Locating Nodes In An SINR Network

Stephan Holzer\textsuperscript{1} and Evangelia Anna Markatou\textsuperscript{2}

\textsuperscript{1} MIT, Cambridge, MA USA
holzer@mit.edu

\textsuperscript{2} MIT, Cambridge, MA USA
markatou@mit.edu

Abstract

The problem of locating nodes on a plane has been extensively studied by the networking community, and numerous systems, like the GPS and indoor navigation systems, are used extensively everyday by millions of people. In this paper, we formally study this problem in a single-hop network in the Signal to Interference plus Noise Ratio (SINR) model, and present an algorithm that can solve it along with communication complexity bounds. We present a randomized algorithm that approximates the location of all nodes within a \((1 + \epsilon)\) factor of the node coordinates, for arbitrary \(0 < \epsilon < 1\).

As part of this algorithm we develop an \(n\)-message broadcast algorithm. Using uniform power our algorithm matches the best known result \([11]\) at \(O(n + \log n)\) rounds, and using power control, we achieve \(O(n)\) rounds, where \(n\) is the number of nodes in the network.

1998 ACM Subject Classification C.2.1,G.1.0

Keywords and phrases SINR, node locating, broadcast, leader election, collision detection

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23
1 Introduction

In this paper we explore the problem of locating the nodes in an SINR network. Locating nodes on a two dimensional space has traditionally been a very popular problem in the networking/telecommunications community [5]. There currently exist widely-used systems, such as GPS [13], that help us locate nodes anywhere on earth.

We explore the problem of locating nodes on the plane from a theoretical distributed systems point of view. Contrary to previous work we do not assume that we have any known access points or any other kind of infrastructure with which the nodes can communicate. We achieve a solution using the limited capabilities of the SINR model. We present a randomized distributed algorithm for determining the location of nodes in a single-hop network under the Signal to Interference plus Noise Ratio (SINR) model, using power control, the ability of nodes to transmit with varying transmission power. In the 'heart' of our approach is a broadcast algorithm that helps anchor three nodes in the plane and instruct every node, in turn, to find its distance from the anchored nodes by adjusting its transmission power.

Our goal is to introduce the node locating problem in the SINR model and provide a solution along with specific complexity bounds. We do not aim to provide an optimal practical algorithm, but to study a problem of theoretical interest and prove that it can be solved.

The structure of this paper is as follows: In Section 2 we formally define the model and the problem statements and in Section 3 we present the related work. Our algorithm consists of 3 parts: (i) a leader election algorithm, (ii) an \( n \)-message broadcast and (iii) a routine that locates all nodes. They are presented in Sections 4, 5 and 6 respectively. We present our conclusions and some open problems in Section 7.

2 Model and Problem Statements

In this section, we present the SINR model and the problems we are considering.

Definition 1. SINR Model

Let \( V \) be a set of \( n \) nodes, that represent wireless devices, deployed in a two-dimensional Euclidian space. Time is divided in synchronous rounds. In each round, a node \( u \) can either transmit a message of at most \( O(\log n) \) bits with transmission power \( P_u \in \mathbb{R}_{>0} \) or listen for another node’s message. Node \( v \in V \) receives a message transmitted by node \( u \in V \), in a round when all nodes in \( I \) also transmit, where \( I \subseteq V \setminus \{u,v\} \), iff \( v \) is listening and:

\[
SINR(u, v, I) = \frac{P_u}{d(u,v)\alpha} N_c + \sum_{w \in I} \frac{P_w}{d(w,v)^\alpha} \geq \beta,
\]

where \( N_c \) is a universal constant denoting ambient noise. The Euclidean distance between two nodes \( u \) and \( v \) is denoted by \( d(u,v) \). The parameter \( \alpha \), typically \( \alpha \in (2, 6] \) [7], is the path loss exponent, and \( \beta > 1 \) denotes the minimum SINR required for a message to be received. \( R \) is the ratio between the longest and shortest distance between two nodes in the network.

We assume \( N_c \), \( \alpha \) and \( \beta \) are known. Similar to [6], we assume that \( R \) is bounded by a polynomial in \( n \). We assume a single-hop network, where any pair of nodes can effectively communicate with transmission power \( P \), assuming no interference from other nodes. The maximum power with which a node might transmit is \( P_{\text{max}} \). Unless stated otherwise, the nodes broadcast with transmission power \( P \). The minimum distance between any two nodes in the network is 1. We say that an event happens with high probability (w.h.p.) if it occurs
with probability greater than $1 - \frac{1}{n}$. We present a number of algorithms, some require an upper bound $N$ of $n$, or a certain power range. We assume $N = \text{poly}(n)$. Some of our algorithms also require collision detection: when a node is listening, it can distinguish whether no nodes are broadcasting, one node is broadcasting or at least two nodes are broadcasting.

In this paper we discuss four problems: (i) the problem of locating all nodes in a network, (ii) the problem of broadcasting $n$ distinct messages stored in $n$ different nodes to all the nodes in the network, (iii) the contention resolution problem, and (iv) the leader election problem. We developed new algorithms for problems (i) and (ii), and propose a modification on the contention resolution algorithm presented in [6], to generate a leader election algorithm.

\begin{itemize}
  \item \textbf{Definition 2.} Locating the nodes in a network. Given $n$ nodes in a two-dimensional space, every node $v \in V$ $(1 + \epsilon)$-approximates and broadcasts its coordinates $(x,y)$, in reference to a common origin, for any choice of $\epsilon \in (0, 1)$.
  
  \item \textbf{Definition 3.} $n$-Message Broadcast. Given $n$ distinct messages stored in $n$ different nodes in a network consisting of $n$ nodes, ensure that all $n$ nodes receive all $n$ messages.
  
  \item \textbf{Definition 4.} Leader Election. Given $n$ nodes in a single-hop SINR network, eventually elect exactly one leader node, with all nodes knowing whether or not they were elected.
  
  \item \textbf{Definition 5.} Contention Resolution. The contention resolution problem assumes $n$ nodes in an SINR network. The problem is solved in the first round in which exactly one node transmits.
\end{itemize}

3 Related Work and Our Results

In this paper we focus on three problems: (i) Leader Election, (ii) Broadcast, and (iii) Node Locating. We develop a broadcast algorithm that is based on a leader election routine, and a node locating algorithm that depends on the broadcast algorithm.

3.1 Leader Election

The leader election problem has many variations, but its essence lies in: given $n$ nodes, elect exactly one as a leader. In this paper, we study the version where we require that everyone in the network knows whether or not they are the leader. We also consider \textit{contention resolution}, which is the problem of having exactly one node broadcast in a round in an SINR network.

Plenty of papers concerning leader election and its many variations applied on multiple-access channel (MAC) models have been published since the ALOHA protocol [1], e.g., [3], and [16]. The most efficient published contention resolution routine in the SINR model using uniform power is by Fineman et al.[6]. They present an algorithm that solves contention resolution in $O(\log n + \log R)$ rounds w.h.p. in a single-hop network. Using power control Halldórsson et al. [9] developed an algorithm that achieves leader election in the SINR model in constant communication rounds.

