Distributed Proximity Maintenance in Ad Hoc Mobile Networks

Jie Gao* Leonidas J. Guibas† An Nguyen†

April 14, 2005

Abstract

We present an efficient distributed data structure, called the D-SPANNER, for maintaining proximity information among communicating mobile nodes. The D-SPANNER is a kinetic sparse graph spanner on the nodes that allows each node to quickly determine which other nodes are within a given distance of itself, to estimate an approximate nearest neighbor, and to perform a variety of other proximity related tasks. A lightweight and fully distributed implementation is possible, in that maintenance of the proximity information only requires each node to exchange a modest number of messages with a small number of mostly neighboring nodes. The structure is based on distance information between communicating nodes that can be derived using ranging or localization methods and requires no additional shared infrastructure other than an underlying communication network. Its modest requirements make it scalable to networks with large numbers of nodes.

1 Introduction

Collaborating intelligent mobile node scenarios appear in a wide variety of applications. Consider aircraft flying in formation: each plane must be aware of the locations and motions of its neighbors in order to properly plan its trajectory and avoid collisions. If some aircraft are fuel tankers, each plane may need to determine the nearest tanker when its fuel is low. Or take the case of a search team in a rescue operation where collaboration among team members is essential to guarantee coordinated search and exhaustive coverage of the rescue area. Current research provides many other similar multi-node collaboration examples, from the deployment of sensors in a complex environment by a set of mobile robots [15] to the intelligent monitoring of forests by suspended mobile sensors [16]. In all these scenarios a loosely structured collaborative group of nodes must communicate in order to engage in joint spatial reasoning, towards a global objective. The spatial reasoning required almost always involves proximity information — knowing which pairs of nodes are near each other. Proximity information plays a crucial role in these scenarios because nearby nodes can interact, collaborate, and influence each other in ways that far away nodes cannot.

Motivated by such examples, we consider in this paper the task of maintaining proximity information among mobile nodes in an ad hoc mobile communication network. We provide a data structure that allows each node to quickly determine which other nodes are within a given distance of itself, to estimate an approximate nearest neighbor, and to perform a variety of other proximity related tasks. As a matter of fact, proximity is important not only for the tasks the network has to perform, such as those illustrated above, but for building the network infrastructure itself. Mobile nodes typically use wireless transmitters whose range is limited. Proximity information can be essential for topology maintenance, as well as for the formation of node clusters and other hierarchical structures that may aid in the operation of the network. For example, it is sometimes desirable to perform network deformation so as to achieve better topology with lower delay [6].

There has not been much work in the ad hoc networking community on maintaining proximity information. A closely related problem is to keep track of the 1-hop neighbors, i.e. the nodes within communication range, of each node. This is a fundamental issue for many routing protocols and the overall organization of the network. But even this

*Center for the Mathematics of Information, California Institute of Technology, Pasadena, CA 91125. Email: jgao@ist.caltech.edu. Work was done when the author was with computer science department, Stanford University.

†Department of Computer Science, Stanford University, Stanford, CA 94305. E-mail: guibas, anguyen@graphics.stanford.edu.

simple problem is not easy. If nodes know their own positions and those of their neighbors, then a node can be alerted by its neighbor if that neighbor is going to move out of the communication range. But knowing when new nodes come within the communication range is more challenging. A commonly used protocol for topology discovery is for all the nodes to send out "hello" beacons periodically. The nodes who receive the beacons respond and thus neighbors are discovered. However, a critical issue in this method is how to choose the rate at which "hello" messages are issued. A high beacon rate relative to the node motion will result in unnecessary communication costs, as the same topology will be rediscovered many times. A low rate, on the other hand, may miss important topology changes that are critical for the connectivity of the network. Unfortunately, as in physical simulations, the maximum speed of any node usually gates this rate for the entire system. Recently Amir *et al.* proposed a protocol for maintaining the proximity between 'buddies' in an ad hoc mobile network so that a node will be alerted if its buddies enter a range with radius R [2]. The basic idea is that each pair of buddies maintains a strip of width R around their bisector. The bisector is updated according to the new node locations when one of the nodes enters the strip. Although the nodes move continuously, the bisector is only updated at discrete times. This scheme, however, does not scale well, when the number of buddies is large — as would be the case when we care about potential proximities among all pairs of nodes.

The general problem of maintaining proximity information among moving objects has been a topic of study in various domains, from robot dynamics and motion planning to physical simulations across a range of scales from the molecular to the astrophysical. It would take us too far afield to summarize these background developments in other fields on distance computations, collision detection, etc. One relatively recent development in proximity algorithms that provides the basic conceptual setup for the current work is the framework of *kinetic data structures* (or KDSs for short) [4, 13]. The central idea in kinetic data structures is that although objects are moving continuously, an underlying combinatorial structure supporting the specific attribute(s) of interest changes only at discrete times. These critical events can be detected by maintaining a cached set of assertions about the state of the system, the so-called *certificates* of the KDS, and exploiting knowledge or predictions about the node motions. A certificate failure invokes the KDS repair mechanism that reestablishes the desired combinatorial structure and incrementally updates the certificate set. Many KDSs related to proximity have appeared in recent years [9, 5, 11, 12, 3, 1].

All current KDS implementations are centralized, requiring a shared or global event queue where events are detected and processed. In our setting, a centralized event queue would require that location/trajectory updates from all the nodes be sent to a central location, leading to unacceptably high communication costs. These centralized approaches are thus not directly applicable to ad hoc mobile networks, where a distributed implementation is always desirable and often necessary for the additional reasons of fault tolerance and load balancing. We set as our goal to develop a distributed protocol for proximity maintenance whose computation and communication costs scale well with the size of the network and whose proximity query processing is output-sensitive, in that querying neighboring areas is cheaper than querying far away regions.

A first contribution of this paper is to extend traditional centralized kinetic data structures to the domain of ad hoc mobile networks. We introduce the notion of *distributed kinetic data structures* (dKDSs for short) that demand no shared infrastructure other than a communication network among the mobile nodes and are therefore ideally suited to ad hoc mobile network settings. In a dKDS each node holds a small number of the centralized KDS certificates that are relevant to its portion of the global state. The KDS is maintained globally by exchanging messages among the nodes and updating the local certificate sets. As with any KDS implementation, an important issue to address is the interface between the KDS and the node motions — this directly determines the complexity of predicting or detecting KDS certificate failures. We describe two possible interfaces between motion information and the kinetic data structure: the *shared flight-plan* and the *distance threshold* motion models, each with its own advantages.

Our second contribution is the development of a lightweight and efficient distributed kinetic data structure for proximity maintenance among mobile nodes. The structure is based on our earlier work [10] on proximity maintenance in a centralized setting. In [10] we introduced the DEFSPANNER data structure that forms a sparse *graph spanner* [8] of the complete graph on the nodes, when edges are weighted via the Euclidean distance. A spanner has the property that for every pair of nodes there is a path in the spanner whose total length is a $(1+\epsilon)$ approximation of the Euclidean distance between the nodes. Thus the spanner implicitly and compactly encodes all distance information. Here ϵ is a parameter that trades-off the approximation quality against the spanner size (number of edges needed). The D-SPANNER structure introduced in this paper, a dKDS for proximity maintenance among mobile nodes, represents a major reworking of our DEFSPANNER structure so as to make it decentralized. This is a highly non-trivial task, as

