Protocol Design for Dynamic Delaunay Triangulation*

Dong-Young Lee and Simon S. Lam
Department of Computer Science
The University of Texas at Austin

TR-06-48 December 1, 2006

*Theorem 4 (Correctness Condition) herein is presented as Theorem 1 in TR-07-59 (revised, September 1, 2008) where a more structured and detailed proof of the theorem is provided in its Appendix.

Protocol Design for Dynamic Delaunay Triangulation *

Dong-Young Lee and Simon S. Lam

{dylee, lam}@cs.utexas.edu

Department of Computer Sciences

The University of Texas at Austin

Abstract

Delaunay triangulation (DT) is a useful geometric structure for applications such as routing, clustering, broadcast, distributed virtual reality systems, and multiplayer on-line games. 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. (Conceptually nodes are points in a Euclidean space.) 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. To support applications of distributed DT, we present protocols for greedy routing, clustering, broadcast, and multicast within a radius. Each node in our greedy routing, broadcast and multicast protocols does not maintain any per-session state. We also discuss and prove correctness for the application protocols.

1 Introduction

With almost a hundred years of history, DT [1] and Voronoi diagram [2] have been widely used in many applications in different fields of science and engineering, including computer science. A triangulation in 2D space means, for a given set of nodes, constructing edges between pairs of nodes such that the edges

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

form a non-overlapping set of triangles that cover the convex hull of the nodes. DT in 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, 5, 6].

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.[1] 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.

[1] We will define a distributed DT and its correctness more carefully in section 2.

In practice, nodes may join and leave concurrently. Furthermore, nodes may fail at any time, immediately breaking 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 [13] or a complete graph of clusters [14]. 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 monotonically increases in GRPB, 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 to make sure that 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 it 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 a correct distributed DT some time after churn and failures stop.

The organization of this paper is as follows. In section 2, we introduce concepts and defintions of distributed DT and also present application protocols. In section 3, we present a necessary and sufficient correctness condition for a distributed DT, which was used as a guide to design our protocols. The join and leave protocols are presented and proved correct for serial joins and leaves. Our maintenance protocol is then presented as well as an accuracy metric for evaluating protocol performance. In section 4, experimental results are presented to demonstrate scalability of our protocols and their performance for systems under churn and with node failures. 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[7, 10, 11] and 3D[8] spaces.)

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 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

We first define a Voronoi diagram.

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 $D(x,y)$ denote 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. We say that two Voronoi cells are neighbors of each other if they share a common facet.

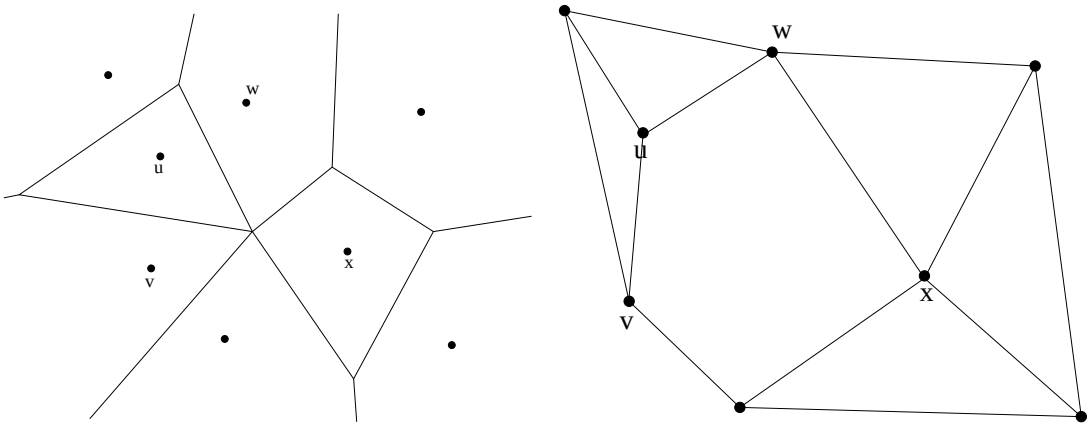
Definition 2. *Consider a set of nodes S in a Euclidean space. $VC_S(u)$ and $VC_S(v)$ are neighboring Voronoi cells, or* **neighbors** *of each other, if and only if $VC_S(u)$ and $VC_S(v)$ share a facet.*

Figure 1(a) shows a Voronoi diagram in a 2-dimensional space. 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 $VC_S(x)$ shares only a point. Similarly, in a 3-dimensional space, Voronoi cells that share only an edge or a point are not neighbors.

Then we define DT as follows.

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.*

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. Figure 1(b) shows the DT of nodes in Figure 1(a). Note that facets of a Voronoi cell



(a) Voronoi diagram (b) Delaunay triangulation

Figure 1: A Voronoi diagram and the corresponding DT in a 2-dimensional space.

perpendicularly bisect edges of a DT. Therefore, a DT is the dual of a Voronoi diagram.[2] Let us denote the Voronoi diagram of S as $VD(S)$, and the DT of S as $DT(S)$.

