Computer Communications 36 (2013) 1581–1591

Contents lists available at ScienceDirect

Computer Communications

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

Relative localization for wireless sensor networks with linear topology ☆

Xiaojun Zhu [a], Xiaobing Wu [a,*], Guihai Chen [a,b]

[a] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
[b] Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, Shanghai 200240, China

ARTICLE INFO

Article history:
Received 4 March 2013
Received in revised form 17 May 2013
Accepted 26 July 2013
Available online 3 August 2013

Keywords:
Wireless sensor networks
Localization
One-dimension
Linear network
Spatial ordering

ABSTRACT

Sensors can be deployed along a road or a bridge to form a wireless sensor network with a linear topology where nodes have a spatial ordering. This spatial characteristic indicates relative locations for nodes and facilitates such functions as object tracking and monitoring in the network. We attempt to derive this ordering with RSSI (Received Signal Strength Indicator) as the only input, which does not require attaching extra hardware to the sensor nodes. This requires a method for translating RSSI into spatial constraints. There are two candidate methods in the literature. The first method uses connectivity information among the nodes to calculate their relative locations. But analysis of real-world trace data indicates that it does not work well in reality. The second method assumes that closer nodes receive higher RSSI. However, we have proved that the relative localization problem is actually NP-hard under such an assumption. Fortunately, the problem turns out to be efficiently solvable if we adopt a new observation slightly different from the above-mentioned closer-higher RSSI assumption. This observation that the closest node always receives the highest RSSI is verified by the analytical results of the same real-world trace data. We then propose a spatial ordering method based on the observation, and evaluate it through various field experiments. Results show that the proposed method achieves an accuracy of over 99%.

1. Introduction

Location information is critical for many wireless sensor networks [2,3]. Relative localization is to locate sensors with respect to each other such that the resulting locations may be up to an arbitrary rotation and translation to the ground-truth. It is often performed in scenarios where anchors are unavailable. In this paper, we consider relative localization in 1-dimensional (1-D) sensor networks. Practical 1-D networks include sensor networks deployed along a road to facilitate vehicular networks and that along a bridge for structural health monitoring [4]. We define the relative locations to be the ordering of nodes in the network field. Such ordering is important for functions like object tracking and monitoring in the network.

We target at deriving the ordering without attaching extra hardware to sensor nodes. Therefore, for the input information, we restrict it to RSSI (Received Signal Strength Indicator) only. We believe that an RSSI-based solution can be rapidly and inexpensively deployed, since RSSI is readily available in most wireless platforms. Another approach to get the ordering is to manually label the nodes during deployment. Our method can benefit this process in two aspects. First, it can help verify the ordering obtained manually. Currently, sensor motes do not contain a unique identifier in the hardware. Distinguishing between nodes is achieved by setting a node ID for each node when the node is programmed (usually by typing the ID in a command window in the case of TinyOS [5]). It is not uncommon for programmers to worry that some nodes are mislabeled, especially when there are too many of them. Second, our method can be used to re-establish the ordering in situations where node IDs are changed, which is possible during a software upgrade or when newly joined nodes result in ID collision. Though sometimes keeping track of node IDs across the change is possible, it may again contain error and our method can help detect this error. Our solution is an attractive alternative to ordering derivation in that it does not require specialized hardware and our experiments show that it has high accuracy.

A key step of our solution is to translate RSSI into spatial constraints. There are two such methods in the literature. The first is to infer connectivity information from RSSI and then assume that two nodes are connected if and only if they are within the transmission range of each other. Unfortunately, we show by real-world trace that this assumption does not always hold due to signal propagation anomaly, and the solution based on this method hardly works in practice. The second is to assume that closer nodes observe larger RSSI, which implicitly requires that closer nodes

☆ A preliminary version of this paper appeared in [1].

* Corresponding author. Tel.: +8613505184210.
E-mail addresses: gxjzhu@gmail.com (X. Zhu), wuxb@nju.edu.cn (X. Wu), gchen@nju.edu.cn (G. Chen).

0140-3664/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comcom.2013.07.007

should at least receive all signals received by farther nodes. Strictly following this assumption will fail just like the first method, we relax it by removing the implicit requirement. The relaxed version assumes that, among the nodes that have successfully received signals from the sender, larger RSSI indicates shorter distance. To see the difference, consider the situation where nearby nodes do not receive a message received by a distant node. This situation is not consistent with the original assumption, but may be consistent with the relaxed version. Unfortunately, we prove that the relative localization problem under this assumption is NP-hard to solve.

After analyzing these failures and examining the real-world data, we notice that, though closer nodes may not observe larger RSSI, the closest node usually observes the largest RSSI. This changing from "closer" and "larger" to "the closest" and "the largest" surprisingly makes a quite reliable observation, which is verified by a real-world trace data. Directly using this observation as a verification rule can already solve the problem, but its running time grows exponentially with the input size. We further study the properties of this observation and reduce significantly the number of candidate orderings that need to be verified. Our final solution runs in polynomial time.

To evaluate the effectiveness of our solution, we conduct experiments in three different roads in campus, including a straight road, a road with a right angle turn, and an S-shaped road. In each road, we deploy sensor nodes along the road and collect RSSI traces for about 2 h. Applying our solution to these traces shows that our solution works efficiently in all three roads, achieving an accuracy of over 99%. Additionally, we study the impact of other factors on the performance of our solution. These factors include inter-node distance, obstacle, and wireless interference. Results show that in some scenarios we need to collect more data to obtain the nodes' ordering, but once we get an ordering, it matches the ground-truth in over 99% cases in all the experiments we conducted.

The rest of the paper is organized as follows. Section 2 summarizes related work. Section 3 formulates the problem. Section 4 examines two kinds of candidate solutions. We present our approach in Section 5, discuss its implementation issues in Section 6 and evaluate the approach in Section 7. Section 8 concludes the paper.

2. Related work

Localization methods can be divided into two categories: range-based methods and range-free methods [6]. Range-based methods usually assume that nodes can estimate the distance or angle, which requires extra hardware. Thus we focus on the range-free methods, which use readily available information, such as connectivity information, or RSSI.

2.1. Connectivity-based localization

The assumption about connectivity is that two nodes can hear from each other if and only if they are within a certain distance. Various works follow this assumption [7–13]. Among them, some researchers consider 1-D scenario. Bischoff et al. [12] give a range-free localization method for 1-D sensor networks. Their work has provable performance guarantee. Lotker et al. [13] give *range free ranking* for nodes in 1-D networks. In this approach, the two end nodes are given as anchor nodes with one flooding a message and the other terminating the flooding. During the flooding, each node computes its *rank* in the spatial ordering. Though having elegant theoretical foundation, these works have not been tested in practice. In this paper, we find from real-world data that connectivity information, or connectivity assumption, does not work well in solving our problem.

2.2. RSSI-based localization

Besides connectivity information, RSSI is also a popular choice for localization. The reason is that RSSI is available in most existing wireless platforms and is believed to be a "free-lunch". At first, researchers use absolute RSSI values to estimate physical distance between nodes, which has poor quality of distance estimation and poor localization accuracy. Recently, some researchers realize that RSSI performs better when only using it to indicate near-far relationship, which leads to better solutions for localization problems [14–16]. The observation they rely on is that shorter distance results in larger RSSI. In [15], a node estimates its distance to different anchors by comparing the observed RSSI values. This information helps refine the node's location to be within intersection of different rings. Zhong et al. [16] propose *signature distance* to measure the distance between neighboring nodes. The key is to characterize each node by a *signature*, a neighbor list ordered by their RSSI. Guo et al. [14] use a mobile beacon to locate sensor nodes. The beacon moves along designed routes and continuously broadcasts messages. Each node records the observed RSSI and compares these RSSI values to locate itself. We will examine an approach based on this observation later. We find that, in our scenario, this observation does not work well either. Recent progress in distinguishing radio paths [17] can help adapt our solution to indoor environment.

3. Problem formulation

For convenience, we use the terms "sensor", "node", and "sensor node" interchangeably in the rest of the paper. Since an ordering is a sequence or permutation of node IDs, the terms "ordering", "sequence", and "permutation" are used interchangeably in this paper.

