Protocol Design for Dynamic Delaunay Triangulation *

Dong-Young Lee and Simon S. Lam
Department of Computer Sciences
The University of Texas at Austin
{dylee, lam}@cs.utexas.edu

Abstract

Delaunay triangulation (DT) is a useful geometric structure for networking applications. In this paper we investigate the design of join, leave, and maintenance protocols to construct and maintain a distributed DT dynamically. We define a distributed DT and present a necessary and sufficient condition for a distributed DT to be correct. This condition is used as a guide for protocol design. We present join and leave protocols as well as correctness proofs for serial joins and leaves. In addition, to handle concurrent joins and leaves as well as node failures, we present a maintenance protocol. An accuracy metric is defined for a distributed DT. Experimental results show that our join, leave and maintenance protocols are scalable, and they achieve high accuracy for systems under churn and with node failures. We also present application protocols for greedy routing, clustering, broadcast, and multicast within a radius, and discuss and prove their correctness.

1. Introduction

With almost a hundred years of history, DT [1] and Voronoi diagram [2] have been widely used for many applications in different fields of science and engineering, including computer science. A triangulation in a 2D space means, for a given set of nodes, constructing edges between pairs of nodes such that the edges form a non-overlapping set of triangles that cover the convex hull of the nodes. DT in a 2D space is usually defined as a triangulation such that the circumcircle of each triangle does not include any node other than the vertexes of the triangle. DT can be similarly generalized for higher dimensions.

An interesting property of DT is that it connects a node to other nodes that surround the node. This property may be useful in simulation-type applications, including distributed virtual reality systems and multiplayer on-line games, since an entity in a simulation usually interacts with other entities around it. For example, a molecule interacts with other molecules around it, and a character in on-line games mostly interacts with other characters around it. Furthermore, we also design a protocol to multicast a message within a given radius from the source node, which will be useful for many simulation-type applications such as multiplayer on-line games.

Another property of DT in networking context is that greedy routing always succeeds on a DT [3]. In greedy routing, a node forwards a message to one of its neighbors that is closest to a given destination node. Note that greedy routing on an arbitrary graph is prone to the risk of being trapped at a local optimum, i.e., routing stops at a non-destination node that is closer to the destination than any of its neighbors. However, on a DT it is guaranteed that greedy routing always succeeds to find the destination node. Note that greedy routing does not always find a shortest route. However, the quality of the greedy route is often very good, since the length of an optimal route between a pair of nodes on a DT is within a constant time of the direct distance [4].

While our approach is more system-oriented compared to previous work, our protocols are also based on a rigorous theoretical foundation. In a distributed DT, each node in a system keeps a set of its neighbor nodes. We specify a distributed DT by the *neighbor sets* of all nodes. A distributed DT is correct when it is equivalent to its corresponding centralized DT. That is, a distributed DT is correct when each node has the same set of neighbors as on the corresponding centralized DT. (We will define a distributed DT and its correctness more carefully in section 2.) In section 3, we identify *a necessary and sufficient condition* to achieve correctness. We use this condition as a guide for designing join, leave, and maintenance protocols for constructing and maintaining a distributed DT. Our join and leave protocols are proved to be correct in the following sense: If a distributed DT is correct when a new node joins or an existing node leaves and there is no other concurrent join, leave or failure then, at the end of protocol execution, the resulting distributed DT is correct. Thus if a sequence of joins and leaves occur serially (i.e., one finishes before another starts), the distributed DT is correct whenever protocol execution finishes.

In practice, nodes may join and leave concurrently. Furthermore, nodes may fail at any time, immediately breaking

*Research sponsored by National Science Foundation ANI-0319168 and CNS-0434515.

correctness of the distributed DT. Our maintenance protocol has been designed to address such scenarios. We do not have a convergence proof for the maintenance protocol. However, in every one of a large number of experiments conducted to date, our maintenance protocol converged to a correct DT some time after a long period of system churn, during which nodes join and leave (also fail) concurrently and frequently.

Note that even in the case of serial joins and leaves, correctness of a distributed DT is, strictly speaking, broken as soon as a node joins or leaves, and it is recovered only at the end of protocol execution. Therefore a correct distributed DT is *impossible to achieve continually*. We observe that some applications can benefit from an incorrect distributed DT as long as it is sufficiently "accurate." Thus the accuracy of a distributed DT over a long duration of time is a more useful metric in practice than the notion of convergence to correctness. We will define an accuracy metric for a distributed DT, and show that our protocols achieve high accuracy under different scenarios of system churn.

In addition to protocols to construct and maintain a distributed DT, we present several application protocols, including greedy routing, clustering, broadcast, and multicast within a radius. As we discussed earlier, it is known that greedy routing from a node to another node on a DT always succeeds. Then we prove that greedy routing can also be used to locate an existing node that is closest to a given point (or a node that is not in the system yet). As an application of the protocol to find the closest existing node, we present a node clustering protocol. Given a set of nodes and an upper bound on the radius of a cluster, the clustering protocol partitions nodes into clusters of radii within the given upper bound. In the protocol, each cluster has a center node and the center nodes form a distributed DT. Similar approaches to clustering are found in prior work, based on a random graph of clusters [11] or a complete graph of clusters [12]. Note that greedy routing on a random graph is not guaranteed to succeed and a complete graph may result in limited scalability.

Our broadcast protocol is based on the reverse path of greedy routing, and is named GRPB (greedy reverse path broadcast). GRPB does not require any knowledge of global triangulation or per-session state. A node determines its next-hop nodes to forward a broadcast message solely using local information, namely the coordinates of its neighbor nodes and the source node.

We observe that the distance from a source node to each hop in GRPB monotonically increases, since the distance to a destination node decreases in greedy routing. Therefore our protocol to multicast within a given radius easily follows. RadGRPM (radius greedy reverse path multicast) is basically the same as GRPB, except that it additionally checks whether the next-hop nodes are within the radius from the source node. RadGRPM also keeps the advantage of GRPB that it does not require any global information or per-session state. RadGRPM is simple and is useful for simulation-type applications. For example, an explosion of a bomb in a battlefield simulation will affect entities within some range and will be observed within a longer range.