3.2 Broadcast

We consider the problem of wireless broadcast of $n$ messages from $n$ nodes to $n$ nodes. There have been plenty of papers on this topic concerning both (i) global broadcast and (ii) local broadcast. Global broadcast refers to the broadcast of messages to the whole network, and local broadcast refers to the broadcast of messages to the nodes in the broadcasting range of
Locating Nodes In An SINR Network

the sender. The broadcasting range of node $x$ is the maximum distance from which it can receive a clear transmission, assuming no other transmission occurs.

The first paper on global broadcast on radio networks was by Bar-Yehuda et al. [2]. They described an algorithm that works in $O(k \log n \log \Delta + (D + n/\log n) \log n \log \Delta)$ rounds in expectation, where $k$ is the number of messages to be delivered in all $n$ nodes, $D$ is the network diameter and $\Delta$ is the maximum node degree. In the SINR model, Yu et al. [18] describe an algorithm that works in $O((D + k) \log n + \log^2 n)$ rounds w.h.p., where $D$ is the network diameter.

Local broadcast in the SINR model was first studied by Goussevskaia et al. in [8]. The paper presents a randomized asynchronous algorithm that achieves local broadcast in $O(\Delta \log^3 n)$ rounds. There are multiple results in this problem under a variety of assumptions. In the SINR model, local broadcast can be achieved in $O(\Delta \log n)$ [7] with knowledge of $\Delta$ and without it, in $O(\Delta \log n + \log^2 n)$ [10]. Halldórsson et al. [11] present a randomized algorithm that achieves local broadcast within $O(\Delta + \log n)$ rounds w.h.p. using collision detection. Yu et al. [17] present a lower bound for local broadcast of $\Omega(\Delta + \log n)$ w.h.p.

Our Results: We present two broadcast algorithms on a single hop network.

- **Broadcast in $O(n + \log N)$ rounds**: This algorithm matches the result from [11]. We assume collision detection and an upper bound $N$ of $n$.
- **Broadcast in $O(n)$ rounds**: This algorithm requires power control with a power range of at least $2^{O(n)}$, and collision detection. For power ranges $P_{\text{range}} \in [2^{n \log \log n}, 2^{O(n)}]$, our broadcast algorithm terminates in $O(\frac{\log n}{\log \log P_{\text{range}}})$ rounds w.h.p., where $P_{\text{range}}$ is the broadcasting range of node $x$. Both results work with probability greater than $1 - 2/n$.

### 3.3 Node Locating

The problem of determining the location of nodes in a wireless setting has been extensively studied by the networking community. A widely-used example is the Global Positioning System (GPS) [13], which is a centralized system. There are many more results from the practical wireless network community, [12], [15], [19].

We work on determining the position of participants in a theoretical distributed setting. There are a number of interesting networking results here as well, e.g., [4], [14]. Čapkun et al. [4] present a geometric argument for approximating the location of nodes, but determine the distance between two nodes using time based techniques. We assume synchronous rounds, so we don’t have access to such measurements. Niculescu et al. [14] use more measurements than we make available, like the direction from which the message was received. To our knowledge, there have been no results approximating the position of the nodes in a theoretical SINR network.

Our Results: We present algorithms that approximate the location of all nodes in the network with arbitrarily high precision depending on a parameter $0 < \epsilon < 1$. More specifically, if a node’s position coordinates are $(x, y)$, we approximate $(x', y')$, such that $\sqrt{(x - x')^2 + (y - y')^2} \leq \epsilon$.

- **$O(n(\log \log P + \log \frac{P}{\epsilon^2}) + n + \log N)$ rounds**: We assume collision detection and an upper bound $N$ of $n$. This algorithm works with probability greater than $1 - 2/n$.
- **$O(n(\log \log P + \log \frac{P}{\epsilon^2}) + n)$ rounds**: Here, we assume collision detection and power control with a power range of at least $2^{O(n)}$. For power ranges $P_{\text{range}} \in [2^{n \log \log n}, 2^{O(n)}]$, the
algorithm terminates in \( O(n(\log \log P + \log \frac{R}{\epsilon}) + n + \frac{\log n}{\log \log P_{\text{range}}}) \). This algorithm works with probability greater than \( 1 - 2/\sqrt{n} \).

\[ O(n(\log n + \log \log P + \log \frac{R}{\epsilon}) + \log^2 n + \log N) \text{ rounds: Here, we assume an upper bound } N \text{ of } n. \] This algorithm works with probability greater than \( 1 - 3/\sqrt{n} \).

\section{Leader Election}

\subsection{Leader Election in \( O(\log N) \) Rounds}

The contention resolution algorithm presented in \cite{6} guarantees that exactly one node \( v \) will transmit and all other nodes will listen to \( v \)'s message after \( O(\log n) \) rounds. However, the algorithm does not guarantee that node \( v \) will know that it is the only one transmitting. In this section, we modify the algorithm presented in \cite{6} to guarantee that all nodes know whether or not they have been elected as the leader. The original algorithm \cite{6} is:

Each participating node starts in an active state; at the beginning of each round, each node that is still active broadcasts with a fixed constant probability \( p \); if an active node receives a message, it becomes inactive.

Fineman et al. \cite{6}'s algorithm does not depend on knowledge of \( n \), as it does not terminate. In order to ensure termination of the algorithm, we require an upper bound \( N \) of \( n \).

In our leader election algorithm, the nodes run the contention resolution algorithm for \( 2c \log N \) rounds. If node \( v \) is active after \( 2c \log N \) rounds, then w.h.p. it is the leader.

\begin{theorem}
The Leader Election algorithm elects a leader w.h.p. in \( O(\log N) \) rounds.
\end{theorem}

The theorem follows from Theorem 11 of \cite{6} that guarantees that after \( n \) nodes run the contention resolution algorithm \cite{6} for \( c \log n \) rounds there is only one active node left w.h.p..

\subsection{Faster Leader Election in \( O(1) \) or \( O\left(\frac{\log n}{\log \log P_{\text{max}} \epsilon}\right) \) Rounds}

Halldórsson et al. \cite{9} present an optimal leader election algorithm, which, given a power range \( \exp(n^{1/\Theta(1)}) \), can elect a leader in \( t \) rounds, for \( t = O(\log n/\log \log n) \), without knowledge of (an upper bound of) \( n \).

Since our node locating algorithm uses power control, we can use the power range that the nodes already have (\( P_{\text{max}} \)) to achieve faster leader election.

\begin{theorem}
If \( P_{\text{max}} > 2^{n^\left(\frac{\log \log n}{\log \log P_{\text{max}} \epsilon}\right)} \), we can achieve leader election in
\[ O\left(\frac{\log n}{\log \log P_{\text{max}} \epsilon}\right) \]
rounds w.h.p..
\end{theorem}

\begin{remark}
If \( P_{\text{max}} = 2^{O(n)} \), leader election is achievable in \( O(1) \) rounds.
\end{remark}

\section{Broadcast Algorithm in \( O(n) \) rounds}

Now, that we have the tools to elect a leader, we proceed to describe our broadcast algorithm.