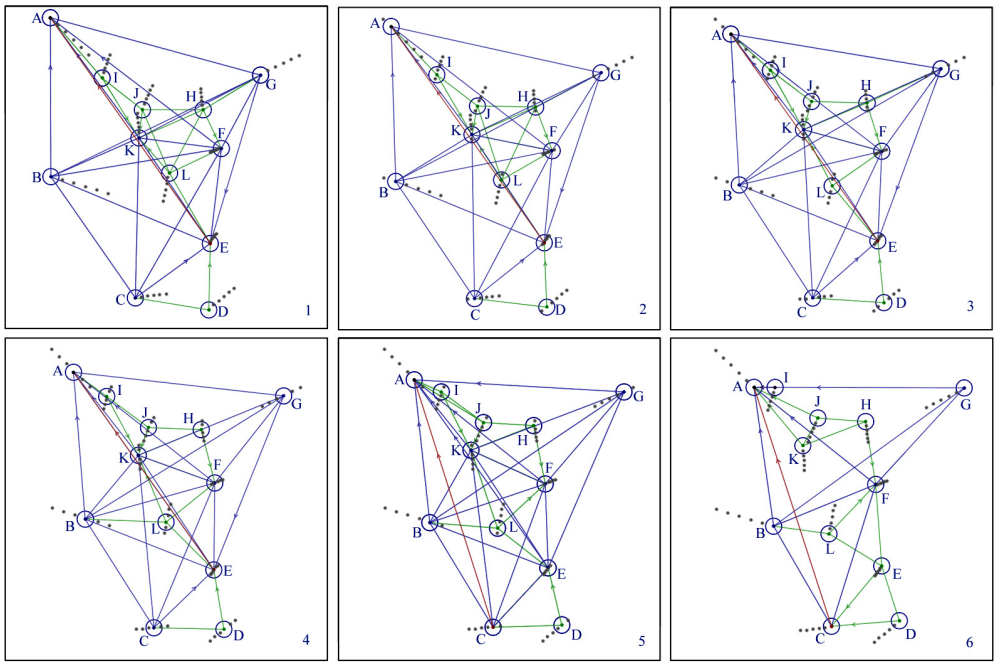

Figure 1. A sequence of D-SPANNERs. We show the trajectories of the nodes by a sequence of gray dots. Each figure shows the D-SPANNER at the current time frame. The D-SPANNER changes relatively slowly even though the nodes move significantly.

classical KDSs gain much of their efficiency by assuming that failed certificates are processed in exact chronological order, one at a time. Instead, in the distributed setting, certificates may fail asynchronously at different nodes and the corresponding repair processes will run in parallel and must be coordinated. To our knowledge, this is the first distributed implementation of any KDS, attesting to the difficulty of the task.

2 D-SPANNER: a distributed kinetic spanner

The D-SPANNER data structure is defined by a sparse graph on the mobile nodes. In Figure 1, we show a number of frames of an animation of the spanner on a small number of moving nodes. These frames show 12 moving nodes and the D-SPANNER data structure we maintain. The number of edges varies from 20 to 31. Note that although the nodes move significantly, the spanner graph, as a combinatorial structure, changes relatively slowly. The relatively slow rate of change for the spanner is due to the fact that it is a *highly non-canonical* structure: for given positions of the nodes, many roughly equally good spanner subgraphs exist. This ensures the *stability* of the spanner: as the nodes move, there is likely to be a spanner for the new configuration that is combinatorially close to the spanner we had for the old positions. Thus the frequency of kinetic updates is small and so is the maintenance overhead the structure imposes.

Once we have a spanner data structure, we can efficiently answer many types of proximity queries. Effectively, the spanner replaces an expensive continuous search over the 2-D plane with a lightweight combinatorial search over the spanner subgraph. For instance, the spanner can be used to detect and avoid collisions between moving vehicles, especially unmanned vehicles, a task that is very difficult for any kind of on-demand discovery scheme. The spanner can do this, because only pairs of nodes with a spanner edges between them can possibly collide [10]. The continuous maintenance of the D-SPANNER guarantees that every possible potential collision is captured and predicted. As another example, a node u can locate all nodes within a distance d of itself by just initiating a restricted broadcast along the spanner edges that stops when the total distance traversed is $(1+\epsilon)d$. The set of nodes thus discovered is guaranteed to include all nodes within distance d of u; a further filtering step can remove all false positives, of which there will

typically be few. The set of nodes within distance d can also be maintained through time so that u gets alerted only when nodes join the set or drop off from the set. Additionally, the D-SPANNER keeps track of a $(1+\varepsilon)$-nearest neighbor for *every* node in the network at *any* time. The neighbor of p in the spanner with shortest edge length is guaranteed to be within distance $(1+\varepsilon)$ the distance between p and q^*, the true nearest neighbor of p. As shown in [10], the D-SPANNER gives us a nice hierarchical structure over the nodes across all scales at any time, since it implicitly defines approximate k-centers of the nodes for any given k. The D-SPANNER also gives a well-separated pair decomposition [7], which provides an N-body type position-sensitive data aggregation scheme over node pairs. For further discussion of all these properties of the spanner, see [10].

In this paper we focus on the maintenance of the D-SPANNER in a distributed environment and analyze its maintenance and query costs. The spanner is overlayed on a communication network among the nodes. We assume the existence of a routing protocol that enables efficient communication between a pair of nodes, so that the communication cost is roughly proportional to the distance between the communicating nodes. The spanner is stored distributedly, having each node keep only its incident edges in the spanner, and is maintained by relevant pairs of nodes exchanging update packets. A query of the graph structure is performed by communicating with other mobile nodes following the edges of the D-SPANNER structure. We show that the D-SPANNER has the following attractive features:

- The D-SPANNER can be efficiently maintained in a distributed fashion under both the *shared flight-plan* and the *distance threshold* models (these will be formally defined below).
- The total communication cost for communicating the flight plans or location information for the initial setup of the D-SPANNER is almost linear in the weight of the minimum spanning tree of the network.
- Even though fixing a failed certificate may involve $O(\log n)$ nodes, the D-SPANNER can be repaired in such a way that at any moment each node updates the D-SPANNER locally using at most $O(1)$ computation and communication steps. Furthermore, multiple certificate failures can be fixed concurrently, without interference.
- The spanner structure is a hierarchy that scales well with the network size and the geometry of the nodes. Due to the hierarchical structure of the spanner, far away node pairs are updated less frequently than close-by pairs. Under reasonable motion assumptions, the communication cost related to the D-SPANNER maintenance incurred by a node p is $O(\log n)$ for each unit distance that p moves.

These properties show that the spanner is a lightweight data structure that gracefully scales to large network sizes. We validate these theoretical results with simulations of D-SPANNERs on two data sets, one with generated vehicle motions and one with real airline flight data. Our simulations show that, for both artificial and real-world data, the D-SPANNER can be maintained efficiently so that, at any time step, only a tiny part of the spanner needs to be updated. We observe that in practice, for reasonable data sets, the spanner has a much smaller spanning ratio compared to the theoretical worst-case bound discussed below. We also present some trade-offs in maintaining the D-SPANNER on different data sets.

The paper is organized as follows. In section 3, we formally define the distributed kinetic structure framework and the associated motion models. The distributed maintenance of the D-SPANNER under the dKDS framework is discussed in section 4. Section 5 covers the issues of applying the D-SPANNER to ad hoc networks. We show simulation results in Section 6 and conclude the paper in Section 7.

3 Distributed kinetic data structures

In the classical KDS setting it is assumed that, in the short run, moving objects follow motions that can described by explicit *flight plans*, which are communicated to the data structure. Objects are allowed to modify their flight plans at any time, however, as long as they appropriately notify the KDS. These flight plans form the basis for predicting when the KDS certificates fail — a typical certificate is a simple algebraic inequality on the positions of a small number of the objects. These predicted failure times become events in a global event queue. Upon certificate failure, the KDS repair mechanism is invoked to remove the failed certificate and update the structure as necessary.

In the distributed setting appropriate for ad hoc mobile communication networks, we must distribute, and possibly duplicate the certificates among the mobile nodes themselves. Furthermore, we must give up the notion that we process

certificate failures in a strict chronological order, as nodes will process certificate failures independently of each other, as they detect them. Although it is not so readily apparent, the classical KDS setting depends quite heavily on the assumption that the KDS repair mechanism is invoked after exactly one certificate failure has occurred. This will no longer be true in the distributed case and thus the dKDS repair mechanism must be considerably more sophisticated. As we already remarked, globally broadcast flight plans are not appropriate in a distributed setting. While in some applications, like a dKDS for aircraft, it makes perfect sense to allow communicating aircraft to exchange flight plans, in others, as in our rescue team example, only something much weaker can be assumed. In the following we propose two motion models that are appropriate for ad hoc mobile networks. They are *shared flight plan model* and *distance threshold model*.