Experimental results show that our protocols are scalable, and work very well under system churn, i.e., when concurrent joins and leaves occur frequently. Even with ungraceful node failures, which inevitably result in an incorrect distributed DT, the maintenance protocol recovers correctness some time after churn and failures stop.

The organization of this paper is as follows. In section 2, we introduce concepts and definitions of distributed DT and also present application protocols. In section 3, we present a correctness condition for a distributed DT. Then we present our join, leave and maintenance protocols, discuss their correctness, and introduce an accuracy metric. In section 4, experimental results are presented. We discuss related work in section 5 and conclude in section 6.

2. Distributed Delaunay triangulation

In this section we introduce DT, Voronoi diagram and distributed DT. Consider a set of nodes. Conceptually, nodes are points in a Euclidean space. The results and protocols in this paper are for d-dimensional spaces, where $d \geq 2$. Most previous results on distributed DT in the literature are limited to 2D [5, 8, 9] and 3D [6] spaces. The only prior work on distributed dynamic DT for d-dimensional spaces is by Simon et al. [7]. See section 5 for more details.

We first define Voronoi diagram of a set of given nodes and then define DT as the dual of the Voronoi diagram. Note that there is another way of directly defining DT using circumcircles of triangles (or circum-hyperspheres of simplexes in higher dimensions), as was briefly introduced in the introduction. Since the properties of DT of interest to us come from Voronoi diagram, we believe that this approach is appropriate in our context. Lastly, we define distributed DT. In a distributed DT, each node maintains a set of its neighbor nodes. We define a distributed DT by the neighbor sets of all nodes.

In the second part of this section, applications of DT are discussed. An important and well-known property of DT is that a simple greedy routing algorithm is guaranteed to succeed on a DT, without being stuck at a local optimum [3]. We prove a similar property that greedy routing can also find the closest node to a given point. Clustering of network nodes is an example for which this property can be utilized. We also present protocols for broadcast and for multicast within a radius, and prove correctness for the protocols.

2.1. Concepts and definitions

Definition 1 *Consider a set of nodes S in a Euclidean space. The* **Voronoi diagram** *of S is a partitioning of the space into cells such that a node $u \in S$ is the closest node to all points within its Voronoi cell $VC_S(u)$.*

That is, $VC_S(u) = \{p \mid D(p,u) \leq D(p,w), \text{for any } w \in S\}$ where $D(x,y)$ denotes the distance between x and y. Note that a Voronoi cell in a d-dimensional space is a convex d-dimensional polytope enclosed by $(d-1)$-dimensional facets.

Definition 2 *Consider a set of nodes S in a Euclidean space. $VC_S(u)$ and $VC_S(v)$ are neighboring Voronoi cells, or* **neighbors** *of each other, if and only if $VC_S(u)$ and $VC_S(v)$ share a facet.*

Definition 3 *Consider a set of nodes S in a Euclidean space. The* **Delaunay triangulation** *of S is a graph on S where two nodes u and v in S have an edge between them if and only if $VC_S(u)$ and $VC_S(v)$ are neighbors of each other.*

Figure 1 shows a Voronoi diagram (dashed lines) for a set of nodes in a 2D space and a DT (solid lines) for the same set of nodes. Note that $VC_S(v)$ and $VC_S(w)$ are neighbors of $VC_S(u)$ but $VC_S(x)$ is not, since $VC_S(u)$ and $VC_S(x)$ share only a point. Similarly, in a 3-dimensional space, Voronoi cells that share only an edge or a point are not neighbors. We also say that u and v are neighbors of each other when $VC_S(u)$ and $VC_S(v)$ are neighbors of each other. Also note that facets of a Voronoi cell perpendicularly bisect edges of a DT. Therefore, a DT is the dual of a Voronoi diagram.[1] Let us denote the DT of S as $DT(S)$.

Definition 4 *A* **distributed Delaunay triangulation** *of a set of nodes S is specified by $\{< u, N_u > \mid u \in S\}$, where N_u represents the set of u's neighbor nodes, which is locally determined by u.*

Definition 5 *A distributed Delaunay triangulation of a set of nodes S is* **correct** *if and only if both of the following conditions hold for every pair of nodes $u, v \in S$: i) if there exists an edge between u and v on the global DT of S, then $v \in N_u$ and $u \in N_v$, and ii) if there does not exist an edge between u and v on the global DT of S, then $v \notin N_u$ and $u \notin N_v$.*

That is, a distributed DT is correct when for every node u, N_u is the same as the neighbors of u on $DT(S)$. Since u does not have global knowledge, it is not straightforward to achieve correctness. We will identify the condition to achieve correctness for a distributed DT in section 3.

[1] In geometry, polyhedra are associated into pairs called duals, where the vertices of one correspond to the faces of the other.

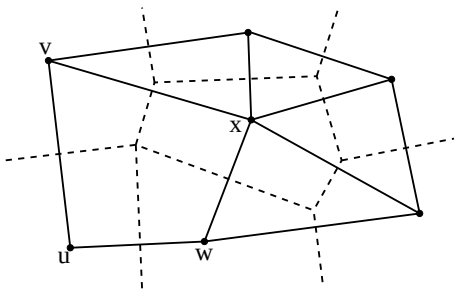

Figure 1. A Voronoi diagram (dashed lines) and the corresponding DT (solid lines) in a 2-dimensional space.