By a distributed DT, we mean that each node $u \in S$ maintains a set N_u of its neighbor nodes.

Definition 4. *A* **distributed Delaunay triangulation** *of a set of node S is specified by $\{< u, N_u > | u \in S\}$, 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 $u, v \in S$:*

- *if there exists an edge between u and v on the global DT of S, $v \in N_u$ and $u \in N_v$,*
- *if there does not exist an edge between u and v on the global DT of S, $v \notin N_u$ 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.

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

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

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 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, 5, 6].

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.

Consider the problem of finding the closest existing node (destination) $d \in S$ to a given point $n \notin S$, starting from a given node $s \in S$. If there are more than one closest nodes to n, the destination may be any one of them. Let v_0 be s. At v_i, the greedy routing algorithm selects the next-hop node v_{i+1} which is closest to n among the neighbor nodes of v_i. If v_{i+1} is closer to n than v_i, greedy routing is repeated at v_{i+1}. Otherwise, routing stops at v_i, which is denoted as v_k. If v_k is the closest node or one of the closest nodes to n, we say the routing succeeds; otherwise we say it fails. In other words, the routing succeeds if $n \in VC_S(v_k)$.

The following theorem shows that the greedy routing algorithm always succeeds as long as it is run on a DT. Bose and Morin [3] proved a similar theorem that greedy routing between *nodes* always succeeds on DT. We use an approach similar to theirs to prove the following theorem.

Theorem 1. *Finding a closest node $d \in S$ to a given point $n \notin S$ using greedy routing always succeeds on a DT of S.*

Proof. We prove by showing that every node $v \neq d$ in Delaunay triangulation has a neighbor that is closer to n. Suppose that $v \neq d$. Draw a straight line L from v to n, and let P the first Voronoi facet which L crosses. Let u be the node in the adjacent Voronoi cell which shares P with v. Therefore there is an edge between v and u in the Delaunay triangulation. Note that P divides the space into two regions S_u^p and S_v^p; points in S_u^p is closer to u than to v. Since n belongs to S_u^p, n is closer to u than v. Therefore if $v \neq d$, v has a neighbor that is closer to n. On the other hand, if $v = d$, the routing stops at v. Since there are a finite number of nodes, eventually a closest node d is found in a finite number of steps. □

Clustering of network nodes

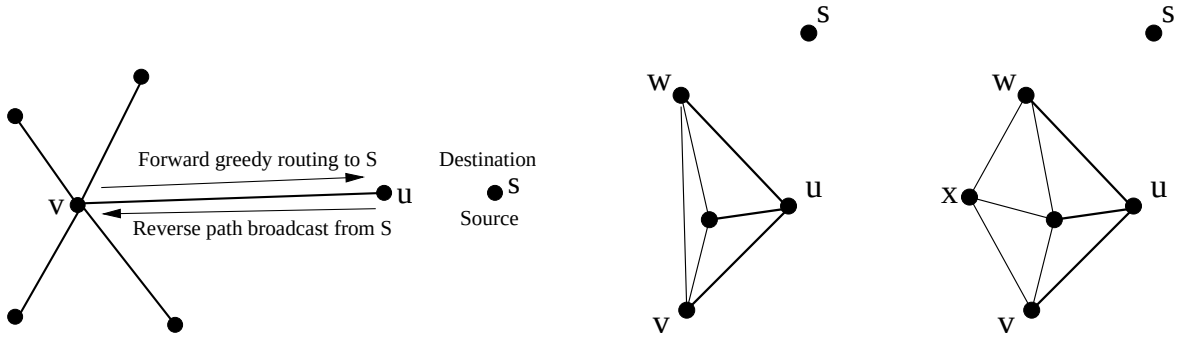
To illustrate an application of finding the closest existing node to a given point, we present a simple clustering protocol of network nodes. The protocol is a distributed version of a clustering algorithm adopted from [12]. The upper bound R of the radius of a cluster is given as a parameter. Nodes are considered sequentially 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 whether 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 [12].

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 [13], clusters form a random graph and a joining node may fail to find the closest existing cluster. In [14], every node maintains links to every other clusters, limiting scalability. The scalability issue is addressed in [14] 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. 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. Figure 2(a) illustrates an example of a reverse path. In forward greedy routing, v selects u as the next hop, since u is its closest neighbor to the destination s. Thus in reverse path broadcast from the source node s, u should forward a message to v,, below line revised if u knows that u is the next hop of v in the forward route. Note that s is the destination in forward greedy routing and the source in reverse path broadcast.



(a) Forward path and reverse path

(b) Ambiguous situation due to limited knowledge

Figure 2: Forwarding in GRPB

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. Specifically, node u in the previous example may not know all the neighbors of v. u only knows the neighbors of u, but still has to determine whether u is the closest node to s among v's neighbor nodes.

