO. If the answer is no, then we claim that the ordering is not correct. Consider the example in Figure 3 with two candidate orderings, 1-2-3-4 and 2-1-4-3. For 1-2-3-4, we can check that BESO holds. For 2-1-4-3, node 4 is the nearest right side neighbor of node 1, which contradicts α_1 under BESO. So it is not the actual ordering.

There are two challenges to apply the idea. First, we can not afford to check all permutations. We tackle this challenge in the next section by verifying end nodes, the two nodes located at the edge of the network field. Second, no ordering or multiple orderings may pass the verification. For the case of no ordering, consider an extreme case where all neighbor lists are empty. Each possible nearest side neighbor is certainly not contained in any list, so no ordering can pass the verification. For the case of multiple orderings, we can consider the example in Figure 4 and find that the given ordering graph is consistent with two orderings simultaneously. In both cases, we conclude the presence of too much noise and then output an error message. Some ordering graphs will yield no ordering, and we discuss this issue in Section V-A.

IV. Ordering Derivation

In this section, we determine whether an ordering graph is consistent with a single ordering. If it is, then find the ordering. Note that two orderings are considered as the same if one is the inversion of the other, because we only care about the relative locations of sensor nodes. For example, the ordering 1-2-3-4 is the same with 4-3-2-1.

The key idea is behind the notion of *end nodes*, the two nodes that located at the edge of the network field. We do not know which one is an end node previously.

A. Verification of End Node

Instead of verifying whether a given permutation is consistent with the ordering graph, we try to verify whether a node could be at the end of a permutation consistent with the ordering graph. The search space of the latter is reduced to the set of node IDs.

If a node is an end node, then all its neighbors are in the same direction. According to BESO, the nearest node should observe the highest RSSI, thus lie at the first place of the neighbor list. Remove the end node from the ordering graph and consider the remaining graph. The nearest node we found must be an end node in the new network. Thus we reach a subproblem. Repeat the process, and we will eventually remove all nodes — unless the original node is not an end node. We describe the process in Algorithm 1, where we slightly change the time of removing node.

Algorithm 1: verify_end_node

```
Input: $(\mathbb{G} = (V, A), e)$, $\mathbb{G}$ is an ordering graph and $e$
       is a candidate end node
Output: $S$, the sequence in which nodes are removed
1  begin
2      $S \longleftarrow \{e\}$;
3      $p \longleftarrow e$;
4      while $| S | < | V |$ do
5          $list \longleftarrow \alpha_p \backslash S$;
6          if $list \neq \emptyset$ then
7              $p \longleftarrow list.firstElement()$;
8              $S.add(p)$;
9          else
10             $S \longleftarrow \emptyset$;
11             Terminate abnormally;
```

Now consider the sequence in which nodes are removed. We care about whether it is consistent with the ordering graph under BESO. Actually, the removing process has already verified it in one direction, and we only need to check it in the other direction. We can simply run Algorithm 1 again with the other end of the obtained sequence, and get a new sequence. Then check whether the two sequences are the same.

An Example Consider the ordering graph in Figure 3. Suppose we want to verify whether node 1 can be an end node. We first observe its neighbor list α_1, from which we pick the first node, node 2. Removing node 1 from the graph, and use node 2 as the new end node. We will find node 3, and then node 4. The sequence is 1-2-3-4. Now we can check whether this sequence is consistent with the ordering graph. By checking, we treat node 4 as an end node, and run the removing process as before. Then we get the sequence 4-3-2-1. So we conclude that node 1 can be an end node, and the corresponding ordering is 1-2-3-4 (or 4-3-2-1).

Suppose we want to verify whether node 2 can be an end node. We will first get the sequence 2-1-3-4. Running the process again with node 4, we get a different sequence 4-3-2-1. We conclude that node 2 can not be an end node.

B. The General Algorithm

The discussion above actually implies the important fact: if two permutations have non-disjoint $\{first\_node, last\_node\}$, then at most one of them can

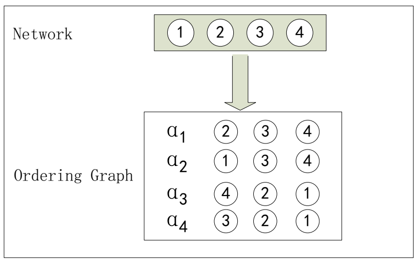


Figure 2. A one-hop linear network and its possible ordering graph.

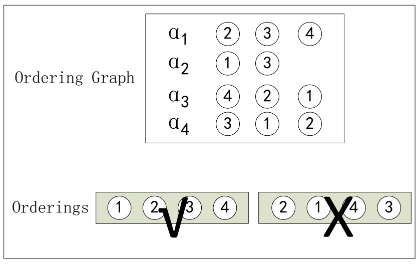


Figure 3. The ordering graph is consistent with 1-2-3-4, not 2-1-4-3.