2.2. Applications of distributed Delaunay triangulation

In this section we present several protocols to illustrate the usefulness of distributed DT for networking applications. We assume for now that a set of nodes S form a distributed DT. Our protocols to construct and maintain a distributed DT are deferred to section 3. We also assume that nodes are associated with their coordinates. When a node "knows" other nodes, it also knows their coordinates. That is, a node knows its own coordinates, its neighbor's coordinates, and the coordinates of other nodes that it knows such as the destination node in routing and the source node in broadcasting. The distance between any two nodes can be calculated from their coordinates.

Greedy routing

A well-known property of DT is that greedy routing always succeeds on a DT [3]. In greedy routing, a node forwards a message to the closest node to the destination among its neighbors. As with many greedy approaches, the greedy routing algorithm is prone to risk of being stuck at a local optimum. That is, on an arbitrary graph, a non-destination node may be closer than any of its neighbors to the destination, thus stopping greedy routing at the node. However, on a DT, it is guaranteed that greedy routing succeeds to deliver a message to the destination node. Furthermore, the quality of the greedy route is often very good, since the length of an optimal route between a pair of nodes on a DT is within a constant time of the direct distance [4].

Finding the closest existing node

Similar to the previous application of greedy routing, a DT may be utilized in finding the closest existing node to a given point. (Note that the given point may not be a node in the DT.) Finding the closest existing node is a common operation in many Internet applications, including server selection, node clustering, and peer-to-peer overlay networks.[2] We have proved [14] that greedy routing always

[2] If topology-aware virtual coordinates (e.g. [17]) are used for Internet applications, application performance would be affected by accuracy of the virtual coordinates.

succeeds to find a closest existing node to a given *point* as long as it is run on a DT, using an approach similar to Bose and Morin's proof [3] that greedy routing always succeeds to arrive at a given destination *node* on a DT.

Clustering of network nodes

To illustrate an application of finding the closest existing node to a given point, we present a simple clustering protocol of network nodes. The protocol is a distributed version of a centralized clustering algorithm adopted from [10].[3] The upper bound R of the radius of a cluster is given as a parameter. In the centralized algorithm, nodes are considered sequentially in deciding whether they should join an existing cluster or create a new cluster. The first node considered creates a new cluster and becomes the center of it, since there is no existing cluster. From the second node on, the considered node is tested to see if its distance to the center of the closest existing cluster is within R or not. If so, the considered node joins the cluster; otherwise it creates its own cluster and becomes the center of it. The algorithm stops when all nodes are considered. Note that the result of clustering may be different depending on the order in which nodes are considered [10].

Our clustering protocol is a distributed version of this centralized algorithm. The main challenge in converting it into a distributed version is to find the closest existing cluster without global knowledge. We solve this problem by utilizing greedy routing on a DT. Recall that each cluster has a center node. In our protocol, existing center nodes form a distributed DT. A non-center node does not participate in the distributed DT. When a node u joins the system, it first finds the closest existing center node by using greedy routing on the distributed DT of the center nodes. Suppose that the center node s_u is found. If the distance from u to s_u is within the upper bound R, u becomes a member of the cluster centered at s_u; otherwise u creates its own cluster, becomes the center node of the new cluster, and joins the distributed DT.

Other distributed approaches to clustering are found in prior work. In [11], clusters form a random graph and a joining node may fail to find the closest existing cluster. In [12], every node maintains links to every other cluster, limiting scalability. (The scalability issue is addressed in [12] by introducing a hierarchy of clusters.) Our protocol finds the closest cluster for a joining node and is scalable.

Broadcast using reverse path

As was discussed earlier, the greedy routing algorithm finds a path from a source node to a destination. Consider such paths from all nodes in S to a node $s \in S$. The union of the paths is a tree rooted at s. Therefore by reversing the direction of each path, we get a broadcast tree from a source node s to every other node in S. We introduce a simple broadcast protocol which utilizes the reverse-path tree. Note that our protocol does not require knowledge of the global triangulation over S. Each node u is assumed only to know its set of neighbor nodes, and determines to which node(s) it should forward a message based on its local knowledge.

[3]There are different measures of goodness in clustering algorithms. The main objective of this algorithm is to get clusters whose radii are within a given upper bound. For clusterings that satisfy the upper bound, a secondary measure may be the number of clusters. This algorithm does not optimize the number of clusters.

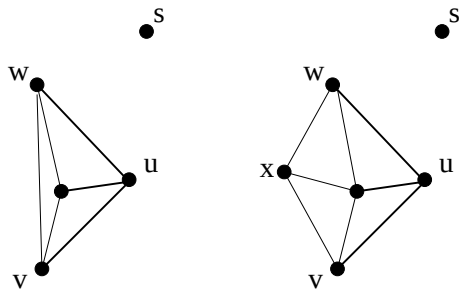

Figure 2. An ambiguous situation due to limited knowledge in GRPB.

The idea of using reverse path for broadcast goes back to as early as 1978 [13]. In the context of DT, HyperCast [5] is the first system to introduce the idea. Our protocol is different in that it is based on greedy routing in an arbitrary dimension while HyperCast is based on compass routing in a 2D space. The major advantage of both approaches is that a broadcast tree does not need to be explicitly maintained. A node can immediately determine next-hop nodes based on the coordinates of its neighbors and the destination node, without maintaining any per-session routing information.

We name our broadcast protocol as GRPB (greedy reverse path broadcast). In GRPB, a node u maintains *a local DT* for u and u's neighbors. For each neighbor v, u forwards a message from a source node s to v if both of the following two conditions hold:

C1 u is closer to s than v is;

C2 in the local DT for u and u's neighbor nodes, there does not exist a node $w \neq u$ such that: **C2.1** w is closer to s than u is, and **C2.2** u, v and w are included in the same triangle (or simplex in a d-dimensional space).

Condition C1 is easy to understand. Suppose C1 is true. Then u does not forward to v if u is sure that another node, say w, is the next hop of v in the forward greedy routing. The necessary and sufficient conditions for such w are: **C2.1** w is closer to s than u, and **C2.3** w is a neighbor of v on the global DT. However, u does not have global information and cannot check C2.3. Hence we specify condition C2.2 which includes C2.3. C2.1 and C2.2 are necessary but not sufficient.