There are n sensor nodes $1, 2, \ldots, n$ deployed along a line. Sensors take turns to broadcast messages, with each message containing a sender ID and a sequence number. Upon receipt of a message, a sensor will read the RSSI of this message and record a tuple $(sender, seq, receiver, rssi)$, where *sender* and *seq* are the sender ID and the sequence number from the message respectively, *receiver* is the current sensor's ID and *rssi* is the RSSI of the received message. We initiate all sequence numbers to be 0 and repeatedly increase them by 1. Consequently, each sequence number corresponds to a collection of tuples. We refer to such a collection with sequence number s as "sample s of the network status", or simply "sample s". A sample can be represented by an $n \times n$ matrix $\mathbf{S}$ with each row corresponding to a sender, each column corresponding to a receiver, and the data field being the corresponding RSSI. This matrix can be asymmetric due to packet loss or different transmission power. Fig. 1 illustrates a sample of a 4-node network.

Samples are the input information for our problem. There are several remarks on a "sample". First, the broadcast messages in the above process can be replaced by normal messages, since we do not rely on the content of the messages. This reduces the

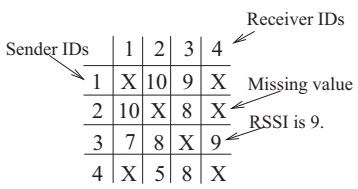


Fig. 1. A sample of network status. This sample is from a 4-node network and is constructed from 9 tuples.

communication overhead. Second, the notion of sample does not necessarily imply centralized processing. It is a summary of input information, and is simply for the ease of presentation. As we can see later, it is possible to use a distributed algorithm. Third, each sample corresponds to a problem instance. Obviously, we can merge several samples to a single one by taking the average, but the resulting sample corresponds to one problem instance.

On the other hand, the output should be the spatial ordering of sensor nodes. This spatial ordering is a permutation of node IDs that reflect nodes' physical locations. We denote such an ordering by π with $\pi(1), \pi(2), \ldots, \pi(n)$ representing the node IDs along the line. Here we consider an ordering π and its reversion $\pi(n), \pi(n-1), \ldots, \pi(1)$ as the same. Our problem can be summarized as follows.

Problem 1. Given a sample **S** of the network status, find the spatial ordering π of sensor nodes.

The key to solve this problem is how to translate the given sample into spatial constraints. We will examine two existing approaches (for translating samples into constraints), study the resulting new problems, and propose our own approach.

Due to the unpredictable nature of wireless signal propagation, no algorithm can guarantee generation of correct ordering for all inputs. In cases where an algorithm cannot find any ordering or cannot distinguish between several orderings based on the input, we allow it to output a *null* instead of randomly picking one possible ordering as output. This design is due to the belief that no information is better than wrong information. Consequently, possible answers of Problem 1 include a wrong ordering, a correct ordering, and a *null*. We would like an algorithm to give an ordering for as many inputs as possible and, if it outputs an ordering, the ordering matches the ground-truth with high probability. To this end, we design the following two performance metrics for evaluating an algorithm:

1. *reliability*, $\eta_R = n_o/n_s$, where n_o is the number of samples on which the algorithm gives non-*null* answers and n_s is the total number of samples.
2. *accuracy*, $\eta_A = n_a/n_o$, where n_a is the number of samples on which the algorithm gives the correct ordering.

We prefer high reliability and high accuracy. Higher reliability means that more samples yield non-*null* answers, given a certain number of samples as input. Thus, on average, the algorithm requires less samples to derive an ordering (i.e., non-*null* answer). Higher accuracy means that a non-*null* answer (ordering) returned by the algorithm matches the ground-truth with higher probability. We will use these two metrics to examine solutions derived from existing works and our solution.

4. Candidate solutions

We motivate our solution by considering two kinds of candidate solutions. These solutions differ in how to translate RSSI information into spatial constraints. For preliminary evaluation, we use a real-trace from the literature [16], where 54 nodes are placed in a line.

4.1. Connectivity-based solution

Two nodes are said to be *connected* if they can receive messages from each other. We can translate RSSI information directly into connectivity information. For an RSSI sample, two nodes are connected if and only if they have RSSI records from each other in the sample. For the sample in Fig. 1, we can find that the following node pairs are connected: (1, 2), (1, 3), (2, 3) and (3, 4). Note that nodes 2 and 4 are disconnected since node 4 cannot receive messages from node 2.

The key assumption translating connectivity information into spatial constraints is that two nodes are connected if and only if they are within a certain distance from each other. This distance is usually referred to as the transmission range. Without loss of generality, we assume that this distance is 1. We can now formally define the problem.

Problem 2. Given an undirected connectivity graph $G = (V, E)$ where V is the set of nodes deployed along a line and E is the set of edges indicating connectivity between nodes, find an ordering π of nodes such that there exist node locations $f : V \longrightarrow R$ satisfying both (1) $f(\pi(i))$ is increasing with respect to the increase of i; and (2) for any $u, v \in V$, $(u, v) \in E$ if and only if $|f(u) - f(v)| \leqslant 1$.

The first condition requires that f follows ordering π and the second condition realizes the connectivity assumption. Though seemingly hard, this problem can actually be solved in linear time by the following transformation.

The problem can be transformed to the recognition of *unit interval graph* [18]. A graph G is a unit interval graph if and only if we can find a set of unit intervals in a real line such that each interval represents a distinct vertex in G and two intervals have intersections if and only if the two corresponding vertices are connected by an edge in G. It is straightforward (by definition) to prove the following theorem.

Theorem 1. *Given an instance of Problem 2, there exists an ordering π satisfying the conditions if and only if G is a unit interval graph. In addition, the ordering of the intervals in G's any unit interval representation satisfies the conditions.*

Finding a unit interval representation of G, on the other hand, can be solved by a very simple and efficient algorithm [18]. This algorithm is based on the following characterization of unit interval graphs ([18]): A graph $G = (V, E)$ is a unit interval graph if and only if there is an ordering of V such that for all $x < y < z, xz \in E \Rightarrow xy, yz \in E$. The algorithm then uses lexicographic breadth-first search, a variant of breadth-first search, three times to find such an ordering. Based on Theorem 1, this found ordering is just the node ordering we want to derive. Now we have our first solution summarized in Algorithm 1, where we first generate the connectivity graph and then use the algorithm in [18] to find the ordering.

Algorithm 1: Ordering_by_connectivity

Input: S, a sample of the network condition
Output: π, nodes' ordering
1 **begin**
2 Generate the connectivity graph $G = (V, E)$ from **S** such that $V = \{1, \ldots, n\}$ and $(u, v) \in E$ if and only if both **S**(u, v) and **S**(v,u) are not empty
3 Run the 3-sweep LBFS algorithm proposed in [18] with G as the input
4 If the algorithm finds a unit interval representation for G, set π as the corresponding ordering of nodes; otherwise, set π as *null*.
5 **end**

Algorithm 1 does not work when applied to real-world trace. One may expect that we will get high reliability but low accuracy for this approach. Indeed, since connectivity information cannot distinguish neighbors, there could be multiple orderings satisfying

the constraints besides the correct one, resulting in high reliability. On the other hand, randomly outputting one of the satisfying orderings results in low accuracy. In practice, it turns out that Algorithm 1 has 0% reliability, i.e., it cannot find even a single ordering (either correct or wrong). In other words, the resulting connectivity graph for every sample is, in fact, not a unit interval graph.

The reason is that connectivity assumption does not hold in the trace. In both the trace and our experiments, we consistently observe a phenomenon that we refer to as *propagation anomaly*. Specifically, some farther nodes successfully receive signals from the sender, but some closer nodes do not. It is "anomaly" since we expect that closer nodes can always receive the signals given that farther nodes can receive them. Obviously, propagation anomaly is not consistent with the connectivity assumption, causing the failure of Algorithm 1. Thus, any assumption inconsistent with propagation anomaly will fail in the same way. We conclude that the connectivity assumption is not suitable for our problem.

4.2. RSSI-based solution

The second approach relies on the following assumption: *The closer a node is to the sender, the stronger signal it receives.* This assumption uses more information than the connectivity approach, and it has been utilized in localization scenarios in several works [15,19,16].