The shared flight plan model: In this motion model we assume that nodes have flight plans, but these are known only to the nodes themselves — unless they are explicitly communicated to others. A node must incur a communication cost to transmit its flight plan to another node. Furthermore, a node receiving a flight plan will assume that it is valid until the plan's owner communicates that the plan has changed. Thus each node has the responsibility of informing all other nodes who hold its flight plan of any changes to it.

The distance threshold model: In this weaker model we only assume that a node knows its own position. Its prediction for future motion is either not available or too inaccurate to be useful. A node u may communicate its current position to node v, as needed by the dKDS. Associated with the (u, v) communication is a distance threshold $\delta(u, v)$; node u undertakes to inform node v if it ever moves more than $\delta(u, v)$ from the position previously communicated. Note that δ is a function of the pair of nodes — different v's may need updates about the changing position of u at different rates.

3.1 Distributed kinetic data structure (dKDS) costs

The evaluation and analysis of a dKDS is also somewhat different from the evaluation of a traditional KDS. A KDS has four desired performance properties: efficiency, responsiveness, locality and compactness [4]. In an evaluation of the properties of a KDS, we usually assume that the nodes follow pseudo-algebraic motions[1]. Efficiency captures how many events a KDS processes, as compared to the number of changes in the attribute of interest. The responsiveness of a KDS measures the worst-case amount of time needed to update its certificate cache after an event happens. Locality measures the maximum number of certificates in which one object ever appears. Finally, compactness measures the total number of certificates ever present in the certificate cache. Low values on these measures are still desirable properties for a dKDS. In the distributed setting, however, we have to include communication costs. First, we want to bound the total communication cost for exchanging flight plans or position information. We compare this with the cost of the optimal 1-median of the communication network[2], which is a lower bound on the communication cost of sending all the flight plans to a central server, assuming that no aggregation is done. The second difference is that, when a certificate kept at node u fails, some certificates held at other nodes of the structure may need to be updated. The cost of communication to the nodes keeping these certificates must be taken into account in the processing cost of the event. Finally, locality is an especially important property for a dKDS, since it determines how evenly one can distribute the set of certificates among the mobile nodes. If a node u is involved in certificates with too many other nodes, not only is u heavily loaded by holding many certificates, but also the update cost for u's flight plan changes is high, since u's new flight plan has to be delivered to many other nodes through costly communication.

There are also practical issues in maintaining a dKDS, because of the involvement of an underlying communication network. Ideally we put the dKDS on top of a TCP-like network layer, so that communication between nodes can be assumed reliable, without packet loss. Otherwise when reliable communication is not available packet delay and loss must be considered in the process of the repair of the structure. We also assume that the communication cost is proportional to the Euclidean distance between the two communicating nodes — this is a reasonable assumption in dense networks. Finally, given that the ad hoc network environment is inherently parallel, special care is needed to make sure that KDS updates can handle race conditions and avoid deadlocks. Later we use the D-SPANNER as an example to show how these issues are handled.

[1] A motion is called algebraic with degree s if each coordinate of the motion is an algebraic function of degree s or less. For a definition of pseudo-algebraic, see [4].

[2] A 1-median of a graph $G = (V, E)$ is defined as $\min_{v \in V} \sum_{u \in V} \tau(u, v)$, where $\tau(u, v)$ is the shortest path length of u, v.

4 The D-SPANNER and proximity maintenance

For an ad hoc mobile network, a spanner is a sparse graph on the nodes that approximates the all pairs of distances. Specifically, a $(1+\varepsilon)$-spanner is a graph on the nodes such that the shortest path distance between p and q in the spanner is at most $(1+\varepsilon)$ times the Euclidean distance of p and q — $(1+\epsilon)$ is called the stretch factor. In [10] we proposed a $(1+\varepsilon)$-spanner, called DEFSPANNER that can be maintained under motion. In this paper, we show that the DEFSPANNER can also be maintained in a distributed fashion, where the nodes have no information about the global state and obtain information only through communication steps whose cost must be taken into account. This distributed version is denoted as a D-SPANNER. We emphasize that while there are many maintainable proximity structures under the KDS setting, D-SPANNER is the first kinetic structure of any kind that can be maintained distributedly. We start by reviewing the centralized DEFSPANNER in [10].

4.1 D-SPANNER

The definition of DEFSPANNER [10] requires the notion of discrete centers. A set of *discrete centers* with radius r for a point set S is defined as a maximal subset $S' \subseteq S$ such that any two centers are of a distance at least r away, and such that the balls with radius r centered at the discrete centers cover all the points of S. Notice that the set of discrete centers needs not be unique.

The DEFSPANNER G on S is constructed as follows. Given a set S of points in the Euclidean space $\mathbb{R}^d$, we construct a hierarchy of discrete centers so that S_0 is the original point set S and S_i is a set of discrete centers of S_{i-1} with radius 2^i, for $i > 0$. Intuitively, the hierarchical discrete centers are samplings of the point set at exponentially different spatial scales. Then we add edges to the graph G between all pairs of points in S_i whose distance is no more than $c \cdot 2^i$, where $c = 4 + 16/\varepsilon$. These edges connect each center to other centers in the same level whose distance is comparable to the radius at that level. Since the set of discrete centers is not unique, the DEFSPANNER is non-canonical. In fact, as we mentioned, this is the main reason why DEFSPANNER admits an efficient maintenance scheme.

The *aspect ratio* of S, denoted by α, is defined by the ratio of the maximum pairwise distance and minimum pairwise distance. In this paper we focus on point sets with aspect ratio bounded by a polynomial of n, the number of nodes. The nodes with bounded aspect ratio is a natural assumption on an ad hoc mobile network. The nodes are physical objects so they usually have a minimum separation. The maximum separation is bounded due to the connectivity requirement of the network. Thus $\log \alpha = O(\log n)$. When convenient, we assume that the closest pair of points has distance 1, so the furthest pair of S has distance α.

We use the following notations throughout the paper. Since a point p may appear in many levels in the hierarchy, when the implied level is not clear, we use $p^{(i)}$ to denote the point p as a node in level S_i. A center $q^{(i)}$ is said to *cover* a node $p^{(i-1)}$ if $|pq| \leq 2^i$. A node $p^{(i-1)}$ may be covered by many centers in S_i. We denote $P(p)$ one of those centers and call it the *parent* of $p^{(i-1)}$. The choice of $P(p)$ is arbitrary but fixed. We also call p a *child* of $P(p)$. A node p is called a nephew of a node q if $P(p)$ and q are neighbors. Two nodes p and q are *cousins* if $P(p)$ and $P(q)$ are neighbors. For a point p in level S_i, we recursively define $P^{j-i}(p)$ as the ancestor in level S_j of $p^{(i)}$ by $P^0(p) = p^{(i)}, P^{j-i}(p) = P(P^{j-i-1}(p))$. We note that if p is in level i, $p^{(j)} = P(p^{(j-1)})$ for each $j \leq i$, i.e. p is the parent of itself in all levels below i. For notational simplicity, we consider p a neighbor of itself in all levels in which it participates.

In [10], we show that a DEFSPANNER (and of course now its distributed version, the D-SPANNER), has a spanning ratio of $1+\varepsilon$ and a total of $O(n)$ edges. A detailed list of results from [10] is shown below.