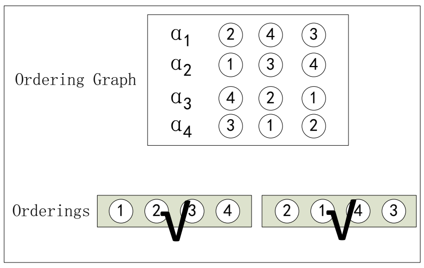


Figure 4. The ordering graph is consistent with two orderings.

pass the BESO verification. The reason is as follows. If there is a common node, then we can only derive one ordering beginning with it.

We iteratively check each node to see whether it can be an end node. If it can, then we find the ordering by Algorithm 1. In cases where we find no ordering or multiple orderings, we claim an error. The algorithm is presented in Algorithm 2. Lines 6 $\sim$ 10 test whether a node can be an end node. Line 11 tests the case of multiple orderings.

Algorithm 2: ordering_derivation

```
Input: $\mathbb{G} = (V, A)$, $\mathbb{G}$ is an ordering graph
Output: $S$, node ordering
1  begin
2      $S \longleftarrow \emptyset$;
3      $candidates \longleftarrow V$;
4      while $candidates \neq \emptyset$ do
5          Pick and remove a node $v$ from $candidates$;
6          $S_1 \longleftarrow verify\_end\_node(\mathbb{G}, v)$;
7          if $S_1 \neq \emptyset$ then
8              $v \longleftarrow S_1.lastNode()$;
9              $S_2 \longleftarrow verify\_end\_node(\mathbb{G}, v)$;
10             if $S_2$ is reversion of $S_1$ then
11                 if $S \neq \emptyset$ then
12                     $S \longleftarrow \emptyset$;
13                     return;
14                 else
15                     $S \longleftarrow S_1$;
16                     remove $v$ from $candidates$;
```

The time complexity of Algorithm 2 is in $O(nm)$ where n is the number of nodes and m is the number of edges. To see this, note that Algorithm 1 runs in $O(m)$ time in the worst case and Algorithm 2 calls it at most $O(n)$ times.

V. Discussions

A. Sample-based RSSI Data Collection

We consider the case where some ordering graphs yield no solution. Instead of using a single ordering graph, we use several ordering graphs without incurring much overhead.

Implicitly or explicitly, we usually average several RSSI readings to reduce noise (*e.g.*, [8]). The drawback is that we do not know how many readings should be averaged before noise is significantly reduced. Additionally, some large noise is possible to "pollute" many clean readings.

Instead, we use raw readings to obtain ordering graphs. We can obtain an ordering graph from a single turn of broadcast. Specifically, after each node has broadcast a message, we get a *sample* of the signal condition of the network. Then the sample can be used to generate an ordering graph. We illustrate a sample in Figure 5(a) and its corresponding ordering graph in Figure 5(b). The underlying philosophy is that we can rely on the built-in noise filter in our approach to eliminate noise.

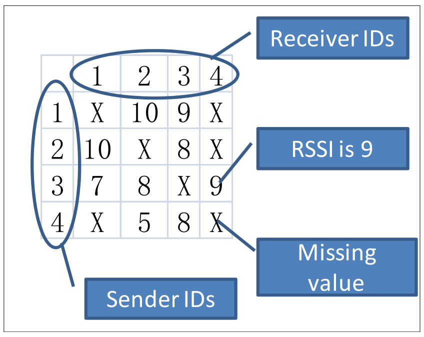
If traditional methods average 10 sets of readings to eliminate noise and get 1 ordering graph, we get 10 ordering graphs. The benefit is that our approach can filter out very noisy samples better than the averaging operation. More importantly, we can dynamically control the collection process until we obtain an ordering. The overhead depends on the channel condition. We can suspend the collection process if too many (by setting a threshold) samples are noisy to avoid unlimited overhead. Since channel condition is changing over time, we can resume the process periodically.

In practice, nodes can obtain RSSI by overhearing normal transmissions, rather than specifically broadcasting null messages. The overhead is further reduced.

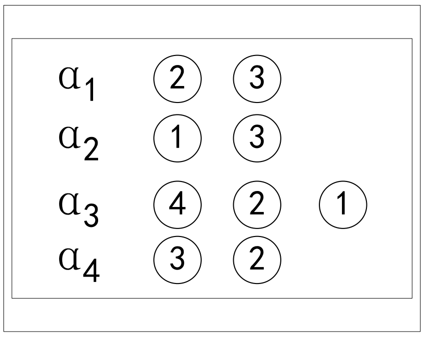
B. Centralized vs. Distributed

The method above is centralized: data are collected to a central station, which then carries out computations to derive the ordering. Here we consider a distributed version.