Unfortunately, schemes under this assumption are unable to deal with propagation anomaly mentioned before. To see this, note that the assumption implicitly requires closer nodes to receive signals received by farther nodes. If we use this assumption directly, the resulting algorithm will also have 0% reliability just like Algorithm 1. Thus, we need to fix this assumption. To make it accommodate propagation anomaly, we change the assumption to be: *Among the nodes that receive signals from the sender, the stronger signal a node receives, the closer the node is to the sender.*

This new assumption has the following properties. First, it can accommodate propagation anomaly: if a node does not receive signals from a sender, then there is no requirement on its distance to the sender. Second, if two nodes both receive the signal but observe the same RSSI, then there is still no requirement about their distances to the sender. We note that, in practice, it is not uncommon for two nodes to observe the same RSSI value due to accuracy limitation. RSSI is an 8-bit integer at most sensor motes, and it is usually within a smaller range in practice. (The range is [-96,-52] for all sensors in Section 7.1.1.).

Using this assumption to infer the ordering, however, is very difficult. We formalize the spatial ordering problem under this assumption as follows and will prove that this problem is actually NP-hard.

Problem 3. Given a sample $\mathbf{S}$, find an ordering π of nodes such that there exists a mapping $f : V \longrightarrow R$ satisfying both (1) $f(\pi(i))$ is increasing with respect to the increase of i; and (2) for any i,j,k such that both $\mathbf{S}(i,j)$ and $\mathbf{S}(i,k)$ are not missing, if $\mathbf{S}(i,j) > \mathbf{S}(i,k)$, then $|f(i) - f(j)| < |f(i) - f(k)|$.

Again, the first constraint requires that f follows ordering π. The second constraint realizes the assumption. The decision version of this problem is as follows.

Problem 4 (*Decision Version of Problem 3*). Given a sample $\mathbf{S}$, does there exist a mapping $f : V \longrightarrow R$ such that for any $i,j,k \in V$ where both $\mathbf{S}(i,j)$ and $\mathbf{S}(i,k)$ are not missing, if $\mathbf{S}(i,j) > \mathbf{S}(i,k)$, then $|f(i) - f(j)| < |f(i) - f(k)|$?

Regarding the hardness of this problem, we have the following theorem.

Theorem 2. *Problem 4 is NP-hard.*

Proof. We prove the theorem by reduction from the Betweenness problem which is known to be NP-hard. It is defined as follows.

Definition 1 (*Betweenness [20]*).

Given a finite set A, a collection C of ordered triples (a,b,c) of distinct elements from A, is there a one-to-one function $g : A \to \{1,2,\ldots,|A|\}$ such that for each $(a,b,c) \in C$, we have either $g(a) < g(b) < g(c)$ or $g(c) < g(b) < g(a)$?

Given an instance of the Betweenness problem, construct the following instance of Problem 4. For each $a \in A$, construct a group of $k_a + 1$ vertices (nodes) $a_0, a_1, a_2, \ldots, a_{k_a}$ in V where k_a is the number of occurrences of a in C. This construction yields $|V| = 3|C| + |A|$. For each $(a,b,c) \in C$, suppose this is the ith occurrence for a, j-th for b, and kth for c (i,j,k start from 1), then set $\mathbf{S}(a_i,b_j) = 1, \mathbf{S}(a_i,c_k) = 0; \mathbf{S}(c_k,b_j) = 1, \mathbf{S}(c_k,a_i) = 0$. Finally, for each $a \in A$, set $\mathbf{S}(a_0,a_i) = 1$ for all $i \geqslant 1$; $\mathbf{S}(a_0,b_j) = 0$ for all $b \neq a$ and $j = 0,\ldots,k_b$. All other elements of $\mathbf{S}$ are missing. This construction can be done in polynomial time. To see this, note that $|V| = 3|C| + |A|$ and $|\mathbf{S}| = |V|^2$. Both are polynomial in $|A|$ and $|C|$. Now we prove that the original Betweenness instance has a function g if and only if the constructed instance has a mapping f. Suppose there is a one-to-one function g for Betweenness instance. Then consider the following f. For each $a \in A$ and $i = 0,\ldots,k_a$, let $f(a_i) = g(a)$. We will verify that f is the desired mapping. Consider any a_i, b_j, c_k such that $\mathbf{S}(a_i,b_j) > \mathbf{S}(a_i,c_k)$. If $i = 0$, then $b = a$ and $c \neq a$ so that $f(a_i) - f(b_j) = 0$ and $f(a_i) - f(c_k) \neq 0$. It holds trivially that $|f(a_i) - f(b_j)| < |f(a_i) - f(c_k)|$. Otherwise ($i \neq 0$), then $(a,b,c) \in C$ so that either $g(a) < g(b) < g(c)$ or $g(c) < g(b) < g(a)$. Therefore, $|f(a_i) - f(b_j)| = |g(a) - g(b)| < |g(a) - g(c)| = |f(a_i) - f(c_k)|$.

On the contrary, suppose there exists a mapping f for the Problem 4 instance. Then set the function g as follows. For each $a \in A$, set $g(a) = |\{b \in A | f(b_0) \leqslant f(a_0)\}|$, i.e., the rank of a_0 among $\{b_0 | b \in A\}$ under f. We show that g is the desired function. On one hand, f has a special property. For any $a \in A$, the vertices $a_0,\ldots,a_{k_a}$ are consecutive under f, i.e., there does not exist $b \neq a$ and $j \geqslant 0$ such that $f(b_j)$ is between $f(a_{i_1})$ and $f(a_{i_2})$ for some i_1, i_2. This is because we have set $\mathbf{S}(a_0,a_i) = 1$ for all i and $\mathbf{S}(a_0,b_j) = 0$ for all $b \neq a$. This special property implies that, if there exist some a_i, b_j, c_k for distinct a,b,c such that $f(b_j)$ is between $f(a_i)$ and $f(c_k)$, then $f(b_0)$ is definitely between $f(a_0)$ and $f(c_0)$. On the other hand, for each $(a,b,c) \in C$, there exist three vertices a_i, b_j, c_k such that $\mathbf{S}(a_i,b_j) > \mathbf{S}(a_i,c_k)$ and $\mathbf{S}(c_k,b_j) > \mathbf{S}(c_k,a_i)$. These two inequalities will force $f(b_j)$ to be between $f(a_i)$ and $f(c_k)$. Consequently, $f(b_0)$ is between $f(a_0)$ and $f(c_0)$ so that $g(b)$ is between $g(a)$ and $g(c)$. □

We exclude this solution from further consideration for three reasons. First, there does not exist a polynomial algorithm to solve the resulting problem, unless $P = NP$. What we currently can do is to use a brute-force search to check all possible permutations. (For any permutation, we can verify whether it is consistent with the assumption by linear programming.) But its running time is prohibitive. Second, although we do not exclude the possibility of approximation algorithms, the exact definition of approximation is beyond the scope of this paper. Third and most importantly, we find that a new observation helps us come up with a solution which runs in polynomial time and works very effectively in practice. We present the observation together with the solution in the next section.

5. Deriving the ordering efficiently

Inspired by the candidate solutions, we study the relationship between signal strength and distance. In this section, we present our new observation and describe how to design an efficient algorithm based on this new observation.

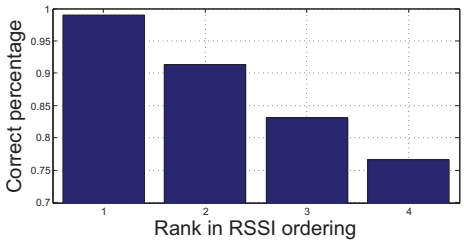
5.1. A new observation

Recall the dataset [16] we used in Section 4, which is collected from 54 sensors deployed along a line. We divide each node's neighbors into two groups according to their physical locations: left side neighbors and right side neighbors. For each group, we sort these nodes by RSSI values in descending order and refer to this sorted list as *RSSI ordering*. Since there are 54 nodes in the network, we have $2 + 2 \times 52 = 106$ RSSI orderings. To see this, note that the two end nodes have either right side neighbors or left side neighbors but not both, resulting in two RSSI orderings; the rest nodes have both left side neighbors and right side neighbors, contributing 2×52 RSSI orderings. Then, we check each RSSI ordering to see whether it reveals the true distance relationship.