Note that in case of a tie between w and u in C2.1, u must forward to v at the cost of possible duplication, since v may or may not choose u as the next hop in the forward greedy routing. Note also that even if node w appears to

be v's neighbor in u's local DT, w may not actually be v's neighbor in the global DT. Figure 2 illustrates an example in a 2D space. The left graph shows u's local DT, in which v and w are neighbors. However, as shown in the right graph, there may exist a node x outside u's local knowledge and thus w may not actually be a neighbor of v. Without including C2.2 in C2, u might erroneously conclude that it does not need to forward to v, since w appears to be the closest node to s among v's neighbors. C2.2 detects such ambiguous situations and requires that u forwards to v at the cost of possible duplication. We performed experiments to broadcast a message using GRPB on a distributed DT of 200 randomly-placed nodes in various dimensions. In our experiments, the number of duplicate messages was from 3% to 10% of the number of nodes.

The following theorem guarantees the correctness of GRPB, namely it delivers a message to all nodes in the system. A proof of the theorem as well as the protocol pseudocode are presented in [14].

Theorem 1 *Let a set of nodes S form a correct distributed DT. The GRPB protocol delivers a message from a source node $s \in S$ to all the other nodes in S.*

Multicast within a radius

In a distributed virtual reality system or a multiplayer online game, multicast within a given radius from a point may be a common operation, since an event may affect nodes within some distance. For example, in a war simulation, an explosion of a bomb will be seen only by soldiers within some distance, and will affect those within a shorter distance. We observe that in the GRPB protocol the distance of next hop from the source monotonically increases, since the distance to the destination monotonically decreases in the forward greedy routing. We utilize this observation in our multicast protocol within a given radius.

In our radius greedy reverse path multicast (RadGRPM) protocol from a source node s to all the other nodes within a given radius r, s first sends the message to all its neighbors within r. Then for each neighbor node v, a node u forwards a message to v if the following condition holds as well as C1 and C2 in GRPB:

C3 the distance from s to v does not exceed the radius r.

For pseudocode and correctness proof of RadGRPM, we refer interested readers to [14].

3. Protocol design

Our distributed DT protocols handle join, leave, and maintenance operations. Our join protocol ensures that a joining node obtains enough information to identify its correct neighbors and that the joining of the new node is notified to all existing nodes affected by the joining node, so that the resulting distributed DT is correct after protocol execution. Similarly, our leave protocol notifies the deletion of a leaving node to all affected nodes so that the resulting distributed DT is correct after protocol execution. Our join and leave protocols are proved to be correct only for serial joins and leaves.

We assume that nodes may join, leave or fail at any time. In addition to node failures, which inevitably result in an incorrect distributed DT, concurrent joins and leaves of multiple nodes may result in an incorrect distributed DT as well. To address such scenarios, we introduce a maintenance protocol which is run periodically to detect and repair any errors in the system state. (When the distributed DT of a set of nodes is incorrect, for convenience, we say "the system state is incorrect" or "the system state has errors.") Lastly, to simplify our protocol descriptions, we assume reliable delivery of protocol messages. In a real implementation, additional mechanisms such as ARQ or simply TCP can be used to ensure reliable message delivery.[4]

3.1. System model

Our approach to construct a distributed DT is as follows. We assume that each node is associated with its coordinates in a d-dimensional Euclidean space. Each node has prior knowledge of its own coordinates, as is assumed in previous work [5, 6, 7, 8, 9]. The mechanism to obtain coordinates is beyond the scope of this study. Coordinates may be given by an application, a GPS device[16], or topology-aware virtual coordinates[17].[5] Also when we say a node u knows another node v, we assume that u knows v's coordinates as well.

Let S be a set of nodes to construct a distributed DT from. We will present protocols to enable each node $u \in S$ to get to know a set of its nearby nodes including u itself, denoted as C_u, to be referred to as u's *candidate set*. Then u determines the set of its neighbor nodes N_u based on C_u. Specifically, u determines N_u by calculating a local DT of C_u, denoted by $DT(C_u)$. That is, $v \in N_u$ if and only if there exists an edge between u and v on $DT(C_u)$.

3.2. Correctness condition for a distributed Delaunay triangulation

Recall that a distributed DT is correct when for every node u, N_u is the same as the neighbors of u on $DT(S)$. Since N_u is the set of u's neighbor nodes on $DT(C_u)$ in our model, to achieve a correct distributed DT, the neighbors of u on $DT(C_u)$ must be the same as the neighbors of u on $DT(S)$. Note that C_u is local information of u while S is global knowledge. Therefore in designing our protocols, we need to ensure that C_u is "enough" for u to correctly identify its global neighbors. If C_u is too limited, u cannot identify its global neighbors. For the extreme

[4] Due to the overhead of opening and closing connections, TCP may not be a practical choice.

[5] Application performance on a DT may be affected by the accuracy of virtual coordinates.

case of $C_u = S$, u can identify its neighbors on the global DT since $DT(C_u) = DT(S)$; however, the communication overhead for each node to acquire global knowledge would be extremely high.

Theorem 2 (Correctness Condition) *Let S be a set of nodes and for each node $u \in S$, $u \in C_u$ and $C_u \subset S$. Let N_u, $u \in S$ be the set of u's neighbor nodes on $DT(C_u)$. A distributed DT of S is correct if and only if, for every $u \in S$, C_u includes all the neighbor nodes of u on $DT(S)$.*

Theorem 2 identifies a necessary and sufficient condition for a distributed DT to be correct, namely: the candidate set of each node must contain all of its global neighbors. A proof of the theorem is presented in [14]. In the following subsections, we use the above correctness condition as a guide to design our protocols.