On a high level, a leader node constructs a binary tree, and arranges the other nodes on it such that every leaf of the tree has at most one node in it. The leader node traverses the
Locating Nodes In An SINR Network

First, the nodes run a leader election algorithm to elect leader node $v$. Then, the leader node $v$ constructs a tree, initially containing just the root. All nodes $u, u \in V - \{v\}$ start on the root node. In each round, the leader node visits a tree node, and if there are more than one nodes on it, it orders the nodes to drop down the tree, creating two new tree nodes in the process. The leader visits all the tree nodes performing a depth-first search while the tree is being constructed, giving each node a chance to broadcast once it’s alone on a leaf. Below, we describe the algorithm in more detail:

Each round is split in 3 timeslots. The leader node keeps a list of all the tree nodes it has not visited yet, initially containing just the root node, and updating every time a new node is created. In every round, node $v$ visits a new tree node $t$ and removes it from the list.

Leader node $v$ follows this protocol, visiting a new tree node each round until the list is empty:

**Timeslot 1:** The leader node $v$ visits a new tree node $t$ from its list of unvisited nodes and broadcasts a message asking if there are any nodes on $t$.

**Timeslot 2:** The nodes respond whether or not they are on tree node $t$.

**Timeslot 3:** The leader node $v$ acts depending on the message it received in the previous timeslot.

(i) If $v$ received silence, node $v$ does nothing.
(ii) If $v$ received a message containing an ID, node $v$ assigns a priority to the node that send the message.
(iii) If $v$ heard more than one node broadcast, $v$ orders the nodes that are on tree node $t$ to move further down the tree. The nodes then flip a fair coin individually, and randomly decide on which of the two new branches they want to go, left or right. Node $v$ then adds the new tree nodes in its list of unvisited nodes.

### 5.1 Broadcast Algorithm Pseudocode

**Algorithm 1** Broadcast $O(n)$

1: $ID = $ a uniformly at random selected integer from $[1, N^4]$
2: Run a Leader Election Algorithm.
3: if Elected Leader then Run Algorithm 2
4: else Run Algorithm 3
5: end if

### 5.2 Analysis of Broadcast Algorithm

We show that the Broadcast Algorithm gives every node an opportunity to broadcast its message while all other nodes are listening.

**Theorem 8.** The Broadcast Algorithm ensures that all nodes in an SINR network broadcast their message and acquire a unique priority in $[0, n − 1]$ with probability greater than $1 − 1/n − 1/N^6$.

**Proof.** The Broadcast Algorithm runs a leader election protocol, and then the elected leader node $v$ facilitates the broadcast. First, the nodes pick an ID. These IDs are unique with
Algorithm 2 Broadcast as executed from the leader node $v$

1: $p = 1$, priority = 0
2: $UnvisitedTreeNodes = [root]$, a list initially containing just root
3: while $UnvisitedTreeNodes$ is not empty do (on every round)
   4:   Timeslot 1:
   5:   $currentTreeNode = pop$ the first element from list $UnvisitedTreeNodes$
   6:   Broadcast "Are you on tree node $currentTreeNode$?"
   7:   Timeslot 2: Listen.
   8:   Timeslot 3:
   9:   if the leader node receives a message containing a number $z$ then
   10:      Broadcast "The node identified by $z$ gets priority $p$" and set $p = p + 1$
   11:   else if the leader node receives noise (more than one nodes are broadcasting) then
   12:      Broadcast "Nodes on $currentTreeNode$ flip a coin and go down the tree"
   13:      Add $concatenate(currentTreeNode, left), concatenate(currentTreeNode, right)$ in the list $UnvisitedTreeNodes$
   14:   else
   15:      Do nothing
   16:   end if
   17: end while

Algorithm 3 Broadcast as executed from non leader node $u$

1: priority = $\emptyset$, $treeNode = root$
2: while priority = $\emptyset$ do (on every round)
3:   Timeslot 1: Listen.
4:   Timeslot 2:
5:   if $u$ received message on first timeslot containing $treeNode$ then
6:      Broadcast "I am here and my identification number is ID"
7:   end if
8:   Timeslot 3: Listen
9:   if $u$ receives message "The node identified by $ID$ gets priority $p$" and it sent a message in the previous timeslot then priority = $p$
10:  if $u$ receives "Nodes on $treeNode$ flip a coin and go down the tree" then
11:    $treeNode = concatenate(treeNode, r)$, $r$ is a random element of $[left, right]$
12:  end if
13: end while

probability greater than $1 - 1/N^6$. Leader election finishes in $O(\log N)$ or $O(\frac{\log n}{\log \log \frac{\text{max}}{\text{min}}})$ rounds depending on the leader election algorithm used, w.h.p.. The leader node $v$ then constructs a tree, initially placing all nodes $\in V - \{v\}$ on the single node of the tree, the root node. Node $v$ then visits every tree node $t$, asking the nodes if they are on tree node $t$. We assume collision detection and thus $v$ can recognize if there is one, more than one, or no nodes on that tree node.

If the leader determines that there are more than one nodes on $t$, it instructs the nodes to move down the tree, each time picking between two new branches, until each node $\in V - \{v\}$ is on a leaf alone. If the leader node $v$ determines that $t$ has a single node $u$ on it, $v$ lets node $u$ broadcast its message, when all other nodes are listening, and then it assigns a priority to it. We conclude that the Broadcast Algorithm terminates after all nodes in the network get a chance to broadcast their message with probability greater than $1 - 1/n - 1/N^6$. ◼
**Theorem 9.** The Broadcast Algorithm terminates successfully in $O(n + \log N)$ or $O(n + \frac{\log n}{\log \log n})$ rounds with probability greater than $1 - 2/n$.

**Proof.** The Broadcast Algorithm terminates after a leader is elected and all nodes have an assigned priority, which is when the leader node $v$ finishes its while loop. The while loop terminates when $v$ has visited all the tree nodes. Thus, the time complexity of the Broadcast Algorithm is the time it takes to elect a leader plus a number of rounds proportional to the size of the tree. Leader election finishes in $O(\log N)$ or $O(\frac{\log n}{\log \log n})$ rounds depending on the leader election algorithm used.

In the rest of this proof we focus on calculating the size of the tree. We prove that with probability greater than $1 - 1/n^2$ the size of the tree is smaller than $10n$. In this proof we ignore any leaves of the tree that contain no nodes. These leaves can at most double the size of the tree, so they are asymptotically insignificant.

The size of a binary tree that has $n - 1$ leaves, is $2(n - 1)$. The tree constructed in the Broadcast Algorithm, which has $n - 1$ leaves, might have some nodes with only one, as opposed to two, successors, and thus its size is greater or equal to a binary tree with the same number of leaves. Let $X_i$ be a random variable representing the length of a chain from an inner node $i$ (not a leaf of the tree) whose parent has two successors, to its first descendant that has two children. There are $n - 1$ $X_i$’s. In Figure 1, we present two examples of how $X_i$ is calculated.

The size of the tree constructed in the Broadcast Algorithm is the sum of the random variables $X_i$, plus the number of the leaves, $n - 1 + \sum_{i=0}^{n-1} X_i$. The maximum value that a variable $X_i$ can achieve occurs when only two nodes are on node $i$, as the larger the number of nodes on $i$, the smaller the probability that all of them will pick the same child node to go to. Let $X'_i$ be a random variable that takes $X_i$’s value when tree node $i$ has two nodes on it. Note that $Pr[\sum_{i=0}^{n-1} X_i > k] \leq Pr[\sum_{i=0}^{n-1} X'_i > k]$.