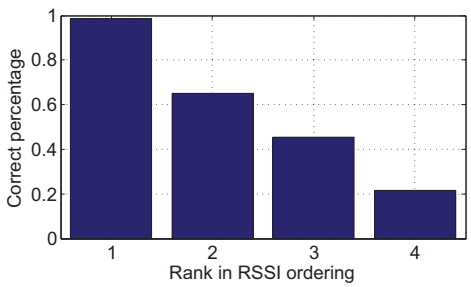
Fig. 2(a) shows that over 99% RSSI orderings find the nearest neighbor (rank 1 in the RSSI ordering) correctly. This percentage decreases as the increase of rank. Less than 80% RSSI orderings tell correctly about the 4th nearest node to the sender. Therefore, it is highly probable that in certain direction from the sender, the nearest node observes the best signal. When nodes are far away from the sender, it is less reliable to infer their near-far relationship by comparing RSSI values. Actually, this is reasonable if we consider the process of signal propagation. Signals spread during propagation and become weaker and weaker. The same noise level can influence weaker signals more easily. As a result, the RSSI ordering becomes less reliable in differentiating distant neighbors.

Our observation is as follows. *Among the nodes in the same direction from the sender, the closest node observes the best signal.* This property has also been verified in our experiments. For example, Fig. 2(b) shows the correct percentage for our data collected from a sensor network with 18 nodes deployed along a straight road (details can be found in Section 7.1.1). Though less than 70% RSSI orderings correctly identify the second closest node, more than 98% RSSI orderings find the closest neighbor.

We compare this observation with the two candidate solutions described in Section 4. First, this observation uses more information than the connectivity approach in that it explores the magnitude of signal strength. Second, it is weaker than the connectivity approach in that this observation allows propagation anomaly in some way. In a certain direction from the sender, as long as the closest node receives the strongest signals, it does not matter whether other nodes in this direction receive signals or not. Thus, the propagation anomaly can exist among distant nodes. Third, it is simpler than the second candidate solution in that only the two closest nodes, each for a direction from the sender, are required to observe a certain signal strength. As we can see later, this observation leads to a polynomial-time solution.



(a) data from the literature [16]



(b) data from our experiment in Section 7.1.1

Fig. 2. Compare RSSI orderings with distance orderings.

5.2. A Naive exponential-time algorithm

The straightforward solution is to use our observation as a *verification rule*. Given a permutation of nodes, we can verify whether this permutation is consistent with the input under our observation. Unlike the second candidate solution, this verification can be done easily without linear programming.

For ease of presentation, we transform the input information, a sample, into an *ordering matrix* defined as follows. For node i, we sort all other nodes in descending order according to their observed signal strength from that node. The sorted list, after removing nodes that do not receive signals from node i, is node i's *neighbor list*, denoted as α_i. The *ordering matrix* is defined to be the set of all neighbor lists. Following this definition, the top of Fig. 3 shows the ordering matrix for the sample in Fig. 1.

An algorithm follows easily. Given an ordering matrix, we use a brute-force search to check all permutations of nodes. For any permutation, we verify its consistency with the ordering matrix by iterating over all nodes. In each iteration, we focus on a distinct node, divide other nodes into two groups, left-side group and right-side group, according to the given permutation, and check each group in the focused node's neighbor list to see whether our observation holds. If all iterations pass the checking, we conclude that the given permutation is consistent with the ordering matrix. For the ordering matrix in Fig. 3, it is consistent with the permutation 1–2–3–4. However, it is inconsistent with the permutation 2–1–4–3. To see this, note that node 1's right-side group consists of nodes 4,3 where the closest one is node 4, while in α_1, node 3 is the closest in this group.

The worst-case running time of this algorithm is $\Omega(n!)$, making it impractical even in medium-sized sensor networks. We explore other properties of our observation to speed up the algorithm.

5.3. A polynomial-time algorithm

The exponential running time of the straightforward solution is mainly resulted from unnecessary checking. For example, if a subsequence 1–2-3 cannot pass the checking because α_3 states that node 1 observes stronger signal than node 2, then all sequences containing this subsequence cannot pass the checking either. We have the following lemma regarding a valid permutation.

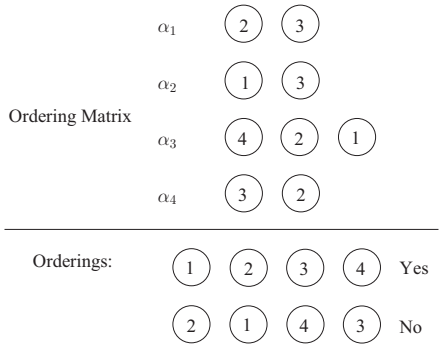


Fig. 3. The ordering matrix (top) for the sample in Fig. 1 and two permutations of nodes (bottom).

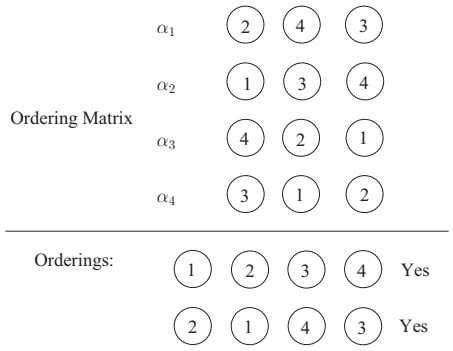


Fig. 4. One ordering matrix may be consistent with two orderings.

Lemma 1. *Given an ordering matrix O and a permutation π, if π is consistent with O, then*

1. *$\pi(2)$ is the first node in $\alpha_{\pi(1)}$;*
2. *the permutation $\pi \setminus \{\pi(1)\}$, i.e., $\pi(2)\pi(3)\cdots\pi(n)$, is consistent with $O \setminus \{\pi(1)\}$, the ordering matrix after removing node $\pi(1)$.*

Although this lemma only characterizes necessary conditions for a permutation to be consistent with the given ordering matrix, the number of permutations satisfying these conditions is limited. In fact, at most *n* permutations can satisfy these conditions, which is shown in the following lemma.

Lemma 2. *Given an ordering matrix O and any node i, there is at most one permutation π with $\pi(1) = i$ satisfying the conditions in Lemma 1.*

Proof. According to Lemma 1, we can actually determine the permutation π as follows. Since $\pi(1)$ is fixed and π satisfies the conditions in Lemma 1, we can determine which node is $\pi(2)$ — it should be the first node in $\alpha_{\pi(1)}$. After fixing $\pi(2)$, we remove $\pi(1)$ from both π and *O* and face a smaller problem. If we can repeat this process until $\pi(n)$ is fixed, then we find one permutation; otherwise, there is no permutation satisfying the conditions. □

This lemma shows that we need to check at most *n* permutations. But how to check each of them efficiently?

Theorem 3. *Given an ordering matrix O and a permutation π, π is consistent with O if and only if both $\pi(1)\pi(2)\cdots\pi(n)$ and $\pi(n)\pi(n-1)\cdots\pi(1)$ satisfy the conditions in Lemma 1.*

Proof. It is straightforward to see "$\Rightarrow$" since both π and its reversion are consistent with *O*. To see "$\Leftarrow$", we only need to verify for each node whether its two closest nodes observe the largest RSSI in their corresponding groups. Now consider node *i*. Its right-side group consists of $\pi(i+1),\ldots,\pi(n)$, and $\pi(i+1)$ receives the strongest signal because π satisfies the conditions; its left-side group consists of $\pi(i-1),\ldots,\pi(1)$, and $\pi(i-1)$ receives the strongest signal because the reversion of π satisfies the conditions. □

This theorem shows that we only need to verify the reversion of the permutation to see whether it satisfies the conditions in Lemma 1.