Theorem 4.1 (in [10]). *For a* DEFSPANNER *on a set of points S with aspect ratio α, the following hold.*

1. *If $q^{(i)}$ is a child of $p^{(i+1)}$, then $q^{(i)}$ and $p^{(i)}$ are neighbors, i.e. there is an edge from each point q to its parent.*
2. *If p and q are neighbors, then the parents $P(p)$ and $P(q)$ are neighbors.*
3. *For any point $p \in S_0$, its ancestor $P^i(p) \in S_i$ is of a distance at most 2^{i+1} away from p.*
4. *The hierarchy has at most $\lceil \log_2 \alpha \rceil$ levels.*

5. *Each node p has $O(1)$ neighbors in any given level, and thus it has at most $O(\log_2 \alpha)$ neighbors totally.*

6. *G is a $(1+\varepsilon)$-spanner when $c = 4 + 16/\varepsilon$.*

When the node move around, the discrete center hierarchy and the set of edges in DEFSPANNER may change. Nodes may lose neighbors or gain neighbors in the spanner. Just as in our earlier discussion of maintenance of the 1-neighbors of a node, discovering the loss of a neighbor is easy, while detecting new neighbors is hard. What makes the spanner work is the fact that property (2) above implies that before two nodes can become neighbors at a given level, their parents must already be neighbors at the next level up. Thus the search for new neighbor pairs can be confined to cousin pairs only.

The D-SPANNER has the same structure as the DEFSPANNER, though the internal constants involved in the structure are larger. The key difference between the D-SPANNER and the DEFSPANNER is in the way the structure is stored and maintained. In the centralized case, the spanner is fully repaired after every event, corresponding to a single certificate failure. In the distributed setting, D-SPANNER is computed and stored distributedly. In particular, each node only stores its own presence in the discrete centers hierarchy and its edges on each level. Certificates are handled locally. When a certificate fails, we communicate with other relevant nodes to have the D-SPANNER repaired. Since the network is inherently parallel and communication takes finite time, other certificates may fail concurrently and multiple repair processes may be active in parallel. We introduce the notion of a relaxed D-SPANNER to enable multiple certificates failures to be handled simultaneously in the network.

4.2 Relaxed D-SPANNER

To make the maintenance manageable, we make use of a variant, the *relaxed* D-SPANNER. The intuition is that whenever a certificate in a D-SPANNER fails it may take $O(\log n)$ communications for the D-SPANNER to be repaired, and up to $O(\log n)$ new edges may be established. Doing all that at once is not possible in a distributed setting. Instead, the repair is done in stages; between stages we relax our constraints on the spanner through the concept of a relaxed D-SPANNER. Relaxed D-SPANNERs make it possible to deal with multiple certificate failures by encoding the simultaneous failures with *relaxed parents*, the removal of which can be done in parallel.

Fix a constant $\gamma > 2$; we call a node $q^{(i)}$ a *γ-relaxed parent*, or simply a *relaxed parent*, of $p^{(i-1)}$ if $|pq| \leq \gamma \cdot 2^i$. A *relaxed* D-SPANNER is similar to a regular D-SPANNER except that if a node $p^{(i)}$ is not covered by any node in S_{i+1}, we do not require p itself to be in S_{i+1} but only require p to have a relaxed parent in S_{i+1}.

From (3) in Lemma 4.1, it is easy to see that for any node p, $|pP^i(p)| \leq 2^{i+1} < \gamma \cdot 2^i$, and thus we can intuitively think of the i-th level ancestors of a node p as its potential relaxed-parent, if p is in level S_{i-1}. When all nodes in a relaxed D-SPANNER have parents, the relaxed D-SPANNER is a D-SPANNER. Analogous to (8) in Theorem 4.1, we can prove that a relaxed D-SPANNER is by itself a $(1+\varepsilon)$-spanner when $c = \gamma \cdot (4 + 16/\varepsilon)$.

Theorem 4.2. *A relaxed* D-SPANNER *is a $(1+\varepsilon)$-spanner when $c = \gamma \cdot (4 + 16/\varepsilon)$.*

When nodes move and some certificates fail, we first make the D-SPANNER into a relaxed D-SPANNER which will be repaired after certain communications made to other nodes. The notion of relaxed parents guarantees that the structure is not far away from a real D-SPANNER so that local communications can fix it up, as the following lemma suggests.

Lemma 4.3. *Let q be a relaxed parent of p in S_i, and $c > 4 + \gamma$.*

1. *If r is a parent of p, then q and r are neighbors in S_{i+1}.*

2. *If p is inserted to S_{i+1}, then the set of neighbors of $p^{(i+1)}$ is a subset of the cousins of q.*

Proof: If r is a parent of p, then $|rq| \leq |rp| + |pq| \leq 2^{i+1} + \gamma \cdot 2^{i+1} < c \cdot 2^{i+1}$. If p is in S_{i+1} and t is a neighbor of $p^{(i+1)}$, $|P(q)P(t)| \leq |P(q)q| + |qp| + |pt| + |tP(t)| \leq 2^{i+2} + \gamma \cdot 2^{i+1} + c \cdot 2^{i+1} + 2^{i+2} < c \cdot 2^{i+2}$. ☑

In the above description we always double the radius when we construct the discrete centers hierarchy. In fact, we chose a factor of 2 just for the simplicity of explanation. We could have chosen any factor $\beta > 1$ and construct a hierarchy of discrete centers such that in level i, the nodes are more than β^i apart, and the edges connect nodes that are closer than $c \cdot \beta^i$. We would then call a node q in S_{i+1} a relaxed parent of a node p in S_i if $|pq| \leq \gamma \cdot \beta^i$, where $\gamma > \beta/(\beta - 1)$. We choose $c > (2\beta + \gamma)/(\beta - 1) > \beta(2\beta - 1)/(\beta - 1)^2$ so that D-SPANNER can be maintained. In the simulations later, we actually use β's different from 2.

4.3 The D-SPANNER under the shared flight-plan model

The maintenance of the D-SPANNER in the centralized KDS framework was analyzed in [10]. In the dKDS setting, the absence of a centralized event queue and the lack of strict order in certificate failure processing invalidates much of the approach.

The basic idea for maintaining a D-SPANNER in a dKDS framework is as follows. Two nodes always communicate their flight plans if they are involved in a certificate. When the points move, the D-SPANNER is updated through a series of *relaxed* D-SPANNERs. When a certificate in a D-SPANNER fails, one or more relaxed parents may appear in the D-SPANNER, making it a relaxed D-SPANNER. Nodes in a relaxed D-SPANNER communicates with each other to restore the structure to a D-SPANNER. During the restoration, other certificates may fail. It is guaranteed, however, that we will eventually get a D-SPANNER after a period of no additional certificate failures, provided that the update time is small enough with respect to the velocity of the nodes and that $c > 4 + \gamma$.

We show that it only requires a constant number of communications to repair the structure locally when a certificate fails or to transform one relaxed D-SPANNER to another. We note that in a KDS setting c must be larger than 4 in order for the D-SPANNER to be maintainable. In the dKDS setting, we require that $c > 4 + \gamma > 6$. It worths pointing out that under the assumption that $c > 4 + \gamma$, a generic relaxed D-SPANNER cannot be maintained in either the KDS framework or the dKDS framework. We use a very specific series of relaxed D-SPANNERs during the maintenance of the D-SPANNER in the dKDS framework to make things work.

The certificates in a dKDS are similar to the certificates in KDS. They are all distance certificates, asserting that the distances between given pairs of nodes are above or below a certain threshold. A certificate about the distance between two nodes u and v is stored in both the event queues of u and v. Since u and v have the flight plans of each other, both of them can evaluate the first time when the certificate fails. Specifically, a node p as a point in S_i maintains four kinds of certificates:

1. *Parent certificate*: asserts that p is covered by its parent $P(p)$ if p has one, i.e., $|pP(p)| \leq 2^{i+1}$;
2. *Short edge certificates*: assert that $|pq| \geq 2^i$ for each neighbor $q \in S_i$ of p;
3. *Long edge certificates*: assert that $|pq| \leq c \cdot 2^i$ for each neighbor $q \in S_i$ of p;
4. *Potential edge certificates*: assert that $|pq| > c \cdot 2^i$ for each non-neighbor cousin q of p.[3]

When a certificate fails, each node involved in the certificate updates its event queue and performs the updates to the D-SPANNER. The updates for the certificates are as follows:

1. *Parent events*: When a parent certificate fails, we make the former parent a relaxed parent.
2. *Short edge events*: When $|pq| < 2^i$, the node with the lower maximum level, say p, removes itself from S_i. The children of p in S_{i-1} now have q as a relaxed parent, and $p^{(i-1)}$ has q as its parent.
3. *Long edge events*: When $|pq| > c \cdot 2^i$, the edge is simply dropped. The long edge certificate on pq is deleted from the certificate lists of both p and q. Accordingly some potential edge certificates between the cousins of p, q are dropped also.