3.3. Join protocol

In our join protocol,[6] we assume that a joining node n is first led to the nearest existing node u, which is guaranteed to be found using greedy routing as discussed in section 2.2. C_n is initialized as $\{n\}$, and n sends a NEIGHBOR_SET_REQUEST messages to u. When u receives NEIGHBOR_SET_REQUEST from n, u puts n into C_u, updates N_u by recalculating $DT(C_u)$, computes N_n^u which is the set of the neighbor nodes of n on $DT(C_u)$, and replies N_n^u to n. When n receives the reply, C_n is updated to include all nodes in the reply, and n determines its neighbor nodes again using the updated C_n. If n finds any new neighbor nodes, n sends NEIGHBOR_SET_REQUEST messages to them. This process is repeated until n does not find any new neighbor node. The protocol pseudocode is presented in [14].

Theorem 3 guarantees that the join protocol, if run on a correct distributed DT, results in a correct distributed DT for a single join. The main ideas of the proof are the following: i) the closest existing node will be a neighbor of a joining node, ii) all neighbor nodes of the joining node are connected by existing neighbor relations, thus it is possible to find them all by following the neighbor relations, and iii) the neighbor nodes of the joining node are also notified of the joining node's addition in the process. A detailed proof of the theorem is presented in [14]. Note that this proof is based on Theorem 2, which determines the condition when a distributed DT is correct.

Theorem 3 *Let S be a set of existing nodes and the distributed DT of S be correct. Let a node $n \notin S$ join to the distributed DT using our join protocol. Assume that there is no other join, leave, or failure. After the join protocol finishes, the updated distributed DT is correct.*

Though the join protocol achieves a correct distributed DT after it finishes, the transient states are not correct, which may result in malfunction of upper-layer applications. For example, a new node in an early stage of the joining process may not have a complete set of neighbors and may not be able to properly forward a message for greedy routing. To address such situations, we introduce a mechanism for a joining node to defer to be a part of the system until it establishes its complete set of neighbor nodes. When an existing node receives NEIGHBOR_SET_REQUEST, it does not immediately update its neighbor set. When the joining node n finishes its joining process, it then notifies all its neighbors that it is safe to update their candidate sets and their neighbor sets to include n. Due to delay of notification message delivery, some transient states may still be incorrect. However, greedy routing will work well even with imperfect states, as to be shown in section 4.

Also note that the join protocol is proved to be correct only for serial joins. In case of concurrent joins, the protocol may not result in a correct distributed DT. Such imperfection is addressed by the maintenance protocol to be presented in subsection 3.5.

3.4. Leave protocol

We first address the case of graceful leaves. The case of ungraceful leaves or failures is addressed by our maintenance protocol in subsection 3.5. A straightforward approach to address graceful leave would be that a leaving node, before it leaves, notifies all of its neighbors that it is about to leave. This simple notification is, however, not enough to maintain a correct distributed DT.

Suppose that a node u leaves and it notifies a neighbor node v that it is leaving. Then v should remove u from C_v and update N_v. The problem is that in some cases v may have a new neighbor w that was not previously a neighbor of v and may not be in C_v. In such cases, the straightforward approach may resulting in an incorrect distributed DT. However, we observe that such w is always a neighbor of u. Therefore it is possible for u to notify v that u is leaving and also introduce w to v, resulting in a correct distributed DT.

When a node u leaves, u calculates a local DT of its neighbor nodes, but not including itself. Then u notifies each of its neighbors, say v, that u is leaving as well as a list of the neighbors of v on the local DT of u. Upon receiving such notification, v updates its candidate set and neighbor set. In addition, a DELETE message that u is leaving is propagated using GRPB. Note that even if u is not a neighbor node of another node x, x may have u in C_x. The DELETE message ensures that u is removed from such C_x, if any. The protocol pseudocode is presented in [14].

The following theorem assures that the leave protocol is correct for serial leaves. The theorem is based on the previous observation that if a node w becomes a new neighbor of v after u leaves, w was a neighbor of u before u leaves.

[6] This protocol is similar to the basic generalized join algorithm in [7].

A complete proof of the theorem is found in [14].

Theorem 4 *Let S be a set of nodes and the distributed DT of S be correct. Let a node $u \in S$ leave the distributed DT using our leave protocol. Assume that there is no other join, leave, or failure. After the leave protocol finishes, the updated distributed DT is correct.*

Note that the leave protocol is correct only for serial leaves. Similar to the case of concurrent joins, concurrent leaves may result in an incorrect distributed DT. Such cases are addressed by our maintenance protocol, to be discussed in subsection 3.5. In our implementation, propagation of a DELETE message is stopped when the message arrives at a node that does not have the leaving node in its candidate set. This modification greatly reduces communication cost, without affecting correctness of the leave protocol in almost all cases. A very rare case where a left node remains in a candidate set and causes incorrectness can be addressed by the maintenance protocol.

Also, similar to the case of a join, transient incorrect states during a leave may result in malfunction of upper-layer applications. Thus it is desirable for a leaving node to defer leaving after making sure that each of its neighbors has updated its neighbor set. This may be achieved by requiring an acknowledgement of a LEAVE message.

3.5. Maintenance protocol

The join and leave protocols are proved correct only for serial joins and leaves, assuming that there is no other concurrent join, leave, or failure. In practice, however, nodes may join and leave concurrently, or even fail at any time, causing errors in the system state. Therefore an additional mechanism is needed to repair errors in the system state. To address system churn and failures, we present a maintenance protocol, which is run periodically to detect and repair errors, if any, in the system state.

From Definition 5, for a distributed DT to be correct, two conditions must be satisfied: i) Each node u must include in its neighbor set N_u all of its neighbors on the global DT, and ii) N_u must not contain any node that is not in the system.

To satisfy the first condition, a node periodically exchanges information with each of its neighbors. Specifically, a node u informs its neighbor node v of the neighbors of v on u's local DT. Note that the process is essentially the same as what is done when a node joins, since the goals of the two protocols are same i.e. each node learns its neighbors on the global DT.