An efficient algorithm follows. It consists of two parts. The first part is in Algorithm 2, which finds candidate orderings, and the second part is in Algorithm 3, which verifies each candidate ordering's consistency and outputs the final result. We describe each part in detail. Given a node, Algorithm 2 finds the permutation in Lemma 2 that can be consistent with the ordering and starts with the given node. This is done by following Lemma 1. We iteratively include a node in the permutation until all nodes are included. Before each iteration, there is one node occurring in the current ordering matrix and already included in the permutation. During the iteration, we infer from this node's neighbor list the adjacent node, include this adjacent node in the permutation (Line 5), delete the old one from the ordering matrix (Line 6), and prepare for the next iteration (Lines 7–8). If this process cannot proceed, then no desired permutation exists (Lines 9–11). In the algorithm, the notation $O \setminus \{i\}$ means to delete node *i* from the current ordering matrix *O*. Outputting *null* means that there is no permutation that begins with the given node and is consistent with the ordering matrix.

Algorithm 2: Find_the_permutation

```
   Input: O, an ordering matrix; i, a node
   Output: π, a permutation of nodes
1  begin
2      π(1) ⟵ i; j ⟵ 2; O ⟵ O\{i};
3      while j ≤ n do
4          if α_i ≠ ∅ then
5              π(j) ⟵ the first element of α_i;
6              O ⟵ O\{i};
7              i ⟵ π(j);
8              j ⟵ j + 1;
9          else
10             π ⟵ null;
11             break;
12 end
```

The second part is straightforward. We enumerate all nodes, find for each node the corresponding candidate permutation by Algorithm 2, and verify this candidate permutation by running Algorithm 2 again as suggested by Theorem 3. Algorithm 3 summarizes the whole process, and is our final solution to Problem 1. There is one additional issue during the design of Algorithm 3. We have known that some sample may not give an ordering (either correct or wrong). In practice, some may give multiple orderings simultaneously. For example, the ordering matrix in Fig. 4 is consistent with two permutations simultaneously. In this case, we again output a *null* due to the inability to distinguish between the correct one and the wrong one (Lines 12–14 in Algorithm 3).

Algorithm 3: Ordering_derivation.

```
Input: S, a sample of the network condition
Output: π, nodes' ordering
1  begin
2  let O be the ordering matrix of S
3  π⟵null
4  candidates⟵{1,...,n}
5  while candidates ≠ ∅ do
6  Pick i and remove it from candidates
7  π′⟵find_the_permutation(O,i)
8  if π′ ≠ null then
9  j⟵π′(n)
10 π″⟵find_the_permutation(O,j)
11 remove j from candidates
12 if π″ ≠ null ∧ π″ is reversion of π′ then
13 if π ≠ null then
14 π⟵null
15 return
16 else
17 π⟵π′
18 end
```

The running time of Algorithm 3 is shown in the following theorem.

Theorem 4. *Algorithm 3 can be implemented to run in $O(nm)$ time where n is the number of sensor nodes and m is the number of non-missing elements of the given sample.*

Proof. The key is to implement Algorithm 2 to take samples as input, instead of ordering matrixes. This avoids the time-consuming step for transforming samples into ordering matrixes at Line 2 in Algorithm 3. In the implementation of Algorithm 2, we maintain a bit vector to indicate which node has been included in the permutation, i.e., has been "deleted" in Line 6. In this case, when executing Line 5, we find the first node in α_i that is not marked; when executing Line 6, we simply mark the corresponding bit in the bit vector. Since each element in the input sample is visited at most once, the running time of Algorithm 2 is linear in the size of the sample. To this end, we use linked lists, instead of arrays, as the data structure for the given sample. The resulting number of elements in the lists should be $O(m)$. Therefore, Algorithm 2 runs in $O(m)$ time. It follows that Algorithm 3 runs in $O(nm)$ time, since it calls Algorithm 2 at most $O(n)$ times. □

6. Implementation issues

We first solve two problems encountered during implementation, i.e., how to collect a sample and how to merge several samples. Then we discuss distributed implementation.

There are two issues in collecting a sample from the network to the sink node. First, in Section 3, we assume nodes can take turns to broadcast messages so that the collected tuples will have the same sequence number. However, in practice, it is difficult to control such order. A natural solution is to design a token-based scheme. A token is transmitted in the network and a sensor can broadcast only if it posses the token. There are new challenges incurred such as how to distribute the token and how to deal with token loss. To avoid such problems, we drop the requirement on controlled broadcasting. Instead, we let nodes broadcast by themselves, using CSMA/CA to avoid collision. The key is to group tuples according to a $(sender, seq)$ pair such that each group corresponds to the same sender and the same sequence number. Then each group can contribute a row to the network sample matrix. We can verify that, as long as all nodes have broadcast a message, this uncontrolled approach can also generate a sample. Second, any data collection protocol, such as CTP [21], can be used to collect all tuples to the sink node, which then executes Algorithm 3 and gives the ordering. If the current sample cannot generate an ordering, a new sample should be collected. In many scenarios such as structural health monitoring applications, the network is designed to periodically collect data to the sink. In this case, the tuples can be piggybacked in regular transmissions to reduce overhead.

Because of probabilistic wireless propagation, a sample may not be consistent with any ordering due to packet loss or noise. To this end, we propose to smooth several samples by taking element-wise average of non-missing values. Formally, given k samples $\mathbf{S_1}, \mathbf{S_2}, \ldots, \mathbf{S_k}$ to be smoothed, the resulting sample $\mathbf{S}$ is computed by setting element $\mathbf{S}(i,j)$ for $1 \leqslant i,j \leqslant n$ to be the average of the multiset $\{\mathbf{S_t}(i,j) | 1 \leqslant t \leqslant k, \mathbf{S_t}(i,j) \neq X\}$, where $\mathbf{S_t}(i,j)$ is the RSSI corresponding to sender i and receiver j in sample $\mathbf{S_t}$, and X is a symbol indicating missing values. Fig. 5 gives an example for smoothing two samples. The influence of the parameter k on the performance of our algorithm will be studied in the next section, which indicates that two or three can already yield satisfactory results.

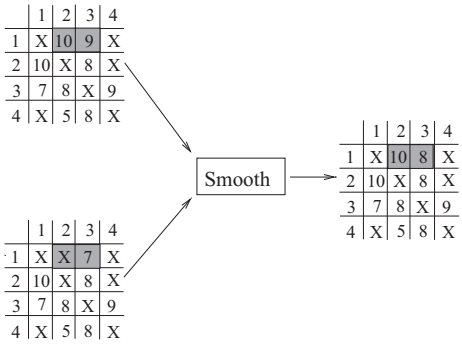


Fig. 5. Smooth two samples.

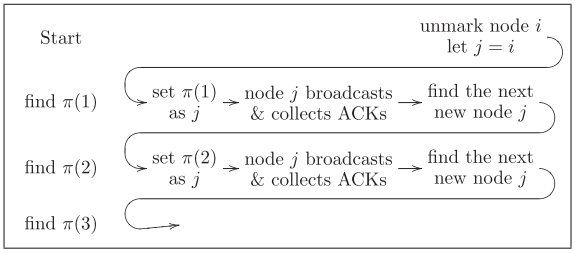


Fig. 6. An iteration initiated by node i. (Distributed implementation of Algorithm 3.)

Algorithm 3 can be adapted to a distributed algorithm. At first, each node is marked as a candidate node and this mark will be removed finally. Any node with this mark can initiate an iteration to find a consistent permutation. Once a node initiates an iteration, we remove its candidate mark, make it broadcast a message, and let others acknowledge this message together with the corresponding RSSI readings. Upon receipt of acknowledgements, the node initializing this iteration can determine the adjacent node by Lemma 1. This adjacent node is the second node in the permutation and it will broadcast a message to find out which one is the third node in the permutation. We illustrate such an iteration in Fig. 6. The process continues until all nodes appear in the permutation. The last node then initiates another iteration and we can check whether the resulting new permutation is the same as the previous one. If it is, then we find a consistent permutation; otherwise, we choose another candidate node to initiate the above process. The overhead of this approach depends on the channel condition. If there is no package loss, then node i should broadcast $O(nN_i)$ messages in the worst case where N_i is the number of neighbors of node i and n is the total number of nodes.

7. Evaluation

The preliminary evaluation in [1] has shown high reliability and high accuracy of our solution in two datasets. In this section, we conduct additional evaluations. We first introduce the experiment setup, and present our results on three different roads, followed by the study of other factors including inter-node distance and wireless interference.