Let $X$ be the sum of all $X'_i$ variables. Let’s calculate the probability that $X = k \cdot n$. Let $1, 2, \ldots, n - 1$ represent the inner nodes of the tree. The probability that tree node $i$ is on the top of a chain of length $l_i$ is $1/2^{l_i}$, as the two nodes need to choose to go to the same child tree node, which happens with probability $1/2$, $l_i$ times. Thus, $Pr[X = k \cdot n]$ is equal to the sum of $1/2^{l_1} \cdot 1/2^{l_2} \cdot \ldots \cdot 1/2^{l_{n-1}}$ over all the possible ways to arrange $k \cdot n$ tree nodes in $n - 1$ chains, $Pr[X = k \cdot n] = \sum_{l_1+l_2+\ldots+l_{n-1}=kn} \frac{1}{2^{l_1}} \cdot \frac{1}{2^{l_2}} \cdot \ldots \cdot \frac{1}{2^{l_{n-1}}} = \sum_{l_1+l_2+\ldots+l_{n-1}=kn} \frac{1}{2^{\sum_{i=1}^{n-1} l_i}}$.

There are less than $\binom{k+n-2}{n-2}$ ways to partition $k \cdot n$ elements in $n - 1$ groups. We bound $\binom{k+n-2}{n-2} \leq \left(\frac{e((k+1)(n-2))}{n-2}\right)^n \leq \left(\frac{e((k+1)n-2)}{n-2}\right)^n \leq \left(\frac{e((k+1)n-2)}{2k+1}\right)^n$, for large enough $n$. Thus, $Pr[X = k \cdot n] \leq \left(\frac{e((2k+1)n)}{(2k+1)}\right)^n$. This is a monotonically decreasing function on $k$, which allows us to bound $Pr[k \cdot n \leq X < (k + 1) \cdot n] \leq n \left(\frac{e((2k+1)n)}{(2k+1)}\right)^n$. Now, let’s calculate the probability that $Pr[X > 10n]$, for large enough $n$. We get that $Pr[X > 10n] = \sum_{i=10n}^{\infty} Pr[X = i] = \sum_{k=10}^{\infty} Pr[k \cdot n \leq X < (k + 1) \cdot n] = \sum_{k=10}^{\infty} n Pr[X = k \cdot n] \leq \sum_{k=10}^{\infty} n \left(\frac{e((2k+1)n)}{(2k+1)}\right)^n \leq ne^{n(0.045n)} \leq n(0.12)^n$. 

![Figure 1](image_url) Here, $X_i = 1$ and $X_k = 2$. 

- **Figure 1** Here, $X_i = 1$ and $X_k = 2$. 

23:8 Locating Nodes In An SINR Network
The size of the binary tree is smaller than $10n + (n - 1)$ with probability greater than $1 - 1/n^2$, as $1 - n(0.12)^n > 1 - 1/n^2$. The algorithm concludes successfully with probability greater than $1 - 1/n - 1/N^6$ from Theorem 8. We use a union bound to calculate the probability that the broadcast algorithm terminates as desired.

We conclude that the Broadcast Algorithm terminates in $O(10n + (n - 1) + \log N) = O(n + \log N)$, or $O(n + \log n \log \log P_{\max})$ rounds with probability greater than $1 - 2/n$. ◀

Remark. Note that when $P_{\max} = 2^{O(n)}$, the Broadcast Algorithm terminates in $O(n)$ rounds with probability $1 - 2/n$.

In the following subsection, we present an alternative broadcasting algorithm that does not require collision detection.

5.3 Broadcast Without Collision Detection

If the nodes do not have collision detection, they can use the broadcast algorithm from [10]. This Alternative Broadcast Algorithm achieves time complexity $O(n \log n + \log^2 n)$.

In more detail, the nodes use a leader election algorithm to elect a leader node $v$, and pick an ID uniformly at random from $[1, N^4]$. Each round is split in two timeslots. In the first timeslot the nodes run the broadcast algorithm from [10], and broadcast a message consisting of their ID, when given the chance. In the second timeslot, the leader node assigns a priority to the node who’s message it received in the previous timeslot.

Theorem 10. The Alternative Broadcast Algorithm ensures that all nodes in an SINR network broadcast their message and acquire a unique priority in $[0, n-1]$ with probability greater than $1 - 3/n$ in $O(n \log n + \log^2 n + \log N)$ rounds.

Proof. Either of our leader election algorithms elect a leader with probability greater than $1 - 1/n$ in $O(\log N)$ rounds. Theorem 3 of [10] shows that their broadcast algorithm guarantees that after $O(n \log n + \log^2 n)$ rounds all nodes have successfully performed a broadcast with probability greater than $1 - 1/n$. The nodes pick unique IDs with probability greater than $1 - 1/N^4 > 1 - 1/n^4$.

The leader node listens to the network and assigns a priority to every node that successfully performs a broadcast. The algorithm won’t be successful if either the leader election algorithm doesn’t elect a leader, [10]’s broadcast fails or the nodes do not pick unique IDs. Using a union bound, we upper bound the probability that the algorithm will fail to $1/n^4 + 1/n + 1/n < 3/n$, for large enough $n$. Thus, the algorithm succeeds with probability greater than $1 - 3/n$ in $O(n \log n + \log^2 n + \log N)$ rounds. ◀

6 Determining the Location of $n$ Nodes

All nodes have been identified and have priorities, which allows us to proceed and $(1 + \epsilon)$-approximate their location. Below, we provide a high-level description of the locating algorithm:

First, we give an algorithm for approximating the distance between two nodes by determining the smallest possible transmission power required for communication between them given no interference from other nodes. Then, we pick three base nodes. The base nodes calculate the distances between them and define a 2-dimensional coordinate system. Finally, all nodes, sequentially, determine their distance from the base nodes and position themselves onto the coordinate system.
For clarification, $\epsilon$ denotes the error in our approximation of the coordinates of node $v$, $\epsilon_d$ denotes the error in our approximation of the distance between nodes $v$ and $u$, and $\epsilon_p$ denotes the error in our approximation of the minimum power needed for communication between nodes $v$ and $u$.

### 6.1 Approximating the Distance between two Nodes

Suppose $u$ and $v$ wish to approximate the distance between them, i.e. compute a value $d'(u,v) = (1 + \epsilon_d)d(u,v)$, for some $\epsilon_d$. They do so by identifying the lowest power $P_l$, such that a message from $u$ can still be successfully received by $v$. Then, they use $d'(u,v) = \alpha \sum P_l N_c \beta$ to approximate the distance between them, which comes from straightforward manipulation of the SINR Equation.

#### 6.1.1 Determining if Transmission Power is Sufficient for Communication

Given an arbitrary power $P_e$, a pair of nodes can determine whether or not $P_e$ is sufficient for a message transmitted from node $u$ to be received by node $v$ with a procedure presented in Algorithm 4. We assume that no other nodes are communicating in the network.