To satisfy the second condition, a node probes its neighbors by sending ping messages periodically. If a probed node does not reply, the node is considered to be not in the system and removed from the candidate set. This mechanism also addresses the case of ungraceful node failures. Note that this probing can be easily integrated with the exchange of information for the first condition.

The maintenance protocol is as follows. A node u sends out NEIGHBOR_SET_REQUEST to its neighbor node v. When v receives the request, it replies with N_u^v, which is the set of neighbors of u on $DT(C_v)$. That is, N_u^v is the set of u's neighbors in v's local view. v also checks whether u is in its candidate set C_v. If $u \notin C_v$, v puts u in C_v. When u receives the reply N_u^v, u checks whether $N_u^v \subset C_u$. If there exists any node in N_u^v that is not in C_u, it is added to C_u. In case u does not receive a reply from v before TIMEOUT, v is considered to have failed and removed from C_u. u also propagates the deletion of v similarly as in the leave protocol. Once C_u is updated, u recalculates the local DT and determines its set of neighbor nodes N_u. If there are any new neighbor nodes in N_u, u sends NEIGHBOR_SET_REQUEST to them. The protocol pseudocode is presented in [14].

From a large number of simulation experiments, we found that the maintenance protocol converged to a correct distributed DT in every experiment, for different dimensionalities (2D to 6D), numbers of nodes (200 to 800), scenarios (random initial graph, severe churn with node failures) as long as the system is not partitioned. Note that it is extremely difficult to prove correctness of the maintenance protocol for any combinations of concurrent joins, leaves and failures. Furthermore, in an environment where system churn occurs continually, another join or leave may occur before the system converges to a correct state. As a result, convergence to a correct system state may be impossible during system churn. Fortunately, some applications can still benefit from an imperfect DT as long as it is "accurate" enough. Therefore the accuracy of a distributed DT over time is more important in practice than eventual convergence to a correct distributed DT for systems under churn.

Even if our maintenance protocol converged to a correct distributed DT some time after churn and failure have stopped in every one of our experiments, this does not mean that our join and leave protocols are no longer needed. Note that it takes time for the maintenance protocol to detect and repair errors, resulting in a lower average accuracy. Furthermore, the maintenance protocol requires a much higher communication overhead than those of the join and leave protocols, and thus should be run only periodically, with the period being a design parameter to be tuned.

We define an accuracy metric of a distributed DT as follows, which is used for all of our experiments. Let DDT_S be a distributed DT of a set of in-system nodes S. We consider a node to be in-system from when it finishes joining to when it starts leaving. (Note that some nodes may be in the process of joining or leaving and not included.) Let $N_{correct}(DDT_S)$ be the number of correct neighbor entries of all nodes and $N_{wrong}(DDT_S)$ be the number of wrong neighbor entries of all nodes on DDT_S. A neigh-

bor entry v of a node u is correct when v is a neighbor of u on the global DT (namely, $DT(S)$), and wrong when u and v are not neighbors on the global DT. Let $N(DT(S))$ be the number of edges on $DT(S)$. Note that edges on a global DT are undirectional and thus are counted twice to be compared with neighbor entries. The accuracy of DDT_S is defined as follows: accuracy(DDT_S) $= \frac{N_{correct}(DDT_S) - N_{wrong}(DDT_S)}{2 \times N(DT(S))}$. A distributed DT is correct if and only if its accuracy is 1.

To demonstrate accuracy and effectiveness of the maintenance protocol, we designed a "ring" scenario beginning with a barely connected graph, in which each node initially knows only one other node. That is, node p_i, $i \geq 1$, has only p_{i-1} in its candidate set and its neighbor set. The number of nodes is 200. Figure 3 shows accuracy of the distributed DT versus time as the maintenance protocol runs. Note that the maintenance protocol achieved a correct distributed DT within a few rounds of protocol execution. The convergence tends to be faster in a higher dimension space, since nodes have more neighbors and information is exchanged faster.

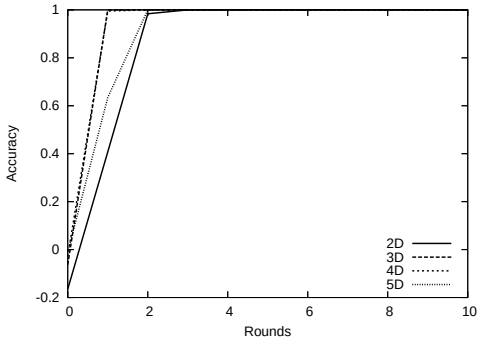

Figure 3. Accuracy of the maintenance protocol in a "ring" scenario.

4. Experimental results

4.1. Scalability

The per-node communication cost of our distributed DT protocols largely depends on the average number of neighbors per node. Since the average number of neighbors of a node on a DT does not increase as the number of nodes in the system increases,[7] the scalability of our distributed DT protocols is generally very good. However, there are two minor factors that affect the per-node cost as the system size increases. First, greedy routing to locate the closest existing node in the join process will take $O(\sqrt[d]{n})$ steps, where d is the dimensionality of the space and n is the number of nodes in the system. In addition, nodes on the boundary of a DT have fewer neighbors than those in the middle.

[7]For a DT of randomly-placed nodes, the average number of neighbors asymptotically depends on the dimensionality of the space.