[3]Note that by (2) in Theorem 4.1, a pair of nodes cannot be neighbors before they first become cousins. Thus the potential edge certificates capture all possible edges that may appear in the near future.

4. *Potential edge events*: When a potential edge fails, a new edge is simply added. p and q communicate with each other about their children. A long edge certificate on pq and potential edge certificates between the cousins of p, q will be added to the certificate lists of both p and q.

For the first two types of certificate failure, we introduce more nodes with relaxed parents who can find their true parents by the following procedure. For a node $p^{(i)}$ with a relaxed parent $q^{(i+1)}$, we find a parent for $p^{(i)}$ using Lemma 4.3, namely, looking among all neighbors of $q^{(i+1)}$ for one that covers $p^{(i)}$. If such a node is found, $p^{(i)}$ has a parent. If not, p is at least 2^{i+1} away from every node in S_i. Thus p must be promoted to level S_{i+1}. There are two possible cases. In the case when $q^{(i+1)}$ has a parent r, it is easy to see that $p^{(i+1)}$ has r as a relaxed parent, and by Lemma 4.3, the neighbors of $p^{(i+1)}$ can be found among the cousins of $q^{(i+1)}$. In the case when q only has a relaxed parent, we have to wait until $q^{(i+1)}$ has a parent before p can be promoted to level S_{i+1}. Note that this is the only time when a node has to wait for another node. When a node has to wait, it always waits for a node in one level higher. This monotonic order guarantees that deadlock cannot occur. We note that the number of communications to search for a parent or to promote a node up one level is $O(1)$. In the worst case, a parent certificate failure may move a node all the way up the hierarchy. In the absence of certificate failures that generate nodes with relaxed parents, all the nodes with relaxed parents will eventually have parents, and thus we eventually obtain a D-SPANNER.

We currently do not handle the case when a relaxed parent of a node moves too far from the node. We could avoid dealing with this situation by noting that when a node $p^{(i)}$ first has a relaxed parent, the distance from it to the relaxed parent is always less than $2^{i+2} < \gamma \cdot 2^{i+1}$. If p and its relaxed parent move slowly enough or γ is large enough, p finds a true parent before its relaxed parent moves far away from it. If p loses its relaxed parent before it can find a parent, we can treat p as a newly inserted node and use the D-SPANNER dynamic update as explained in section 5.

4.4 The D-SPANNER under the distance threshold model

The distance threshold motion model allows even less information, when compared with the shared flight plans motion model — a node only knows its current location. For each level S_i, we take a distance threshold $d_i = \mu \cdot 2^i$, $\mu = (c - 4 - \gamma)/8$, such that for each certificate in S_i involving a pair of nodes (u, v), we let u and v to inform each other their locations whenever any of them moves a distance of d_i from the last exchanged location.

Under this model, each node u predicts a certificate failure based on its current location and the last communicated location of its partner v, denoted by v_0. Node v may have moved since the last time it posted its location. However $|vv_0| \leq d_i$. Since u and v do not have the most updated locations of each other, they may not agree on when a certificate on u, v should fail. For example, in u's view of the world, a long edge certificate on u, v fails, i.e., $|uv_0| \geq c \cdot 2^i$. Then u informs v of appropriate updates and v changes its spanner edges. Although according to v it is possible that the long edge certificate has not failed yet, we can still guarantee that $|uv| \geq (c - \mu) \cdot 2^i$. In summary, under the distance threshold model, we maintain a μ-approximation to an exact D-SPANNER: If a node $p \in S_i$ covers a node q, $|pq| \leq (1 + \mu) \cdot 2^i$; Two nodes $p, q \in S_i$ have $|pq| \geq (1 - \mu) \cdot 2^i$; Two nodes $p, q \in S_i$ with an edge have $|pq| \leq (c+\mu) \cdot 2^i$; Two nodes $p, q \in S_i$ without an edge have $|pq| \geq (c-\mu) \cdot 2^i$. Notice that when two nodes $p, q \in S_i$ have a distance $(c - \mu) \cdot 2^i \leq |pq| \leq (c + \mu) \cdot 2^i$, p, q may or may not have an edge.

In the global view, we maintain a μ-approximate D-SPANNER through a series of γ-relaxed D-SPANNERs, with $\gamma \geq 2(1 + \mu)$. A μ-approximate D-SPANNER is a $(1 + \varepsilon)$ spanner for appropriately chosen parameters c, μ. The updates on certificates failures are the same as in the shared flight plan model. The following lemma is a more general version of Lemma 4.3, which shows that a node with a relaxed parent can either find a parent or promote itself to one level higher under the distance threshold model.

Lemma 4.4. *Let $c > 4 + \gamma$, and q be a relaxed parent of p_1 in S_i at time 1. Assume that all points involved move less than $d_i = \mu \cdot 2^i$, $\mu = (c - 4 - \gamma)/8$, between time 1 and time 2. At time 2,*

1. *If r is a parent of p, then q and r are neighbors in S_{i+1}.*
2. *If p is inserted to S_{i+1}, then the neighbors of $p^{(i+1)}$ are among the cousins of q.*

Proof: To simplify the notation, we use suffix to denote time. If r_2 is a parent of p_2, then $|r_2q_2| \leq |r_2p_2| + |p_2p_1| + |p_1q_1| + |q_1q_2| \leq (1 + \mu)2^{i+1} + d_i + \gamma \cdot 2^{i+1} + d_i < (c - \mu) \cdot 2^{i+1}$. If p_2 is in S_{i+1} and t_2 is a neighbor of $p_2^{(i+1)}$,

$|P(q_2)P(t_2)| \leq |P(q_2)q_2| + |q_2q_1| + |q_1p_1| + |p_1p_2| + |p_2t_2| + |t_2P(t_2)| \leq (1+\mu)\cdot 2^{i+2} + d_i + \gamma \cdot 2^{i+1} + d_i + (c+\mu)\cdot 2^{i+1} + (1+\mu)\cdot 2^{i+2} = (c-\mu)\cdot 2^{i+2}$. ☑

Since we have less information in the distance threshold model, communications may be required for internal verification of the D-SPANNER, without resulting in any combinatorial changes in its structure. The cost of maintaining the D-SPANNER in the distance threshold model is the same as the cost of maintaining the D-SPANNER in the shared flight-plan model plus the cost of regular position updates. In the next section we will show a concrete bound on the maintenance cost.

4.5 Quality of distributed maintenance

In this subsection we study the memory, computation and communication cost of maintaining a D-SPANNER.

4.5.1 Memory requirement for each node

The total number of certificates at any time is always linear in the number of nodes. Each node is involved in $O(1)$ certificates for each level it participates in. Since there are at most $\log \alpha$ levels, each node only has $O(\log \alpha)$ certificates in its queue. The D-SPANNER thus scales well when the network size increases.

4.5.2 The startup communication cost for exchanging flight plans

In order for the nodes in a D-SPANNER to build their event queues, the nodes that are involved in a certificate will have to inform each other of their flight plans. Note however, that communications between two nodes are not of equal costs, with communications between far away nodes costing more than those between close-by nodes. Under our assumption that the cost of a multi-hop communication between two nodes is proportional to the Euclidean distance between them, we show that the cost of exchanging flight plans between the nodes in a D-SPANNER is low by the following theorems.