**Algorithm 4** Determine if $P_e$ is large enough for message reception between nodes $u$, $v$

1. In the first round: $u$ broadcasts "$P_e$" with transmission power $P_e$
2. In the second round:
3. if $v$ received a message in the first round then $v$ broadcasts "$P_e$" with power $P_e$
4. if $u$ receives a message in the second round then Return True end if
5. end if
6. Return False // $P_e$ is not large enough

#### 6.1.2 Approximating the Minimal Required Transmission Power

Now, that the nodes can determine whether or not a transmission power $P_e$ is large enough for message reception, they need to approximate the lowest power $P_l$, with which a message can still be received. Since the possible powers $P_e$ are real numbers, we aim to $(1 + \epsilon_P)$-approximate $P_l$, for some $\epsilon_P$. The nodes first find a 2-approximation for $P_l$ by doing a binary search on $i$ for possible transmission powers $2^i$, and then improve this approximation to a $(1 + \epsilon_P)$-approximation, using binary search on the possible transmission powers.

From the definition of a single-hop network, we know that any two pairs of nodes can communicate with transmission power $P$. The nodes shall try to guess $P_l$’s value to the nearest factor of 2 in time $O(\log \log P)$, in the interval $[1, P]$. They do so by using an exponential variant of binary search. We present this procedure in Algorithm 5 below.

The nodes $(1 + \epsilon_P)$-approximate $P_l$ using a binary search in the interval $[P_e/2, 2P_e]$, where $P_e$ is a 2-approximation for $P_l$, until the size of the interval they are searching in is smaller that $\epsilon_P$, as shown below in Algorithm 6.

**Theorem 11.** Algorithm 6 returns a $(1 + \epsilon_P)$-Approximation for $P_l$ in $O(\log \log P + \log(1/\epsilon_P))$ rounds w.h.p.

**Proof.** First we prove that Algorithm 5 returns a 2-approximation of $P_l$, the minimum transmission power needed for communication between nodes $u$ and $v$ w.h.p. in $O(\log \log P)$ rounds.
Algorithm 5 2-approximation of $P_l$

1: Set $i = 0$, $L = 1$ and $U = \log(P)$.
2: while $L \neq U + 1$ do // In every round
3: Run Procedure 'TransmissionPowerCheck' for $P_c = 2^{[(U+L)/2]}$
4: if 'TransmissionPowerCheck' returned True then $U = [(U + L)/2]$
5: else $L = [(U + L)/2]$
6: end if
7: end while
8: Run Procedure 'TransmissionPowerCheck' for $P_c = 2^L$
9: if 'TransmissionPowerCheck' returned True then Return L
10: else Return U 
11: end if

Algorithm 6 $(1 + \epsilon_P)$-Approximation for $P_l$ for some $u, v \in V$

1: $P_c = \text{Result of running Algorithm 5, Set } L = P_c/2 \text{ and } U = 2P_c$
2: while $U - L \geq \epsilon_P$ do // In every round
3: Run Procedure 'TransmissionPowerCheck' for $P_c = [(U + L)/2]$
4: if 'TransmissionPowerCheck' returned True then $U = \lceil \frac{L+L}{2} \rceil$
5: else $L = \lceil \frac{U+L}{2} \rceil$
6: end if
7: end while
8: Return $P_c = \lceil \frac{U+L}{2} \rceil$

Algorithm 5 searches for a 2-approximation for $P_l$, given upper bound $P$ and lower bound $P_{lo} = 1$. $P_{lo} = 1$ is a lower bound as in order for a message to be received, the equation $\text{SINR}(u,v,0) = \frac{P_{c}}{N_c (d(u,v))^\beta} \geq \beta$ must be satisfied, where $d(u,v), N_c, \beta > 1$. The nodes search for a 2-approximation of $P_l$ in the interval $\{1, P\}$. They do so by performing a binary search on $P_c = 2^i$ for $i \in [0, \log P]$. Thus, the procedure identifies the closest power of 2 to $P_c$, returning a 2-approximation, as $P_c \in [2^k, 2^{k+1}]$ for some integer $k$. This procedure terminates when the binary search terminates, which takes $O(\log i)$ or $O(\log \log P)$.

From Algorithm 5 we know that $P_l$ is in the interval $[P_c/2, P_c]$. Thus, by running a binary search on $[P_c/2, 2P_c]$ until the size of the interval we are searching in is smaller that $\epsilon_P$, we can $(1 + \epsilon_P)$-approximate $P_l$. This binary search requires $O(\log(1/\epsilon_P))$ rounds. Thus, we conclude that Algorithm 6 returns a $(1 + \epsilon_P)$-Approximation for $P_l$ in $O(\log \log P + \log(1/\epsilon_P))$.

6.2 Approximating distance $d(u,v)$

Now, we have all the tools to $(1 + \epsilon_d)$-approximate $d(u,v)$. Using Algorithm 6 nodes $u$, $v$ determine $P_l'$, an $(1 + \epsilon_d)$-approximation for $P_l$, where $\epsilon_P = \alpha \epsilon_d$. Then, they use $d'(u,v) = \sqrt{\frac{P_{c}}{N_c \epsilon_d}}$ to find $d'(u,v) = (1 + \epsilon_d) d(u,v)$, and broadcast the result.

Algorithm 7 $(1 + \epsilon_d)$-Approximating $d(u,v)$ for some $u, v \in V$

1: Run Algorithm 6 to $(1 + \epsilon_d)$-approximate $P_l'$
2: Use $d'(u,v) = \sqrt{\frac{P_{c}}{N_c \epsilon_d}}$ to calculate $d'(u,v)$ and broadcast $d'(u,v)$. 

CVIT 2016
Theorem 12. Algorithm 7 \((1 + \epsilon_d)\)-approximates \(d(u, v)\) for a pair of nodes \((u, v)\) in \(O(\log \log P) + O(\log \frac{1}{\epsilon_d \alpha}) + O(1)\) rounds.

**Proof.** A pair of nodes \((1 + \epsilon_d)\)-approximate \(d(u, v)\) using \(d'(u, v) = \sqrt{\frac{P}{N_{\beta}}}\). Thus, \(d(u, v) \approx (1 + \epsilon_d)\sqrt{\frac{P_{ul}}{N_{\beta}}} = \sqrt{(1 + \epsilon_d)^2 P_{ul}}\). In order to get an \((1 + \epsilon_d)\)-approximation for \(d(u, v)\), the binary search requires an interval of size \(\epsilon P = (1 + \epsilon_d)^2 - 1 \approx 1 + \epsilon_d \alpha - 1 \approx \epsilon_d \alpha\). Algorithm 6 requires \(O(\log \log P + \log(1/\epsilon P))\) rounds. Calculating \(d'(u, v)\) and broadcasting the result takes 1 round. Thus, in total the algorithm takes \(O(\log \log P + \log \frac{1}{\epsilon_d \alpha} + 1)\) rounds.

### 6.3 \((1 + \epsilon)\)-Approximation of the Location of All Nodes

This section details our algorithm for \((1 + \epsilon)\)-approximating the location of all nodes in a network. Here, we define \(\epsilon = 4\epsilon_d R^2 + \epsilon_d + \sqrt{\epsilon_d/2R}\).