Fig. 7 shows our equipments. We have 18 TmoteSky motes with external antenna (normal sensors), one TmoteSky mote without external antenna (the sink), and a laptop for data processing. Normal sensors are the subject of relative localization. Each normal sensor runs TinyOS 2.1.2 [5], and broadcasts a probe message periodically. The probe message contains a sender ID and a sequence number. Upon receiving a probe message, a sensor retrieves the

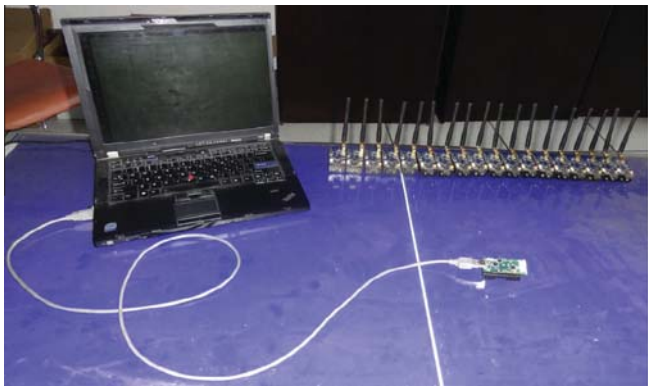


Fig. 7. Experiment equipments.

sender ID and sequence number from the message, reads from its own hardware the RSSI of this message, and adds a tuple $(sender, seq, receiver, rssi)$ into a local cache. When the number of local tuples exceeds a threshold, the sensor sends the cached tuples wirelessly to the sink using the CTP protocol [21]. The sink node also runs TinyOS 2.1.2, and it relays all received packets to the laptop via the USB cable connecting them. The laptop, running Ubuntu 12.04, then organizes received tuples into samples and writes them into a file.

Existing solutions [12,13] to our problem are based on connectivity assumption and they give the same result as Algorithm 1 in Section 4.1 does, so we mainly compare our algorithm with Algorithm 1. Note that the brute-force search solution mentioned in Section 4.2 is impractical, which needs to enumerate all permutations of nodes and solve a linear programming problem for each permutation.

7.1. Experiments in three roads

We consider three scenarios with increased difficulty: a straight road, a road with a right angle turn, and an S-shaped road. Each sensor sends a probe message every ten seconds, and data collection in each scenario lasts for about two hours, the time duration when our laptop runs out of battery. The statistics of the collected data are summarized in Table 1. We study our approach by smoothing different number of consecutive samples.

7.1.1. Scenario 1: straight road

In this scenario, we deploy the 18 normal sensors along a straight road in campus, as shown in Fig. 8. Sensors are roughly spaced by 3 meters, so the network covers about 51 meters of the road. This spacing is chosen such that the two farthest nodes cannot hear from each other reliably. We collect 701 samples in this scenario.

Fig. 9 shows the performance with respect to different numbers of consecutive samples smoothed. Without smoothing ($x = 1$), the reliability is over 94%. It increases to above 98% when every two consecutive samples are smoothed and to above 99% when every

Table 1
Data collected at three roads, where 18 sensor nodes are deployed along a road with specified shape. Nodes' average degree is defined for a sample, so it can be different for different samples. The statistics *min*, *mean* and *max* are with respect to all samples.

Scenario	Road shape	# of samples	Average node degree			RSSI	
			Min	Mean	Max	Min	Max
1	straight	701	11.22	15.21	15.89	−96	−52
2	L-shaped	706	11.17	14.66	15.61	−97	−45
3	S-shaped	706	15.39	16.49	16.94	−95	−43

Fig. 8. Sensor deployment along a straight road. From left to right: the road, a normal sensor, and the sink.

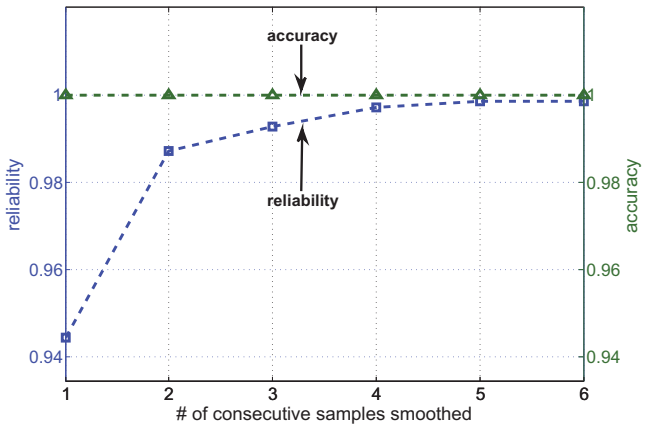


Fig. 9. Performance in Scenario 1.

three consecutive samples are smoothed. We find that smoothing compensates packet loss so it improves the performance. Indeed, even the nearest node does not always receive signals from the sender, especially if a pedestrian walks around. Merging consecutive samples reduces this chance. For this reason, we suggest to merge every two or three consecutive samples in practical deployments.

The performance of Algorithm 1 is not comparable to ours in that it cannot find the correct ordering in even one sample. The reason is due to the propagation anomaly mentioned before. To quantify this phenomenon, we define a *hole* to be a lost directed connection. For example, if nodes are deployed as $1, 2, 3, 4$ and node 1 is the only node that can hear from node 4, then we say there are two holes (nodes 2, 3) associated with node 4. The number of holes in a sample is the total number of holes associated

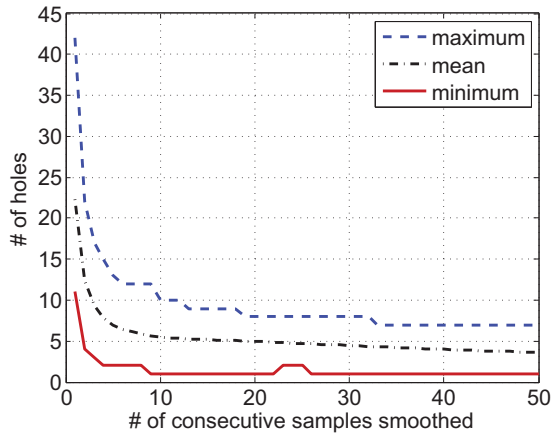


Fig. 10. Number of holes in Scenario 1.

with all nodes. If a sample contains holes, then Algorithm 1 cannot give the correct ordering. There are many holes in the samples, making Algorithm 1 inapplicable. Though smoothing can reduce the number of holes, it cannot eliminate them altogether. Fig. 10 shows the number of holes with respect to the number of consecutive samples being smoothed. Without smoothing, every sample contains more than 10 holes. Some sample contains even more than 40 holes. Holes exist in all samples even if every 50 consecutive samples are averaged.

7.1.2. Scenario 2: right angle turn

We find a T-shaped intersection in campus and deploy the 18 normal sensors around it as in Fig. 11, where each side has 9 sensors. The distance between consecutive sensors is around 3 meters as before, and the sink is placed around the corner. We collect 706 samples.

It is worth mentioning that this scenario slightly deviates from our considered scenario. In fact, the sensor network here consists of two 1-D subnetworks, which join at the right angle turn. Sensors in one subnetwork no longer lie in the same direction of any sensor in the other subnetwork. Therefore, the two candidate solutions in Section 4 do not work in this scenario. However, we can check that our observation can be used approximately, since it only requires about the closest node.

Fig. 12 shows that our algorithm is still very effective in this scenario. Without smoothing, the reliability is about 87% and the accuracy is about 98.5%, which is slightly worse than that of Scenario 1. This is natural due to the turn. However, smoothing only two consecutive samples improves reliability to 99.5% and accuracy to 100%.

Similar to Scenario 1, applying Algorithm 1 to the collected data yields no correct ordering due to the existence of propagation holes defined in Section 7.1.1. These holes exist even if we smooth more than 50 consecutive samples.

7.1.3. Scenario 3: S-shaped road

Due to lack of S-shaped roads in campus, we deploy sensors in a sidewalk following an S-shape as in Fig. 13 to simulate an S-shaped road. Trees are around and the inter-node distance varies from 1 meter to 3 meters depending on the location. We collect 706 samples.