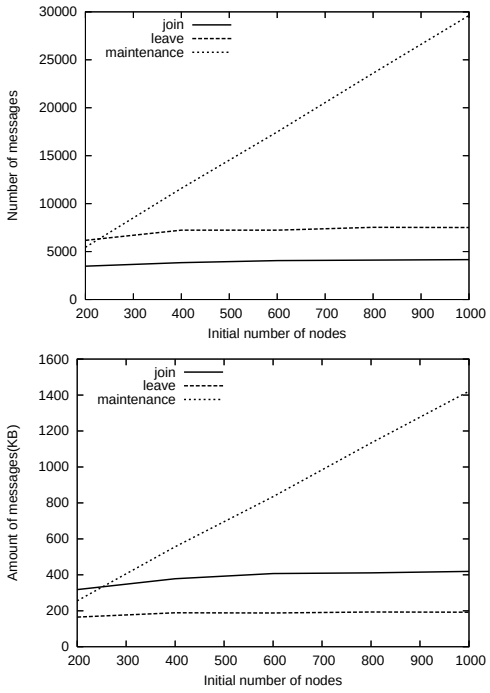

Figure 4. The communication cost of protocols versus number of nodes in 3D.

When the network size is smaller, the fraction of boundary nodes is larger, making the average number of neighbors smaller. Figure 4 shows the number of messages (top) and the amount of messages in Kbytes (bottom) versus system size in 3D. The join and leave curves represent the total costs of 100 serial joins and 100 serial leaves respectively, and are more or less independent of system size showing very good scalability. The maintenance curve represents the per-round cost of the maintenance protocol *for all nodes*, which increases linearly with system size; thus the average cost per node is constant versus system size.

4.2. Performance under churn

Figure 5 (top) shows accuracy of a distributed DT as a function of time for a system under churn, more specifically, when nodes are joining and leaving concurrently but not failing. Initially, the system has 200, 400, or 800 nodes with a correct distributed DT. Then node joins are scheduled following a Poisson process with average inter-arrival time of 1 second, until time 110 second. Graceful node leaves are also scheduled in the same way. That is, 1 node joins and 1 node leaves once every second on the average, using our join and leave protocols.[8] Our maintenance protocol is run once every 10 seconds. In spite of the churn, accu-

[8]By Little's Law, for an initial system size of 200, the average lifetime of a node is 200 seconds. For P2P systems, this is a very high churn rate [15]. Note that accuracy in Figure 5 (top) improves as the system size increases.

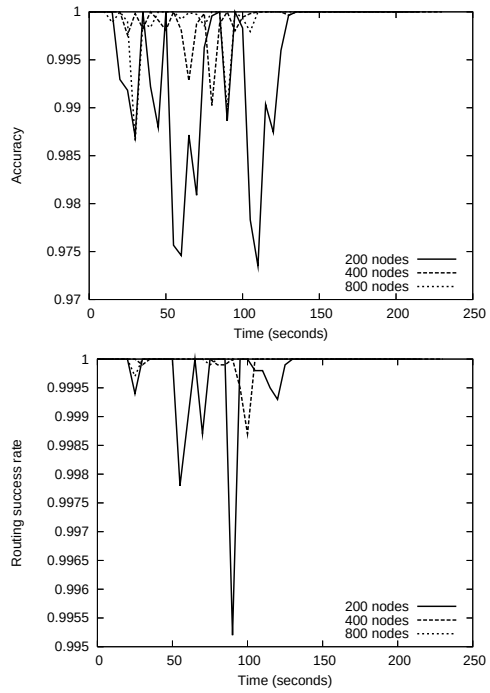

Figure 5. Accuracy (top) and greedy routing success rate (bottom) of a system in 3D under churn versus time.

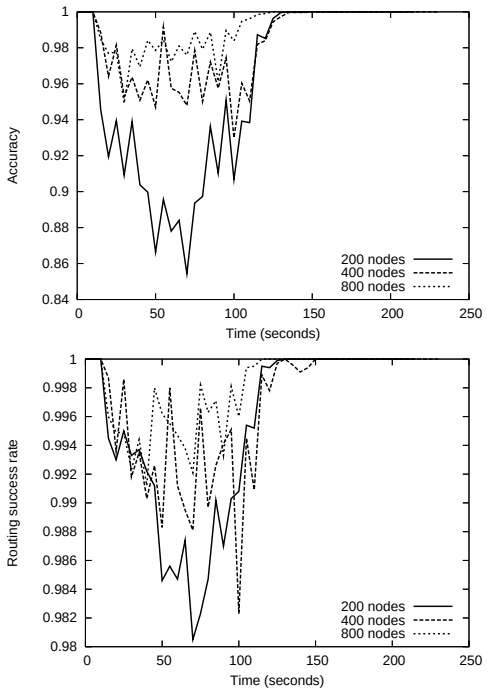

Figure 6. Accuracy (top) and greedy routing success rate (bottom) of a system in 3D with concurrent joins and failures versus time.

racy of the distributed DT remains very high. The small error is due to concurrent joins and leaves, and is repaired by our maintenance protocol periodically. Figure 5 (bottom) shows the success rate of a greedy routing protocol for a system under churn while running our join, leave and maintenance protocols. Note that the success rate is much higher than the accuracy value, due to careful design of our join and leave protocols. In our join protocol, the neighbor nodes of a joining node defer adding the joining node to their neighbor sets until the joining node finishes its joining process and is ready to function properly. Similarly, in our leave protocol, a leaving node continues service until all of its neighbors are notified.

4.3. Performance with node failures

Figure 6 shows accuracy of distributed DT (top) and greedy routing success rate (bottom) for a system in which nodes join and fail concurrently, while running our join and maintenance protocols, but not our leave protocol. Except for nodes failing instead of leaving, we use the same simulation parameters as in subsection 4.2 to compare the impact of node failures versus graceful node leaves.

Both accuracy and greedy routing success rate are much worse than in the previous case of system churn with graceful leaves instead of failures. Since any error caused by a failed node cannot be recovered until the maintenance protocol detects it by a message timeout, the lower accuracy and routing success rate are to be expected considering the high failure rate. Lastly, note that in all simulations the maintenance protocol converged to a correct system state in a few rounds after churning stopped.