Lemma 4.5. *The total length of all the edges in each level S_i is $O(1)$ times the total length of the minimum spanning tree (MST) of the points in S_i.*

Proof: For each point p in S_i, we charge each edge incident on the node to an edge in the MST incident on p. As there are only $O(1)$ edges incident on p, each edge in the MST is charged at most $O(1)$ times. Note also that each edge in the MST has length at least 2^i and that each edge in the level has length at most $c \cdot 2^i$, the claim follows. ☑

As a direct result of Lemma 4.5, we have

Theorem 4.6. *The total length of the edges in a D-SPANNER is at most $O(\log \alpha)$ times the total length of the minimum spanning tree (MST) of the underlying points.*

Proof: The result follows from Lemma 4.5 and from the fact that the total length of minimum Steiner tree and MST are at most a factor of $O(1)$ from each other [14], and that the total length of a minimum Steiner tree decreases when points are removed. ☑

Note also that the cousin pairs in S_i are of a distance at most $c' \cdot 2^i$, where $c' < 2c+4$, and thus, similar to Theorem 4.6, we have

Corollary 4.7. *The summation of the distances between all cousin pairs in a D-SPANNER is $O(\log \alpha)$ times the total length of the MST of the underlying points.*

Notice that if we use the centralized KDS, the communication cost of every node sending their flight plans (or locations) to a central server is at least the weight of the optimal 1-median of the communication network, which is at least the weight of the minimum spanning tree. On the other hand, it is not hard to construct an example[4] such

[4]Assume a list of nodes are staying on the x-axis with distance 1 between adjacent nodes. The communication network is just the chain.

that the optimal 1-median of the network has weight $\Omega(n)$ times the total length of the MST. With this in mind, Theorem 4.6 and Corollary 4.7 imply that, in both the shared flight plans and distance threshold motion model, the startup communication cost of the D-SPANNER, i.e., the cost of communicating the shared flight plans or locations between the pairs of nodes in the certificates, is $O(\log \alpha)$ times the weight of the MST. Thus by using the distributed spanner we save substantially on the communication cost.

In the distance threshold motion model we update the locations of the nodes even if no certificate failures happen. Since the distance threshold, d_i, depends exponentially on i, the level number, nodes on higher levels update their positions less frequently but their communication costs are higher. More precisely, assume that the highest level that a node p appears is i. When p moves a distance d, the total cost of communicating its locations to its neighbors is bounded by $\sum_{j=1}^{i} O(c \cdot 2^j) \cdot d/d_j = O(di) \leq O(d \log \alpha)$. Thus the amortized communication cost of location updates for each node p is bounded by $O(\log \alpha)$ per unit distance that p moves.

4.5.3 The total number of events handled in a D-SPANNER

In [10] we showed that the number of combinatorial changes of a $(1+\varepsilon)$-spanner can be $\Omega(n^2)$ and the number of certificate failures of a D-SPANNER is at most $O(n^2 \log n)$ under pseudo-algebraic motion, where each certificate changes from TRUE to FALSE at most a constant number of times. This claim is still true for the distributed environments. Thus the number of events we process in a D-SPANNER is close to the optimum.

4.5.4 The communication cost of certificate updates in a D-SPANNER

In a D-SPANNER, when a certificate fails, a node can repair the spanner locally with a constant number of communications, though the repair may introduce relaxed parents which have to be dealt with later. Each time a search for a true parent from a relaxed parent takes a constant number of communications, and a node may actually get promoted all the way up the hierarchy. In the worst case the node is promoted in $O(\log \alpha)$ levels. We first note that the cost of promoting a node up a level, introduced by resolving relaxed parents, can be amortized on the short edge events where a node is demoted down a level, since a node can not be demoted without first being promoted. Thus in the following study we neglect the cost of fixing relaxed parents and only consider the communication cost of repairing the certificates.

We bound the communication cost of certificates updates under pseudo-algebraic motions. We examine the update costs for the certificates defined on a particular pair of nodes p, q with p on level i. Nodes p, q may be involved in five different kinds of certificates, as shown in section 4.3. In particular, we characterize the certificate failures into three categories by the distance between p, q when the certificate fails: event A includes the parent event with p being the parent of q and the short edge event; Event B includes the long edge event and the potential edge event; Event C includes the parent event with q being the parent of p. Notice that when event A happens, the distance between p, q is exactly 2^i. When event B happens, the distance between p, q is exactly $c \cdot 2^i$. When event C happens, the distance between p, q is 2^{i+1}. Since p, q follow pseudo-algebraic motion, the number of times that events A (or B, C) happens is only a constant. Furthermore, it is not hard to see that between events in different categories, the distance between p, q must change by at least 2^i. Therefore, suppose the distance between p, q changes monotonically by d, the update cost for certificates failures with p, q is bounded by $O(c \cdot 2^i) \cdot d/2^i = O(d)$. Without loss of generality assume p moves faster than q, then p moves at least a distance $d/2$. Thus we can charge the update cost to p so that p is charged of communication cost $O(1)$ for each unit distance p moves. Since p has a constant number of neighbors in level i and p may appear in at most $O(\log \alpha)$ levels, we combine the costs in all levels together such that on average a node p with highest level h incurs $O(h)$ communication cost for each unit distance it moves. Therefore we have the following theorem.

Theorem 4.8. *Under pseudo-algebraic motion, the communication cost incurred by a node p related to the* D-SPANNER *maintenance is at most $O(\log \alpha)$ for each unit distance that p moves.*

5 The D-SPANNER in ad hoc mobile networks

To maintain a D-SPANNER in an ad hoc mobile network, there are networking issues to be considered. In this section we will discuss how to adapt D-SPANNER to the remaining resources of the nodes to achieve better load balancing, how to handle node insertion and deletion, how network delay and packet loss affect the correctness and efficiency of the maintenance, and how to obtain a trade-off between query cost and the maintenance cost of D-SPANNER.

5.1 Load balancing

In an ad hoc network, the wireless nodes usually have limited resources. Like any hierarchical structures, the nodes on higher levels of the D-SPANNER are more loaded than the nodes on lower levels of the hierarchy. Although the maximum number of levels of the D-SPANNER is only logarithmic in the number of nodes, it is still desirable to adapt the D-SPANNER according to the resources of the nodes. We show how to update the spanner to balance the loads on the nodes. When the nodes move, or even when the nodes remain fixed but the power is drained at different rates in the network, we could distributedly update the D-SPANNER so that the fewer resources a node has, the less it does in the D-SPANNER, due to that fact that the D-SPANNER is highly non-canonical.

Assume that we want to reduce the load on a given node p, and let S_i be the highest level in which p appears. The first thing we can do is to look at the children of p in all levels. If they have alternative parents, we let them use the alternative parents, and thus reduce the number of children of p. If p has some children that do not have any alternative parents, we promote some of those *essential* children of p and remove p from S_i. Specifically, suppose there is a node q in $S_j, j < i$, and that q is in the highest level among all children of p. We show how to remove p from all levels above S_j.

First, we virtually split p into two nodes, the first copy p_1 contains nodes p in level S_j and below. The second copy, p_2 contains node p in level S_{j+2} and above and it is marked for deletion. We remove p from level S_{j+1}, promote q to level S_{j+1} and find all the neighbors of $q^{(j+1)}$ from the children of neighbors of $P(p_2^{(j+1)})$. We set the relaxed parent of $q^{(j+1)}$ to $P(p_2^{(j+1)})$. We also set the parent of $p_1^{(j)}$ and the relaxed parent of all former children of p in S_j to $q^{(j+1)}$. Note that the cost of this local update is $O(1)$, and after the update, we have a relaxed spanner with an exception that p_2 only exists in level $j+2$ and above, and it is marked for deletion.

In subsequent steps, we maintain the relaxed spanner just as normal with some special care taken to nodes that are marked for deletion. Let u be a node that is marked for deletion, and u is in S_k or above. We remove u from S_k. For all children (or relaxed children) of u in level S_{k-1}, we first search for their true parents. If none has u as its parent (or relaxed parent), we are done. If not, we pick v among the children (or relaxed children), promote it up to S_k. The neighbors of $v^{(k)}$ can be found among the children of neighbors of $P(u^{(k)})$. We set the relaxed parent of the remaining children to $v^{(k)}$, and set the relaxed parent of $v^{(k)}$ to $P(u^{(k)})$. We note that in each step, a node that is marked for deletion is removed from one level, and the cost of each step is $O(1)$.