This scenario is to explore the potential of our algorithm. It is more challenging than Scenario 2, because we even cannot divide the network into non-trivial 1-D subnetworks. Most nodes in this scenario are not in the same direction from any other node. Instead, they are at different directions from a sender. In different directions, signals propagate differently, which has already been established [22]. We use this scenario to show the robustness of our algorithm.

Fig. 14 shows degraded performance of our algorithm in terms of reliability. The reliability without smoothing is about 60%, much less than that of the previous scenarios. Smoothing two consecutive samples improves it to about 75%, and smoothing more samples cannot further improve it. This fact indicates that many consecutive samples consistently contain undesired data. Fortunately, the accuracy is over 99% without smoothing and is 100% if only two consecutive samples are smoothed. As a result, in this tough scenario, our algorithm can still work, but it needs more time to collect more samples, compared to the previous two scenarios.

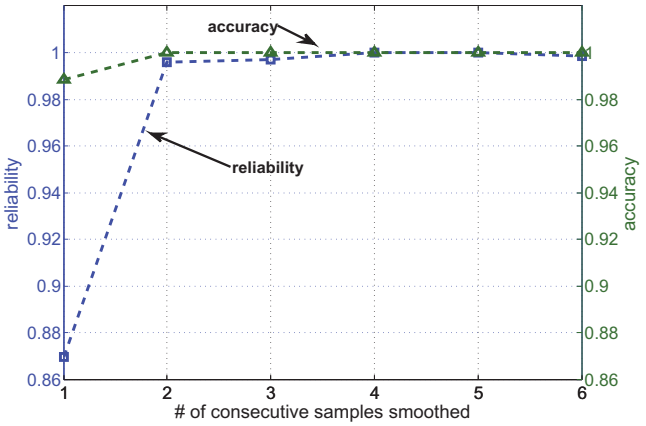


Fig. 12. Performance in Scenario 2.

Fig. 11. Sensor deployment along an L-shaped road.

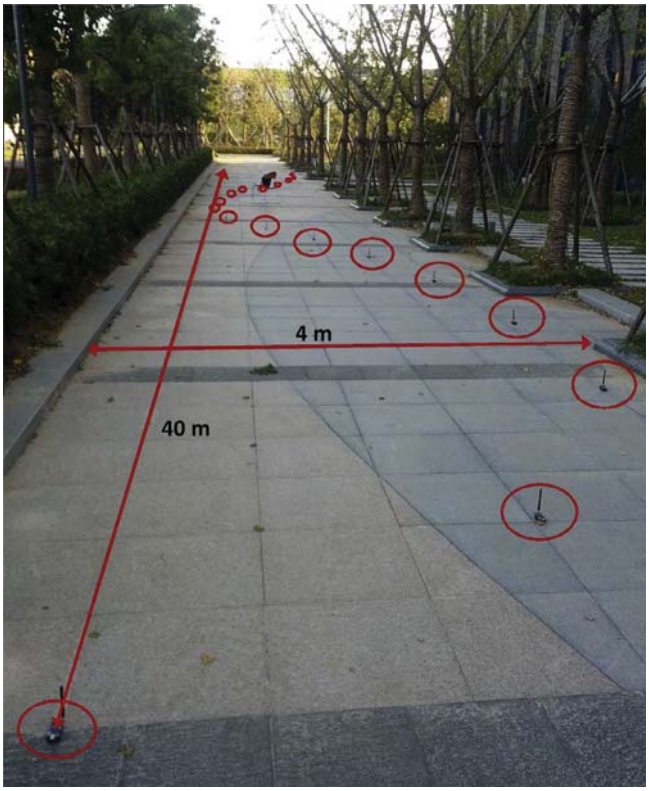


Fig. 13. Sensor deployment in an S-shape.

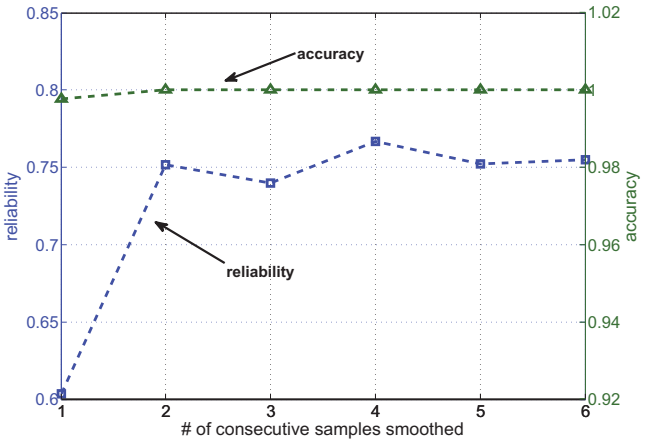


Fig. 14. Performance in Scenario 3.

It is worth mentioning that Algorithm 1 does not work well in this experiment either. When smoothing is not performed, propagation holes exist in all samples as before. Though performing smoothing gives some smoothed samples that do not contain propagation holes, we find that in each of these smoothed samples, more than 15 nodes are indistinguishable by connectivity information because each of them is a neighbor of all the rest 17 nodes.

7.1.4. Remarks

The reliability of our algorithm decreases when the road is not straight, but the accuracy remains over 99%. This suggests that our algorithm only needs more samples to output an ordering for adapting to such non-straight road environment. In addition, smoothing several consecutive samples can improve both reliability and accuracy, where smoothing two or three samples gives the most significant improvement.

7.2. Robustness of our solution

The previous experiments have shown that road shape can influence the performance of our solution. In this subsection, we study the impact of three additional factors including inter-node distance, obstacle, and wireless interference. The experiments are conducted using 10 normal sensor nodes deployed in the same way as Fig. 8. To collect samples faster, we increase the probing rate of sensors to be 2 probes/s so that we can get 2 samples per second (it is 0.1 sample/s in the previous experiments). When the program collects more than 500 samples, we manually terminate the process.

7.2.1. Impact of inter-node distance

We vary inter-node distance from 1.5 m to 4.5 m with an increment of 1.5 m. The result is shown in Table 2. Regarding reliability, we have three observations. First, when inter-node distance is 3 m, the reliability without smoothing is lower than that of the experiment in Section 7.1.1. We believe this is caused by collision of probe messages due to increased probing rate. Second, without smoothing, the reliability decreases when the inter-node distance becomes either smaller or larger. Considering that it remains very similar if every two samples are smoothed, we believe the decreasing of reliability is caused by packet loss. Third, smoothing only two consecutive samples greatly improves the reliability as before. In all three scenarios we consistently obtain a high accuracy of over 99%.

Table 2
Performance (*reliability*, *accuracy*) of our solution under different inter-node distances. The network consists of 10 nodes with probing rate 2 probes/s. Both reliability and accuracy have a range of [0,1], and the higher the value is, the better the performance is.

Dist (m)	# of samples	# of consecutive samples smoothed		
		One (w/o smoothing)	Two	Three
1.5	690	(0.6145, 0.9953)	(0.9840, 0.9985)	(1.0, 1.0)
3.0	1037	(0.7917, 0.9988)	(0.9942, 1.0)	(1.0, 1.0)
4.5	685	(0.6993, 1.0)	(0.9678, 1.0)	(0.9854, 1.0)

7.2.2. Impact of obstacle

The obstacle in this experiment is a person, who will stand at three places to block the line-of-sight signal between nodes. The nodes are spaced by 3 meters, and the three places for obstacle include the midpoint between nodes 1 and 2, the midpoint between nodes 3 and 4, and the midpoint between nodes 5 and 6. Note that human body can block the line-of-sight signal, which is actually the foundation of device-free localization techniques using sensor networks [23]. We also observe this effect in our experiment. Fig. 15 shows node 2's RSSI of signals sent from node 1 when the man stands at the three different locations. It is obvious that when the man stands between 1 and 2, the observed RSSI fluctuates more frequently and has a smaller mean value. The performance of our solution in deriving nodes' ordering is summarized in Table 3. Again, we observe high accuracy in this setting, and it is over 99% in all three scenarios. The reliability becomes lower when the man stands closer to the network center (midpoint between 5 and 6). This is natural since standing at the network center blocks the most number of line-of-sight links.