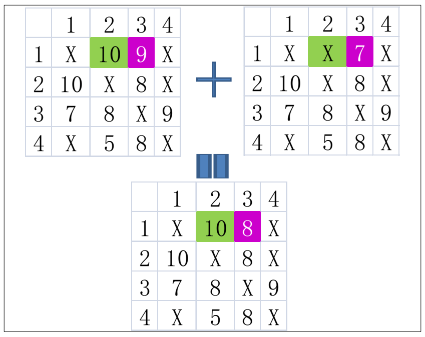
Suppose end nodes are given as anchors like in [10], then the protocol can be as follows. Let one of the end node have rank 1, and make it broadcast a message. At the same time, make the receivers acknowledge that message along with the RSSI readings. By our BESO observation, the end node can now determine which node has rank 2. Then the node with rank 2 can find out the node with rank 3. The process is repeated until some node has rank n where n is the number



(a) A sample



(b) Its ordering graph



(c) Smoothing two samples

Figure 5. Samples and smoothing.

of nodes. The last node must be the other end node. This is a distributed version of Algorithm 1.

In the presence of noise, we add a "roll back" operation. For example, if we have determined that the node sequence 3-1-4 and then node 4 finds node 2 is the nearest neighbor on the right side, then we should conclude that the sequence should be 3-1-4-2. However, node 2 may disagree. It may find that node 4 is not the nearest on its left side, which can happen due to package loss. In this situation, we cannot continue the process because BESO does not hold. We roll back the process and make the furthest reachable neighbor of node 2 to determine its next neighbor again. This modification corresponds to the "double check" operation in Algorithm 2 (Lines 6,9,10).

If end nodes are unknown, then we should try each one, as Algorithm 2 does. Only if some node acts as an end node and the above process terminates with all nodes having ranks, we can finish the process.

The overhead depends on the channel condition. If there is no package loss, then node i should broadcast and receive N_i messages in the end-node-known situation, and nN_i messages in the end-node-unknown situation in the worst case where N_i is the number of neighbors of node i and n is the total number of nodes. Therefore, we may prefer a centralized approach if end nodes are unknown, and a distributed approach if end nodes are given.

VI. Evaluations

A. Setup

Datasets We use two datasets. One is collected by ourselves from a network consisting of 7 Iris nodes in a volleyball yard. Nodes are spaced from 0.75 to 1.0 meter and they can hear from each other. It is a small network, but provides insight into the challenges resulted from noise and interference. The other dataset is from a published work [8], where the network is multi-hop and consists of 54 MicaZ motes(As stated in the dataset specification, several motes have no readings recorded, so we actually have data from 48 motes). Each dataset is a collection of *samples* we mentioned before. The one-hop dataset contains 960 samples and the multi-hop dataset contains 200 samples.

We compare our method under different *smoothing* numbers, which is the number of consecutive samples we smooth. To illustrate the smooth operation, we give an example in Figure 5(c). We have mentioned before that conventional method for eliminating noise is to average over all samples. Its smoothing number is equal to the total number of samples.

Metrics We use two metrics for evaluating an algorithm: 1) *Correct ratio*, the number of samples yielding the correct ordering / the number of samples yielding an ordering; 2) *Accept ratio*, the number of samples yielding an ordering / the number of samples. We want both ratios to be high. High correct ratio means high confidential results. On the other hand, high accept ratio means a few samples the algorithm needs, resulting in small overhead.

B. Results

Figure 6 shows that accept ratio increases with the increase of smoothing number for both datasets. However, the improvement increases slowly when smoothing number is above 2, which implies that it is unnecessary to smooth all readings. With smoothing number set as 2, the accept ratio is already over 90%.

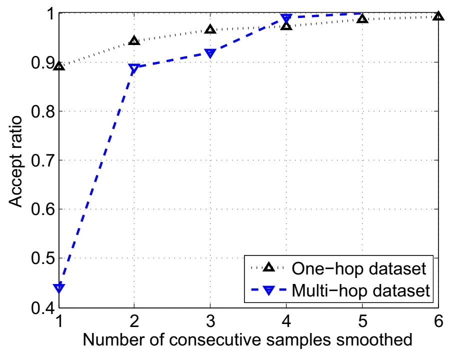


Figure 6. Accept ratio.

Figure 7 shows that correct ratio is above 99% for both datasets with smoothing number above 2. For multi-hop dataset, the correct ratio keeps as 100%. The high correct ratio comes from our robust observation.