By the above procedure, we promoted q to level S_{j+1} and remove p from all levels above S_j. This procedure can be repeated such that p can be further removed from high levels in the hierarchy with its work load reduced.

5.2 Dynamic update

In an ad hoc network, it is advantageous to allow nodes to go to sleep and to wake up later. It is also desirable to have the flexibility of introducing new nodes into the D-SPANNER or removing depleted nodes from it. We show that we can dynamically insert and delete nodes from the D-SPANNER.

Delete a node from a D-SPANNER is easy: we simply mark the node for deletion. In the same way as shown in the previous subsection, we will eventually get rid of the node in at most $O(\log \alpha)$ steps.

Insertion of a node p into a D-SPANNER can also be done efficiently. Note that we could easily extend the insertion algorithm for static settings by effectively walking down the hierarchy from top down to insert p as in [10]. We notice, however, that in an ad hoc network, it is often easy to obtain the neighbors of a node in the D-SPANNER at the bottom level, as the neighbors should be among the nodes within a communication range of the node being inserted. Suppose by 1-hop broadcasting we know a neighbor q of p, we show that we could insert p quickly.

The insertion of a new node p is done in two steps. In the first step, we let p to find out its highest level and its parent in that level. In the second step, we insert p to all levels below that highest level. The search for the highest level in which a node p participates is done as following. Assume that we currently determine that p must be in level i or above, and that p has a neighbor $q^{(i)}$. From (2) in Theorem 4.1, $P(p^{(i)})$ would be a neighbor of $P(q^{(i)})$. Thus we can search the neighbors of $P(q)$ and if any of them cover p, we found a parent for p. If not, p must be in S_{i+1} or above, and $P(q)$ is a neighbor of p in S_{i+1}, and the search continues. Note that the number of communications needed in each level is $O(1)$. Once we find out the highest level of p, we insert p in the levels one by one from top down. In each step, p is inserted in a level, and the neighbors of p in that level can be trivially computed easily using (2) in Theorem 4.1.

5.3 Trade-offs between maintenance cost and query cost

There are tradeoffs between the cost of maintaining a D-SPANNER and the cost of performing a query. In particular, when the nodes in the network move very fast, we trade in the query cost for lower maintenance cost by only maintaining some higher levels of D-SPANNER. More precisely, we can artificially increase the unit distance between the nodes in the D-SPANNER and ignore the condition that all nodes in the bottom levels must be more than a unit distance away from each other.

Note that all the construction and maintenance still work without this condition. The only problem is that the D-SPANNER now have many more edges at the bottom level. Secondly, instead of explicitly maintaining all bottom edges, we maintained them implicitly. For each node in the bottom level, we only maintained its parent in S_1. Whenever the neighbors of a node p are needed, we compute them by searching among the nephews of its parent.

We would like to point out that when the distance threshold model is used, even when we are not interested in making the performance trade off, we are forced to choose a large unit distance in the D-SPANNER simply to satisfy Lemma 4.4.

5.4 Network delay and packet loss

Communication by a routing protocol may introduce unbounded network delay or even packet loss. Therefore the updates of certificates or motion information may arrive out of order or even get lost. However, the D-SPANNER is a rather robust structure, and such situations are not fatal. As long as the spanner is still a relaxed D-SPANNER, it will eventually get fixed by our kinetic update scheme. If by the newest location information the spanner is not consistent, for example, a node does not have a relaxed parent, then we always repair the structure by deletion and re-insertion of a node according to its most current location information.

6 Simulations

To show that the D-SPANNER is well behaved in practice, we computed and maintained the D-SPANNER in two sets of simulations. The simulations validate our theory that the D-SPANNER is a sparse structure that can be maintain efficiently under motion.

6.1 Cars in a downtown 'Manhattan-like' area

In the first simulation, we considered a set of moving cars in an 11 by 11 block region of a city downtown, see Figure 2. There were 20 one-lane road, 5 roads in each of the North, South, East, and West directions. Cars entered the region on one side and left the region on the other side. Each car moved at a random but constant speed between 0.2 and 0.3 block per second. Each car stopped if necessary to avoid collisions with other cars then moved again at its assigned speed. We allowed cars to disappear (say to stop in parking structures) within the city blocks.

The number of cars n in the downtown area was a parameter of the simulation. In our simulations, n took one of the following values: 30, 50, 100, 150, 200, and 250. We kept the number of cars in a simulation constant by introducing a new car whenever some other car left the scene.

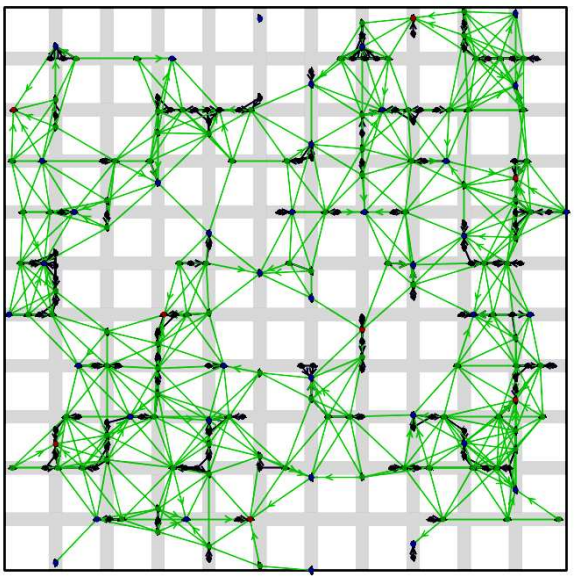

Figure 2. The bottom 2 levels of a D-SPANNER of the cars are shown. Edges with arrows indicate parent child relation in the D-SPANNER.

We constructed and maintained a D-SPANNER on the cars. We used $\beta = 3$ and $c = 4.1$, i.e. the nodes in level S_i were at least 3^i apart, and edges in level S_i were at most $c \cdot 3^i$ long.

6.1.1 Work load and memory requirement on each node

Figure 3 shows the averages over time for the maximum and the average degrees of the D-SPANNER on the cars. The degree of a node is proportional to its memory requirement and is also a measure of the work it does in maintaining the D-SPANNER.

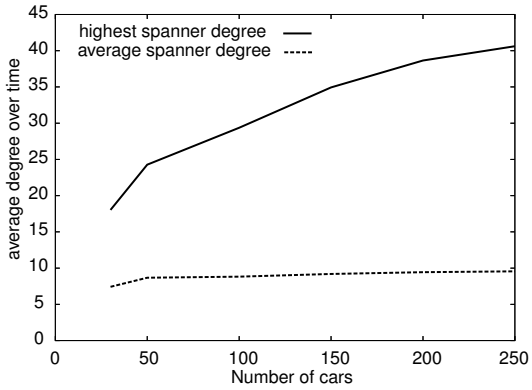

Figure 3. The average degree of a node in a D-SPANNER is approximately constant, and the maximum degree in a D-SPANNER growths slowly.

It is clear from Figure 3 that the average degree of the D-SPANNER was approximately constant and the maximum degree of a node grew slowly when the number of cars increased. The result agrees with the theory, which predicts that the size of the spanner is linear, and the maximum degree grows as $O(\log n)$.

We note that the average degree of the nodes in the D-SPANNER was around 9. The D-SPANNER was thus reasonably sparse, say comparing to planar graphs which have an average node degree of 6.

6.1.2 Spanning ratio

When $\beta = 3$ and $c = 4.1$, the theory predicts that the D-SPANNER has a spanning ratio approximately 16.36. The spanning ratio of the D-SPANNER in our experiments was much smaller. In all experiments, the spanning ratio fluctuated between 2 and 4 most of the time, and the average spanning ratio over time was about 2.7.