5. Related work

The first protocol to construct a distributed DT was proposed by Liebeherr and Nahas [5]. The protocol utilizes the locally equiangular property of DT in a 2D space. Nodes are assumed to have pre-assigned logical coordinates in a 2D space. Each node checks whether the equiangular property holds among itself and its neighbor nodes. Whenever a violation is detected, the node flips triangles to maintain a correct DT. Their application was application-layer multicast, called HyperCast. Since compass routing on a DT is guaranteed to succeed, a multicast tree can be implicitly determined for a given source using reverse path.

Steiner and Biersack [6] proposed a distributed approach to construct a distributed DT in a 3D space. The tetrahedron which includes a joining node is determined and split. Then the new tetrahedra are checked whether they include any nodes in their circumspheres and flipped if necessary.

Simon et al. [7] proposed two sets of distributed algorithms for d-dimensional spaces: basic generalized algorithms and improved generalized algorithms.[9] Each set

[9]The improved generalized insertion algorithm is designed to lower

consists of an entity insertion algorithm and a deletion algorithm. They assume that no $d+1$ nodes are on the same hyperplane and no $d+2$ nodes are on the same hypersphere. It is also assumed that a new node is in the interior of the convex hull of existing nodes. Our join protocol is similar to their basic generalized insertion algorithm, but we proved correctness for our join and leave protocols without any of their assumptions. Their improved generalized algorithms, which are the focus of their paper, use a fundamentally different approach from ours; in particular, our protocols are based on our correctness condition and theirs are based on an already-proven centralized flip algorithm. In addition, while their work is focused on explaining correct operation of their algorithms in different scenarios for a single insertion or deletion,[10] we rigorously proved correctness for our join and leave protocols. We also performed simulation experiments to measure the communication cost of our protocols and to investigate accuracy under concurrent node joins and leaves.

While DT has been extensively studied in computational geometry, most work in the field focuses on centralized algorithms. Ohnishi et al. [8] proposed an incremental algorithm to construct a distributed DT in a 2D space. Yoo et al. [9] proposed a distributed algorithm to maintain DT for moving nodes in a 2D space.

6. Conclusions

While DT has been known and used for a long time, the design of protocols for constructing and maintaining a DT for a dynamic system has not received much attention. In this paper, we investigate the design of join, leave and maintenance protocols for a set of nodes to construct and maintain a distributed DT dynamically, as well as several application-level protocols to support DT applications. We present a necessary and sufficient condition for a distributed DT to be correct, which was used as a guide for our protocol design. Experimental results show that our protocols are scalable and they achieve high accuracy for systems under churn and with node failures.

References

[1] B. Delaunay, Sur la sphère vide, Otdelenie Matematicheskii i Estetvennyka Nauk, 7:793-800, 1934.

[2] G. Voronoï, Nouvelles applications des paramètres continus à la théorie des formes quandratiques, Journal für die Reine and Angewandte Mathematik, 133:97-178, 1908.

[3] Prosenjut Bose and Pat Morin, Online routing in triangulations, SIAM Journal on Computing, 33(4):937-951, 2004.

computational cost by reducing duplicated computation among nodes.

[10] A version of their paper submitted for publication has correctness proof for their improved entity deletion algorithm for 3D only.

[4] D.P. Dobkin, S.J. Friedman, and K.J. Supowit, Delaunay graphs are almost as good as complete graphs, Discrete Computational Geometry, 5:399–407, 1990.

[5] Jorg Liebeherr, Michael Nahas, Application-Layer Multicasting With Delaunay triangulation Overlays, IEEE Journal on Selected Areas in Communications, VOL. 20, NO. 8, October 2002.

[6] Moritz Steiner, Ernst Biersack, A fully distributed peer to peer structure based on 3D Delaunay Triangulation, Algotel 2005.

[7] Gwendal Simon, Moritz Steiner, Ernst Biersack, Distributed Dynamic Delaunay Triangulation in d-Dimensional Spaces, Technical report, Institut Eurecom, August 2005.

[8] Masaaki Ohnishi, Ryo Nishide, Shinichi Ueshima, Incremental Construction of Delaunay Overlaid Network for Virtual Collaborative Space, The third international Conference on Creating, Connecting and Collaborating through Computing, 2005.

[9] Taewon Yoo, Hyonik Lee, Jinwon Lee, Sunghee Choi, Junehwa Song, Distributed Kinetic Delaunay Triangulation, CS/TR-2005-240, KAIST, Korea, 2005.

[10] Min Sik Kim, Taekhyun Kim, Yong-June Shin, Simon S. Lam, Edward J. Powers, Scalable clustering of Internet paths by shared congestion, IEEE Infocom 2006, April 2006.

[11] Xin Yan Zhang, Qian Zhang, Zhensheng Zhang, Gang Song, Wenwu Zhu, A Construction of locality-aware overlay network: mOverlay and its performance, IEEE journal on selected areas in communications, Vol. 22, No. 1, Jan 2004.

[12] X. Brian Zhang, Simon S. Lam, Huaiyu Liu, Efficient Group Rekeying Using Application Layer Multicast, Proceedings of 25th IEEE ICDCS, Columbus, Ohio, June 2005.

[13] Yogen K. Dalal and Robert M. Metcalfe, Reverse path forwarding of broadcast packets, Communications of the ACM, Volume 21, Issue 12, Dec 1978.

[14] Dong-Young Lee and Simon S. Lam, Protocol design for dynamic Delaunay triangulation, The Univ. of Texas at Austin, Dept. of Computer Sciences, Technical Report TR-06-48, Oct 2006.

[15] S. Sariou, P. K. Gummadi, and S. D. Gribble, A measurement study of peer-to-peer file sharing systems, In Proc. of Multimedia Computing and Networking, January 2002.

[16] Tommaso Melodia, Dario Pompili, Ian F. Akyildiz, A Communication Architecture for Mobile Wireless Sensor and Actor Networks, Proceedings of IEEE SECON 2006, September 2006.

[17] T. S. Eugene Ng and Hui Zhang, Predicting Internet Network Distance with Coordinates-Based Approaches, Proceedings of INFOCOM 2002.