We assume a 2-dimensional metric space, and exploit the spatial restraints to achieve a fast run time. A point in a 2-dimensional metric space can be fully defined given its distance to three non-collinear points. For this algorithm we pick three base nodes, and then calculate the distances between them. The three base nodes create a 2-dimensional coordinate system and then every other node approximates its distance from the base nodes and places itself on the coordinate system. Below, we present the full algorithm followed by the pseudocode.

The first base node \(u\) is the node with the highest *priority*. Node \(u\) determines its distance from all nodes \(v \in V - \{u\}\). The node \(v\) that is the furthest away from \(u\) is picked as the second base node. Then, node \(v\) calculates its distance from all nodes \(v \in V - \{u, v\}\). Finally, nodes \(u\) and \(v\) determine which of the remaining nodes is furthest from the line that goes through \(u\) and \(v\). As shown in Figure 2, the nodes need to solve \((d(u, w)^{\prime})^2 = y^2 + (x + d(u, v)^{\prime})^2\) and \((d(v, w)^{\prime})^2 = y^2 + x^2\) for \(y\) to determine the third base node \(p\).

The three nodes then create a 2-dimensional coordinate system. Node \(u\) picks coordinates \((0, 0)\). Then node \(v\) picks coordinates \((0, d(u, v)^{\prime})\). Node \(w\) solves the following system of distance equations \(d(u, w)^{\prime} = \sqrt{(0 - x_w)^2 + (0 - y_w)^2}\) and \(d(v, w)^{\prime} = \sqrt{(0 - x_w)^2 + (d(u, v)^{\prime} - y_w)^2}\) to determine its coordinates \((x_w, y_w)\). Node \(w\) has two choices for its coordinates, and it randomly picks one. This choice determines the orientation of the \(y\) axis in our coordinate system.

After all the base nodes have broadcasted their coordinates, the remaining nodes in descending priority order, approximate their distances from the base nodes and place themselves in the 2-dimensional coordinate system using the distance equations to determine their two coordinates. Each node \(z\) has three distance equations, and two unknowns to
determine (its coordinates). Thus, by solving the system of equations, \( z \) can determine its coordinates and broadcast them, so that all other nodes know its location. The pseudocode is presented below in Algorithm 8.

**Algorithm 8** (1 + \( \epsilon \))-Approximating all nodes’ relative location

1. Run the Broadcast Algorithm or the Alternative Broadcast Algorithm
2. Let \( u \) be the node with the highest priority.
3. for \( z \in V - \{u\} \) do
   4. Nodes \( z \), and \( u \) run Algorithm 7
4. end for
6. Let node \( v \) be the node furthest from \( u \).
7. for \( z \in V - \{u,v\} \) do
   8. Nodes \( z \), and \( v \) run Algorithm 7
9. end for
10. Let node \( w \) be the node furthest from the line that passes from node \( u \) and \( v \).
11. Nodes \( u,v \) and \( w \) approximate and broadcast their coordinates
12. for \( z \in V - \{u,v,w\} \) and \( z' \in \{u,v,w\} \) do
   13. Run Algorithm 7 for \((z,z')\), node \( z \) approximates and broadcasts its coordinates
14. end for

**Theorem 13.** Algorithm 8 (1 + \( \epsilon \))—approximates the relative location of all nodes, such that \( \sqrt{(x - x')^2 - (y - y')^2} \leq \epsilon \), where \((x,y)\) are the real coordinates, and \((x',y')\) are our approximate coordinates.

**Proof.** Let’s first determine the accuracy with which we determine the location of the three base nodes, and then calculate the error in our approximation of the location of any node. As a reminder, \( \epsilon = 4\epsilon_dR^2 + \epsilon_d + \sqrt{\epsilon_d}/2R \).

The first base node \( u \) assumes coordinates \((0,0)\). The second base node \( v \) takes coordinates \((0,d'(u,v))\). We know that \( d(u,v) \leq d'(u,v) \leq (1 + \epsilon_d)d(u,v) \). Thus, \( v \)'s coordinates can be shifted by at most \( \epsilon_d \) from its real location. The third base node \( w \) approximates its coordinates based on its distance from nodes \( u \), and \( v \).

Node \( w \) has a \((1 + \epsilon_d)\)-approximation of its distance from base node \( u \). Thus, \( w \) knows that its position is in the annulus of two circles both centered around \( u \), one with radius \( d(u,w), \) which is the actual distance between nodes \( u \) and \( w \), and the other one with radius \( (1 + \epsilon_d) \cdot d(u,w) \), as seen in Figure 3. Note that the annulus would be much thinner, but for clarity we drew it thicker. Let’s look at the second base node \( v \). Again, node \( w \) will be located in the annulus of two circles both centered around \( v \), one with radius \( d(v,w) \), and the other one with radius \( d(v,w)' = (1 + \epsilon_d) \cdot d(v,w) \)

Now, the circle centered around node \( u \) with radius \( d(u,w) \), and the circle centered around node \( v \) with radius \( d(v,w) \) can intersect in (i) one point, or (ii) two points. They cannot intersect in more points, as the centers of the circles have to be distance 1 apart.

i. Suppose the two circles intersect at two points. The two annuli’s intersection area maximizes when both annuli have the maximum possible width, which occurs when the inner circle of the annulus has radius \( R \). Additionally, we show that the intersection areas maximize when the centers of the annuli are as close to each other as possible. Under our assumptions in Section 2, the minimum distance between two nodes is 1. In Figure 4, we present the worst case. The intersection area has hight \( h \), and width \( L \). First, let’s calculate \( h \). Height \( h \) is the distance between the peak of an isosceles triangle with two sides of length \( R \), and base
Figure 4 Node $w$ is located in the intersection of the red annulus with the blue annulus. The intersection area has height $h$, and width $L$. The purple circles, centered around each node with radius $1/2$, denote the space around each node where there can’t be any other purple circle centered at a node. (Not to scale)

Figure 5 Node $w$ is located somewhere in the pink shaded area. To maximize the potential of $w$’s annulus to include both possible areas for $z$ in its annulus, we place $w$ as far from $z$ as possible, and as close as possible to the line going through $u$ and $z$. (Not to scale)

$b_x$, and the peak of an isosceles triangle with two sides of length $R’ = (1 + \epsilon_d)R$, and base $b_x$. Thus, $h = \sqrt{R'^2 - b_x/4} - \sqrt{R^2 - b_x/4} < \sqrt{2\epsilon_d}R$. Now, let’s calculate $L$. Observe that $L$ is the base of an isosceles trapezoid, whose sides have length $R$, and the diagonals have length $(1 + \epsilon_d)R$. By Ptolemy’s theorem, $((1 + \epsilon_d)R)^2 = L \cdot b_x + R^2$. Thus, $L \approx \frac{2\epsilon_d R^2}{b_x}$. Here, we also observe that indeed the distance between the two base nodes is inversely proportional to $L$, and maximizes for $b_x = 1$, at $L = 2\epsilon_d R^2$. Note that this is worst than the worse case scenario, as $u$ is the furthest node from $v$. We conclude that each of the intersection areas are smaller than a circle centered at $w$ with radius $\epsilon_d R^2$. Note, that node $w$ has a choice of two points to choose from, and it randomly picks one of them.

ii. Suppose the two circles intersect at one point. Then, there is only one intersection area. Using similar geometric arguments, we determine that the area of the intersection of the annuli is smaller than a circle centered at $w$ with radius $2\epsilon_d R^2$. The details of this computation can be found in the Appendix.