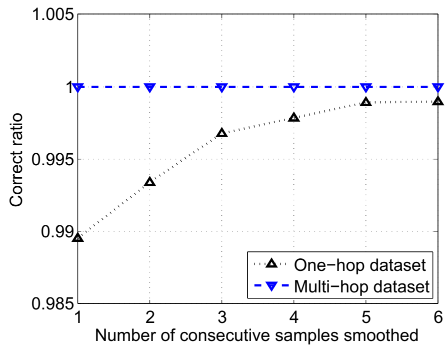


Figure 7. Correct ratio.

In summary, with smoothing number set as 2, we can obtain an ordering which is correct with probability over 99%. We can even boost this probability by taking a maximum vote of several outputs.

C. Comparisons

In our framework, we can use other information. We implemented two other algorithms. One is based on BETO: check whether a given permutation is consistent with the ordering graph under BETO. It turns out that this algorithm has 0% accept ratio in both datasets. There are always a few nodes that can not correctly tell the near-far relationship between distant neighbors. More importantly, we can not identify such nodes if the true node sequence is unknown.

The other algorithm is based on connectivity information. If we assume uniform transmission range, we can formulate the ordering derivation as a *unit interval graph recognition* problem, which is a well explored research area [13]. On the other hand, if we allow nodes to have different transmission ranges, then the problem becomes an *interval graph recognition* problem [14]. Since connectivity information can not distinguish between nodes with the same neighbors, we use the second dataset to test it. Both methods lead to 0% accept ratio. This phenomenon can be explained as follows. The underlying assumption of connectivity is that signals have no "hole" during propagation. That is, if nodes a, b, c are placed as a-b-c and c can hear from a, then b should also be able to hear from a. However, in practice, this does not hold necessarily, especially when nodes b and c are far from a.

VII. Conclusion

In this paper, we study the problem of inferring spatial ordering of sensor nodes. We find a reliable observation which states that in a certain direction the nearest node will observe the highest RSSI. This observation is used to design a verification algorithm for nodes at the edge of the network field, which in turn leads to an algorithm for deriving the ordering. We evaluate it using two datasets and observe that it derives the correct ordering in most samples.

Acknowledgement

The authors would like to thank the anonymous reviewers for the helpful comments and suggestions. The work is partly supported by China NSF grants (60825205, 61073152, 61021062).

References

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks:a survey," *Computer Networks*, vol. 38, no. 4, pp. 393 – 422, 2002.

[2] J. Hightower and G. Borriello, "Location systems for ubiquitous computing," *Computer*, vol. 34, no. 8, pp. 57–66, 2001.

[3] Z. Guo, Y. Guo, F. Hong, X. Yang, Y. He, Y. Feng, and Y. Liu, "Perpendicular intersection:locating wireless sensors with mobile beacon," in *Proceedings of IEEE RTSS*, 2008.

[4] C. Liu, K. Wu, and T. He, "Sensor localization with ring overlapping based on comparison of received signal strength indicator," in *Proceedings of IEEE MASS*, 2004.

[5] S. Kim, S. Pakzad, D. E. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *Proceedings of ACM IPSN*, 2007.

[6] Z. Zhong, T. Zhu, D. Wang, and T. He, "Tracking with unreliable node sequences," in *Proceedings of IEEE INFOCOM*, 2009.

[7] TinyOS, "http://www.tinyos.net."

[8] Z. Zhong and T. He, "Achieving range-free localization beyond connectivity," in *Proceedings of ACM SenSys*, 2009.

[9] R. Bischoff and R. Wattenhofer, "Analyzing connectivity-based multi-hop ad-hoc positioning," in *Proceedings of IEEE PerCom*, 2004.

[10] Z. Lotker, M. M. de Albeniz, and S. Perennes, "Range-free ranking in sensors networks and its applications to localization," in *ADHOC-NOW*, 2004.

[11] X. Zhu, D. Luo, and G. Chen, "Proximity list: Rssi-assisted relative localization in one dimensional wireless sensor networks," *Journal of Software*, vol. 20, no. zk, pp. 257–265, 2009.

[12] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proceedings of ACM SenSys*, 2009.

[13] D. G. Corneil, "A simple 3-sweep lbfs algorithm for the recognition of unit interval graphs," *Discrete Applied Mathematics*, vol. 138, no. 3, pp. 371–379, 2004.

[14] D. G. Corneil, S. Olariu, and L. Stewart, "The lbfs structure and recognition of interval graphs," *SIAM J. Discrete Math.*, vol. 23, no. 4, pp. 1905–1953, 2009.