7.2.3. Impact of wireless interference

The setting here is the same as that in Section 7.2.2, except that we replace the person by a jammer, a sensor node that broadcasts jamming messages at a rate of 4 packets/s on the same channel as the normal sensor nodes. We place the jammer at the same locations where the person stands in Section 7.2.2, and study the performance of our solution based on the collected data. Table 4 shows that reliability changes with respect to the jammer location, while the accuracy remains over 99% in all three scenarios without smoothing. Different from the obstacle experiment, when the jammer moves from the midpoint between 3 and 4 to the midpoint between 5 and 6, the reliability does not change much. This is

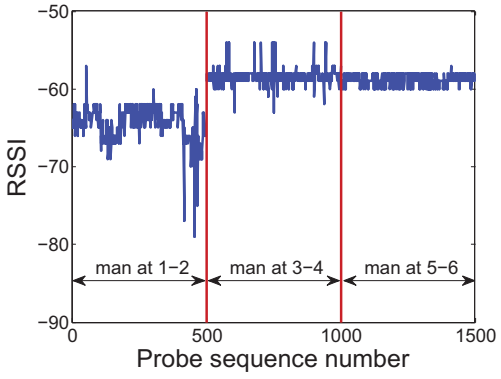


Fig. 15. Influence of human body on the observed RSSI at node 2 for signals sent from node 1. Nodes labeled 1 to 10 are deployed sequentially along a straight road with a spacing of 3 meters. We can see that when the man stands at the midpoint between nodes 1 and 2, the RSSI has a smaller mean value.

Table 3
Performance (*reliability*, *accuracy*) of our solution under different placements of an obstacle (man). Placement "1–2" means the obstacle is at the midpoint between nodes 1 and 2.

Placement	# of samples	# of consecutive samples smoothed		
		One (w/o smoothing)	Two	Three
1–2	952	(0.8172, 1.0)	(0.9937, 1.0)	(0.9958, 1.0)
3–4	501	(0.7665, 0.9974)	(0.8540, 1.0)	(0.8778, 1.0)
5–6	506	(0.6759, 0.9971)	(0.7644, 1.0)	(0.7798, 1.0)

Table 4
Performance (*reliability*, *accuracy*) of our solution under different placements of a jammer. The jammer is a sensor node that actively broadcasts jamming messages at the same channel. Placement "1–2" means the jammer is at the midpoint between nodes 1 and 2.

Placement	# of samples	# of consecutive samples smoothed		
		One (w/o smoothing)	Two	Three
1–2	549	(0.7596, 1.0)	(0.9945, 1.0)	(1.0, 1.0)
3–4	581	(0.6575, 1.0)	(0.9828, 1.0)	(1.0, 1.0)
5–6	579	(0.6736, 0.9923)	(0.9862, 1.0)	(0.9965, 1.0)

because, when the jammer is between 3 and 4, it can already jam the whole network so that the performance for the two locations (3–4 and 5–6) is very similar. Jamming leads to increased packet loss, which can be compensated by collecting more samples.

7.2.4. Remarks

We considered the influence of three factors on our algorithm. In all cases, we achieve high accuracy over 99%, but the reliability varies from 61% to 82% when smoothing is not performed, and is over 70% when every three samples are smoothed. In cases where reliability is low, we need to collect more samples to obtain an ordering.

8. Conclusions

In this paper, we study the problem of inferring spatial ordering of sensor nodes. We examine two candidate solutions developed from existing ideas, with one assuming that nodes can hear from each other if and only if they are within transmission range, and the other assuming closer nodes observe larger RSSI. Both candidate solutions do not work well in practice. After changing "closer" to "the closest" and "larger" to "the largest" in the second approach, we find that the new assumption is quite reliable in practice. We then design an efficient algorithm to derive the ordering of nodes. We conduct real-world experiments to study the performance of our algorithm in terms of reliability and accuracy, where reliability measures the percentage of samples yielding non-*null* answers among all samples, and accuracy measures the percentage of correct answers among non-*null* answers. We study various factors including road shape, inter-node distance, obstacle and wireless interference. In all experiments we conduced, the reliability is over 60% with an accuracy over 99%.

Acknowledgment

The authors would like to thank all the reviewers for their helpful comments. The work is partly supported by the program B for Outstanding PhD candidate of Nanjing University (201301B014), China 973 project (2012CB316200) and China NSF grant (61133006, 60903179, 61021062).

References

[1] X. Zhu, G. Chen, Spatial ordering derivation for one-dimensional wireless sensor networks, in: Proceedings of IEEE ISPA, 2011.
[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, Wireless sensor networks: a survey, Computer Networks 38 (4) (2002) 393–422.
[3] J. Hightower, G. Borriello, Location systems for ubiquitous computing, Computer 34 (8) (2001) 57–66, http://dx.doi.org/10.1109/2.940014.
[4] S. Kim, S. Pakzad, D.E. Culler, J. Demmel, G. Fenves, S. Glaser, M. Turon, Health monitoring of civil infrastructures using wireless sensor networks, in: Proceedings of ACM IPSN, 2007.
[5] TinyOS. <http://www.tinyos.net>.
[6] Y. Liu, Z. Yang, X. Wang, L. Jian, Location, localization, and localizability, Journal of Computer Science and Technology 25 (2) (2010) 274–297.
[7] X. Zhu, X. Wu, G. Chen, Refining hop-count for localisation in wireless sensor networks, International Journal of Sensor Networks 12 (4) (2012) 232–243.
[8] L. Doherty, K. pister, L. El Ghaoui, Convex position estimation in wireless sensor networks, in: Proceedings of IEEE INFOCOM, 2001.
[9] A. Karbasi, S. Oh, Distributed sensor network localization from local connectivity: performance analysis for the hop-terrain algorithm, in: Proceedings of ACM SIGMETRICS, 2010.
[10] X. Wang, J. Luo, Y. Liu, S. Li, D. Dong, Component-based localization in sparse wireless networks, IEEE/ACM Transactions on Networking 19 (2) (2011) 540–548.
[11] Y. Shang, W. Ruml, Y. Zhang, M.P. Fromherz, Localization from mere connectivity, in: Proceedings of ACM MobiHoc, 2003.
[12] R. Bischoff, R. Wattenhofer, Analyzing connectivity-based multi-hop ad-hoc positioning, in: Proceedings of IEEE PerCom, 2004.
[13] Z. Lotker, M.M. de Albeniz, S. Perennes, Range-free ranking in sensors networks and its applications to localization, in: ADHOC-NOW, 2004.
[14] Z. Guo, Y. Guo, F. Hong, X. Yang, Y. He, Y. Feng, Y. Liu, Perpendicular intersection: Locating wireless sensors with mobile beacon, in: Proceedings of IEEE RTSS, 2008.
[15] C. Liu, K. Wu, T. He, Sensor localization with ring overlapping based on comparison of received signal strength indicator, in: Proceedings of IEEE MASS, 2004.
[16] Z. Zhong, T. He, Achieving range-free localization beyond connectivity, in: Proceedings of ACM SenSys, 2009.
[17] D. Zhang, Y. Liu, X. Guo, M. Gao, L.M. Ni, On distinguishing the multiple radio paths in rss-based ranging, in: Proceedings of IEEE INFOCOM, 2012.
[18] D.G. Corneil, A simple 3-sweep lbfs algorithm for the recognition of unit interval graphs, Discrete Applied Mathematics 138 (3) (2004) 371–379.
[19] M.L. Sichitiu, V. Ramadurai, Localization of wireless sensor networks with a mobile beacon, in: Proceedings of IEEE MASS, 2004.
[20] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman & Co., 1979.
[21] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, P. Levis, Collection tree protocol, in: Proceedings of ACM SenSys, 2009.
[22] T. He, C. Huang, B.M. Blum, J.A. Stankovic, T. Abdelzaher, Range-free localization schemes for large scale sensor networks, in: Proceedings of ACM MobiCom, 2003.
[23] J. Wilson, N. Patwari, Radio tomographic imaging with wireless networks, IEEE Transactions on Mobile Computing 9 (5) (2010) 621–632.