We conclude that we can approximate the location of $w$, such that our approximation is at most $2\epsilon_d R^2$ away from $w$. Now, let’s see what happens when a non-base node $z$ approximates its location given its distance from the base nodes. Let’s first assume that we know the exact locations of the base nodes. Given a similar argument to the above, $z$ can determine its coordinates such that its approximation is at most $2\epsilon_d R^2$ away from $z$ or its symmetrical point on the line connecting nodes $u$ and $v$. We can use the third base node $w$ to better approximate the location of $z$.

According to our restraints the third base node $w$ has to be in the pink shaded area in Figure 5. We consider the worst case scenario, when the intersection areas are as close to each other as possible. We make sure that the intersection areas are as close as possible, by setting $b = 2\epsilon$. If $b$ is any smaller than that, the two intersection areas are close enough to each other, that they fall under our approximation error, and we do not need to distinguish between them. There are two ways in which node $w$’s annulus can intersect with both intersection areas. (i) It might intersect once with each of the two intersections areas, or (ii) it might be so large, such that it encompasses both intersection areas.

i. Suppose that node $w$’s annulus intersects once with each of the two intersections of
the two annuli. We maximize $w$’s annulus by setting $d(w, z) = R$. We also place $w$ as close to the line that goes through $u$ and $v$ as possible to decrease its distance to both intersection areas. The point closest to $w$ from the symmetric intersection area is the symmetric point of $z$ across the line connecting $u$ and $v$. We get a contradiction, proving that $R'' > R(1 + \epsilon)$, where $R''$ is the distance from $w$ to $z$’s symmetric point, as shown in Figure 5. Here $R''^2 = R^2 + h^2$. We want to show that $R'' > R(1 + \epsilon_d)$, which is true when $h > \sqrt{2\epsilon_d}R$. We have $h = 2(4\epsilon_d R^2 + \epsilon_d + \sqrt{\epsilon_d/2}R) > \sqrt{2\epsilon_d}R$.

ii. Using similar geometric arguments we show that $w$’s annulus is not too large. A detailed computation can be found in the Appendix.

We conclude that given the exact coordinates of the base nodes, we can calculate the coordinates of all other nodes with an error smaller than $2\epsilon_d R^2 + \sqrt{\epsilon_d/2}R$. Adding the error from calculating the coordinates of the base nodes, we conclude that we can approximate the location of any node $z$, such that our approximation is at most $4\epsilon_d R^2 + \epsilon_d + \sqrt{\epsilon_d/2}R$ away from $w$. That is, if $(x, y)$ denote the real location of $z$, and $(x', y')$ denote our approximation, we guarantee that $\sqrt{(x - x')^2 + (y - y')^2} \leq 4\epsilon_d R^2 + \epsilon_d + \sqrt{\epsilon_d/2}R \leq \epsilon$. ▶

Theorem 14. Algorithm 8 $(1 + \epsilon)$-approximates the location of all nodes in $O(n(\log \log P + \log \frac{R}{\epsilon \alpha}))$ rounds in addition to the rounds required by the broadcast algorithm with probability greater than $1 - 3/n$.

The proof follows from Theorems 8, 9, 10, and 13.

7 Conclusions & Future Work

We have shown that it is possible to locate the nodes in a network under the SINR model, using the model’s limited capabilities, and provided communication complexity bounds. However, there are still many open problems in this area.

Our algorithm assumes perfect knowledge of the SINR constants. One could examine how an error in those would affect the location approximation, or if perhaps the problem can be solved without assuming knowledge of these constants. Our algorithm is sequential in nature, it would be interesting to examine if a more parallel algorithm could achieve similar results with a faster runtime.

Acknowledgements. We would like to thank Hsin-Hao Su, Nancy Lynch and Magnús M. Halldórsson for the helpful comments and discussions.

References

Locating Nodes In An SINR Network


\section*{A Extended Proof}

\textbf{Theorem 15.} Algorithm 8 $\left(1 + \epsilon \right) \approx$ approximates the relative location of all nodes, such that $\sqrt{(x - x')^2 - (y - y')^2} \leq \epsilon$, where $(x, y)$ are the real coordinates, and $(x', y')$ are the approximate coordinates.

\textbf{Proof.} Let’s first determine the accuracy with which we determine the location of the three base nodes, and then calculate the error in our approximation of the location of any node. As a reminder, $\epsilon = 4 \epsilon_d R^2 + \epsilon_d + \sqrt{\epsilon_d / 2R}$.

The first base node $u$ assumes coordinates $(0,0)$. There is no error here. The second base node $v$ takes coordinates $(0, d'(u, v))$. We know that $d(u, v) \leq d'(u, v) \leq (1 + \epsilon_d) d(u, v)$. Thus, $v$’s coordinates might be shifted by at most $\epsilon_d$ from its real location. The third base node $w$ approximates its coordinates based on its distance from nodes $u$ and $v$. Let’s see how good this approximation is.

Node $w$ has a $(1 + \epsilon_d)$-approximation of its distance from base node $u$. Thus, $w$ knows that its position is in the annulus of two circles both centered around $u$, one with radius $d(u, w)$, which is the actual distance between nodes $u$ and $w$, and the other one with radius $(1 + \epsilon_d) \cdot d(u, w)$, as seen in Figure 6. Note the the annulus would be much thinner, but for clarity we drew it thicker.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure6}
\caption{Node $w$ is located in the red annulus centered around node $u$. (Not to scale)}
\end{figure}

Let’s look at the second base node $v$. Again, node $w$ will be located in the annulus of two circles both centered around $v$, one with radius $d(v, w)$, and the other one with radius $d(v, w)' = (1 + \epsilon_d) \cdot d(v, w)$. Here, we have two cases: the circle centered around node $u$ with radius $d(u, w)$, and the circle centered around node $v$ with radius $d(v, w)$ can intersect in (i) one point, or (ii) two points.

(i) Suppose the two circles intersect at two points. The two annuli’s intersection area maximizes when both annuli have the maximum possible width, which occurs when the inner circle of the annuli has radius $R$. Additionally, we show that the intersection areas maximize when the centers of the annuli are as close to each other as possible. Under our assumptions in Section 2, the minimum distance between two nodes is 1. In Figure 7, we present the worst case.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure7}
\caption{Node $w$ is located in the intersection of the red annulus with the blue annulus. The intersection area has height $h$, and width $L$. The purple circles, centered around each node with radius $1/2$, denote the space around each node where there can’t be any other purple circle centered at a node. (Not to scale)}
\end{figure}

The intersection area has height $h$, and width $L$. First, let’s calculate $h$. Height $h$ is the distance between the peak of an isosceles triangle with two sides of length $R$, and base $b_x$, and the peak of an isosceles triangle with two sides of length $R'$, where $R' = (1 + \epsilon_d) R$, and base $b_x$. Thus, $h = \sqrt{R'^2 - b_x^2/4} - \sqrt{R^2 - b_x^2/4} < \sqrt{2\epsilon_d R}$. \hfill CVIT 2016
Now, let’s calculate $L$. Observe that $L$ is the base of an isosceles trapezoid, whose sides have length $R$, and the diagonals have length $(1 + \epsilon_d)R$. By Ptolemy’s theorem, $((1 + \epsilon_d)R)^2 = L \cdot b_x + R^2$. Thus, $L \approx \frac{2\epsilon_d R^2}{b_x}$. Here, we also observe that indeed the distance between the two base nodes is inversely proportional to $L$, and maximizes for $b_x = 1$, at $L = 2\epsilon_d R^2$. Note that this is worst than the worse case scenario, as $u$ is the furthest node from $v$.