6.1.3 Stability

Figure 4 shows the average number of events processed each second in the simulations. Even though the cars move fast, the rate at which the events happen is low, suggesting that the D-SPANNER is stable. We note that most certificate failures are on the long edge and the potential edge certificates. These two types of failures were cheap, as we only have to remove or to add an edge to the D-SPANNER to fix them. There are much fewer failures of the other two types which gave rise to relaxed parents and the involved procedure to remove them.

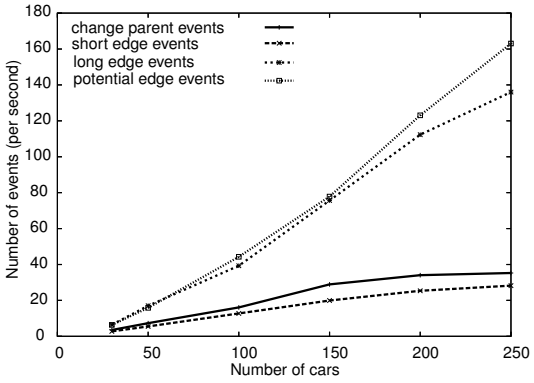

Figure 4. The number of times the D-SPANNER needs to be updated is small. The majority of events requires simple edge adding or removal.

6.2 D-SPANNERs on US flights

In the second set of experiment, we simulate a D-SPANNER on the flights in the US. The flight trajectories were from real flight data on July 27, 2002. There were from 4000 to 5000 air planes during the period of simulation. As the planes took off, moved, or landed, we dynamically inserted nodes into our D-SPANNER, kinetically updated the structure, or removed them from the structure.

In our flight data each plane obtained its location about once every 60 seconds, and the planes often moved at a speed from 300 to 500 miles per hour, a fairly high speed compared to the distance separating the air planes. Under this condition, we are forced to increase the unit distance in the D-SPANNER, see subsection 5.3. The unit distance in the modified D-SPANNER we maintained was 16 miles. We essentially fully maintained a D-SPANNER on a dynamically selected set S_1 of air planes that were at least 48 miles from each other, and for each of those air planes, maintained a collection of air planes it covered, i.e. air planes within its 48 miles radius.

As in the simulation for cars, we constructed a spanner with $\beta = 3$ and $c = 4.1$. Figure 5 shows a spanner in the simulation. We found that on average there were about 14,000 edges in the modified D-SPANNER. The spanning ratio of the D-SPANNER restricted to the nodes in level S_1 was 3.15 on average. Note that because there are many many implicit edges, if we consider the entire D-SPANNER with all those implicit edges, the spanning ratio would be much lower.

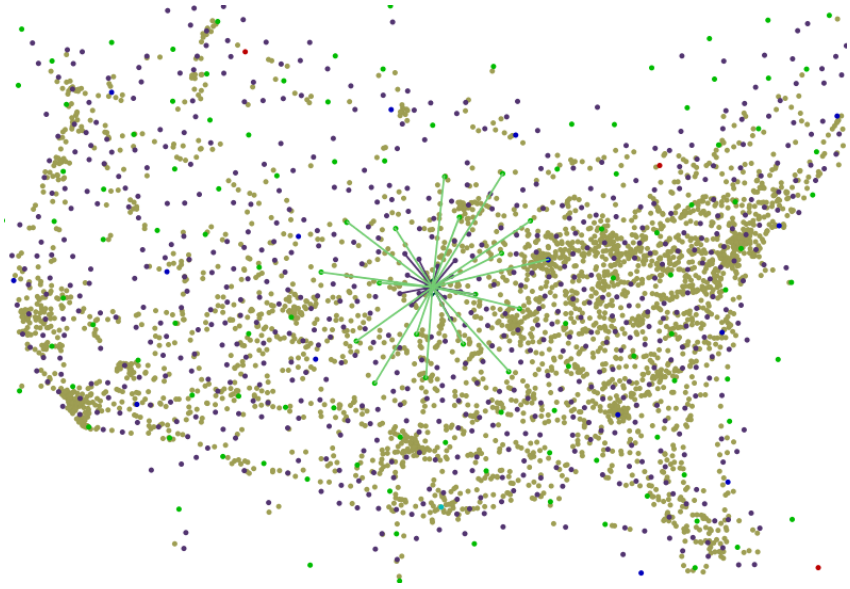

Figure 5. The flights in the US. The edges incident on a selected node are also shown.

Because of the large unit radius, the maintenance of the structure was relatively cheap. The entire structure processed 3.98, 1.63, 3.03, and 2.88 events per second for parent events, short edge events, long edge events, and potential edge events respectively.

We note that from time to time, due to missing data, some planes changed their positions too much between their consecutive position updates. When a node moves more than a certain distance threshold, we remove the node from the D-SPANNER, update its position, then insert it back to the D-SPANNER.

7 Conclusion

We have demonstrated a novel data structure for maintaining proximity information in a distributed way, among mobile nodes communicating via an ad hoc network. Our D-SPANNER structure is lightweight and scalable, allowing a variety of proximity queries over all scales while balancing the load across the network. It is a distributed structure intrinsically affixed to the mobile nodes themselves and not to some terrestrial infrastructure. Thus its performance is naturally coupled only to changes in the relative distances among the nodes and factors out any global motions of the entire system.

Acknowledgements: The authors gratefully acknowledge the support of NSF grants CCR-0204486 and CNS-0435111, as well as the DoD Multidisciplinary University Research Initiative (MURI) program administered by the Office of Naval Research under Grant N00014-00-1-0637.

References

[1] P. Agarwal, J. Basch, L. Guibas, J. Hershberger, and L. Zhang. Deformable free space tilings for kinetic collision detection. *International Journal of Robotics Research*, 21(3):179–197, 2002.

[2] A. Amir, A. Efrat, J. Myllymaki, L. Palaniappan, and K. Wampler. Buddy tracking - efficient proximity detection among mobile friends. In *Proc. of the 23rd Conference of the IEEE Communications Society (INFOCOM)*, volume 23, pages 298–309, March 2004.

[3] J. Basch, J. Erickson, L. J. Guibas, J. Hershberger, and L. Zhang. Kinetic collision detection between two simple polygons. In *Proc. of the 10th ACM-SIAM symposium on Discrete algorithms*, pages 102–111, 1999.

[4] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. *J. Alg.*, 31(1):1–28, 1999.

[5] J. Basch, L. J. Guibas, and L. Zhang. Proximity problems on moving points. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 344–351, 1997.

[6] A. Basu, B. Boshes, S. Mukherjee, and S. Ramanathan. Network deformation: traffic-aware algorithms for dynamically reducing end-to-end delay in multi-hop wireless networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 100–113. ACM Press, 2004.

[7] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42:67–90, 1995.

[8] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[9] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Discrete mobile centers. *Discrete and Computational Geometry*, 30(1):45–63, 2003.

[10] J. Gao, L. Guibas, and A. Nguyen. Deformable spanners and applications. In *Proc. ACM Symposium on Computational Geometry*, pages 190–199, June 2004.

[11] L. Guibas, J. Hershberger, S. Suri, and L. Zhang. Kinetic connectivity for unit disks. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 331–340, 2000.

[12] L. Guibas, A. Nguyen, D. Russel, and L. Zhang. Collision detection for deforming necklaces. In *Proc. 18th ACM Symposium on Computational Geometry*, pages 33–42, 2002.

[13] L. J. Guibas. Kinetic data structures — a state of the art report. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Proc. Workshop Algorithmic Found. Robot.*, pages 191–209. A. K. Peters, Wellesley, MA, 1998.

[14] D. S. Hochbaum and D. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33:533–550, 1986.

[15] A. Howard, M. J. Matarić, and G. S. Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots Special Issue on Intelligent Embedded Systems*, 13(2):113–126, 2002.

[16] W. J. Kaiser, G. J. Pottie, M. Srivastava, G. S. Sukhatme, J. Villasenor, and D. Estrin. Networked infomechanical systems (NIMS) for ambient intelligence. CENS Technical Report 31, UCLA, December 2003.