We conclude that each of the intersection areas are smaller than a circle centered at the intersection of the inner circles of the annuli, of radius $\epsilon_d R^2$. Note that there are two possibilities for the location of $w$. In this case $w$ randomly picks one of the choices.

(ii) Suppose the two circles intersect at one point. We observe two cases, (a) $d(u, v) \geq \max(d(u, z), d(v, z))$, (b) $d(u, v) < \max(d(u, z), d(v, z))$.

(a) Here we consider the case when $d(u, v) \geq \max(d(u, z), d(v, z))$. The intersection of the two annuli here maximizes when the inner circles of the annuli have radius $R$, as this is when each annuli maximizes its width, as depicted in Figure 8.

Node $w$ is located in the intersection of the red annulus with the blue annulus. Here, $R' = (1 + \epsilon_d)R$. We calculate $h$, the height of the area of intersection, $h = \sqrt{R'^2 - R^2}$, and thus $h = \sqrt{((1 + \epsilon_d)R)^2 - R^2}$, which leads to $h \approx \sqrt{2\epsilon_d R}$. Node $z$ is in one of the two intersections of the two annuli has height $\leq 2\epsilon_d R$, and width $\epsilon_d R$. We conclude that each of the intersection areas is smaller than a circle centered at the intersection of the inner circles of the annuli, of radius $2\epsilon_d R$.

(b) Here we consider the case when $d(u, v) < \max(d(u, z), d(v, z))$, as depicted in Figure 9. The intersection of the annuli again maximizes when the annuli are as big as possible. Additionally, the closer the centers of the annuli, the larger their width, and thus the largest their intersection area will be. Again, we consider a worse than the worst case where $u$ and $v$ are 1 away from each other, and thus the larger annulus has inner radius $R$, and the smaller one, has inner radius $(R - 1)$. Let’s calculate how large the intersection area is. It’s width is at most $\epsilon_d(R - 1)$. The length can be calculated using the two isosceles triangles. We have equations $h^2 + (L/2)^2 = (1 + \epsilon_d)^2(R - 1)^2$, and $(h + 1)^2 + (L/2)^2 = R^2$. Solving for $L$, we get that $(L/2)^2 = \epsilon_d^2 R^4 + 4\epsilon_d^2 R^2 - 2\epsilon_d R^2$, and thus $L < 4\epsilon_d R^2$. Thus, the intersection area is smaller than a circle centered at the intersection of the inner circles of the annuli, of radius $2\epsilon_d R^2$.

We conclude that we can approximate the location of $w$, such that our approximation is
at most $2\epsilon_d R^2$ away from $w$.

Now, let’s see what happens when a non-base node $z$ approximates its location given its distance from the base nodes. Let’s first assume that we know the exact locations of the base nodes. Given a similar argument to the above, $z$ can determine its coordinates such that its approximation is at most $2\epsilon_d R^2$ away from $z$ or its symmetrical point on the line connecting nodes $u$ and $v$. We can use the third base node $w$ to better approximate the location of $z$.

According to our restraints the third base node $w$ has to be in the pink shaded area in Figure 5. We consider the worst case scenario, when the intersection areas are as close to each other as possible. We make sure that the intersection areas are as close as possible, by setting $b = 2\epsilon$. If $b$ is any smaller than that, the two intersection areas are close enough to each other, that they fall under our approximation error, and we do not need to distinguish between them.

There are two ways in which node $w$’s annulus can intersect with both intersection areas. (i) It might intersect once with each of the two intersections areas, or (ii) it might be so large, such that it encompasses both intersection areas.

i. Now, suppose that node $w$’s annulus intersects once with each of the two intersections of the two annuli. We maximize $w$’s annulus by setting $d(w, z) = R$. We also place $w$ as close to the line that goes through $u$ and $v$ as possible to decrease its distance to both intersection areas. The point closest to $w$ from the symmetric intersection area is the symmetric point of $z$ across the line connecting $u$ and $v$. We get a contradiction, proving that $R'' > R(1 + \epsilon)$, where $R''$ is the distance from $w$ to $z$’s symmetric point, as shown in Figure 5. Here $R'' = R^2 + h^2$. We want to show that $R'' > R(1 + \epsilon_d)$, which is true when $h > \sqrt{2\epsilon_d} R$. We have $h = 2(4\epsilon_d R^2 + \epsilon_d + \sqrt{\epsilon_d^2/2R}) > \sqrt{2\epsilon_d} R$.

ii. We make sure that $w$’s annulus is not too large, when we choose the second base node. For all nodes $w \in V$, $d(u, v) \geq d(u, w)$. To maximize the width of $w$’s annulus, we set $d(w, z) = R$. Let’s calculate the minimum width $l$ that $w$’s annulus has to be to encompass both intersection areas as shown in Figure 5. In this case, $w$’s annulus needs to be wide enough that it encompasses $v$’s annulus from the first intersection area to the next. First, we solve for the height $h$ of the isosceles triangle from node $u$ to node $z$ and its symmetrical point. We get that $h^2 = 1 - (\epsilon)^2$, and $l = 1 - \sqrt{1 - \epsilon^2}$. Thus, $l = \frac{\epsilon^2}{\sqrt{1 + \sqrt{1 - \epsilon^2}}} \approx \epsilon^2/2$. We conclude that $l > \epsilon^2/2 > \epsilon_d R$, as $\epsilon = 4\epsilon_d R^2 + \epsilon_d + \sqrt{\epsilon_d^2/2R}$.

We conclude that given the exact coordinates of the base nodes, we can calculate the coordinates of all other nodes with an error smaller than $2\epsilon_d R^2 + \sqrt{\epsilon_d^2/2R}$. Adding the error from calculating the coordinates of the base nodes, we conclude that we can approximate the location of any node $z$, such that our approximation is at most $4\epsilon_d R^2 + \epsilon_d + \sqrt{\epsilon_d^2/2R}$ away from $w$. That is, if $(x, y)$ denote the real location of $z$, and $(x', y')$ denote our approximation, we guarantee that $\sqrt{(x - x')^2 + (y - y')^2} \leq 4\epsilon_d R^2 + \epsilon_d + \sqrt{\epsilon_d^2/2R} \leq \epsilon$.\vspace{0.5cm}
Node $w$ is located somewhere in the pink shaded area. To maximize the potential of $w$’s annulus to include both possible areas for $z$ in its annulus, we place $w$ as far from $z$ as possible, and as close as possible to the line going through $u$ and $z$. 