The idea of using reverse path for broadcast goes back to as early as 1978 [15]. In the context of DT, Hypercast [7] 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 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 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 conditions for such w are:

C2.1 w is closer to s than u

C2.2 u, v, and w are included in the same triangle (or simplex) in u's local DT

C2.3 w is a neighbor of v on the global DT

Note that C2.1 and 2.3 are necessary and sufficient. However, u does not have global information and cannot check C2.3. Hence we specifiy 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(b) illustrates an example in 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. The protocol pseudocode is given in Figure 3.

The following theorem guarantees the correctness of GRPB, namely it delivers a message to all nodes in the system. As explained before, the two conditions of GRPB are necessary, but not sufficient. Therefore some duplicate messages may be forwarded. We performed experiments to broadcast a message using

```
Start_broadcast(msg) of node u
; u is a source node
for all v ∈ N_u do
  Send(v, BROADCAST(msg, u))
end for

On u's receiving BROADCAST(msg, s)
; u is a recipient of a BROADCAST message
Deliver(msg)
for all v ∈ N_u do
  if v satisfies conditions C1 and C2 from s then
    Send(v, BROADCAST(msg, s))
  end if
end for
```

Figure 3: Greedy reverse path broadcast (GRPB) protocol at a node u.

GRPB on a distributed DT of 200 randomly placed nodes in various dimensions. Ideally the number of messages for each broadcast should be the number of nodes minus 1 when there is no duplication. In our experiments, the number of duplicate messages was from 3% to 10% of the number of nodes.

Theorem 2. *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.*

Proof. We prove the theorem by showing that if there exists an edge from u to v in the global reverse path tree, the GRPB protocol also forwards a message from u to v.

Assume that the theorem is not true. Suppose that a node u fails to forward to its neighbor v when there exists an edge from u to v in the global reverse path tree, that is when u is the closest node from s among the neighbors of v. Note that v is a neighbor of u on the local Delaunay triangulation of u. Then there exists a node w which is a mutual neighbor of u and v on the local Delaunay triangulation of u, and the distance between w and s is shorter than the distance between u and s, but w is not a neighbor of v on the global Delaunay triangulation. (If w is a neighbor of v, the next hop of v in the forward path should not be u since u is not the closest to s among v's neighbors.) On the local Delaunay triangulation of u, remember that there exists a simplex which includes u, v and w. Let the simplex p. Note that p does not exist on the global Delaunay triangulation, since w is not a neighbor of v. and then the space of p is occupied by other simplexes. Let x one of the simplexes and which includes u and v. Let $x_1 ... x_k$ the other nodes of x other than u or v. Then $x_1 ... x_k$ are neighbors of u in the global Delaunay triangulation and in the local Delaunay triangulation of u. Then on the local Delaunay triangulation of u, since v and

$x_1 ... x_k$ are neighbors of u, there exists the same simplex x. It is impossible that x and p co-exist on the local Delaunay triangulation of u, since they overlap. □

Multicast within a radius

In a distributed virtual reality system or a multiplayer on-line game, an entity or a player interacts with other entities or players that are located around it in the virtual space. Suppose that entities or characters in a distributed virtual reality system or a multiplayer on-line game are represented as nodes. Then the DT of the nodes is a good interconnection topology since neighbors of a node in DT are nodes that surround the node in the virtual space.

In addition to interaction between neighboring nodes, multicast within a given radius from a point is another 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 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 radius r, s first sends the message to all its neighbors within the radius 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.

Essentially the protocol is the same as the original GRPB protocol, except that forwarding stops when the distance from the source exceeds the given radius in C3. Pseudocode of the protocol is given in Figure 4. Theorem 3 guarantees that RadGRPM delivers the message to all nodes within a given radius. The proof is straightforward since the distance from the source node monotonically increases whenever the message is forwarded by GRPB.

Theorem 3. *Let a set of nodes S form a correct distributed DT. The RadGRPM protocol delivers a message from a source node $s \in S$ to all nodes within a radius r from s.*

Proof. By Theorem 2, the original GRPB protocol delivers a message to all the other nodes in S. Since the distance from s monotonically increases whenever a message is forwarded and the forwarding stops when the distance from s exceeds r, all the nodes along the original multicast path after stopping have

```
Start_radius_broadcast(msg, rad) of node u
; u is a source node
for all v ∈ N_u within rad from u do
  Send(v, BROADCAST(msg, rad, u))
end for

On u's receiving BROADCAST(msg, rad, s)
; u is a recipient of a BROADCAST message
Deliver(msg)
for all v ∈ N_u do
  if v satisfies conditions C1, C2 and C3 from s then
    Send(v, BROADCAST(msg, rad, s))
  end if
end for
```

Figure 4: The radius greedy reverse path multicast(RadGRPM) protocol at a node u.

distances longer than r from s. Therefore the RadGRPM protocol delivers the message to all the nodes within the radius r. □

3 Protocol Design

Our distributed DT protocols consist of a join, a leave, and a maintenance protocol. 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.[3]

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

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-dimentional Euclidean space. Each node has prior knowledge of its own coordinates, as is assumed in previous work [7, 8, 9, 10, 11]. The mechanism to obtain coordinates is beyond the scope of this study. Coordinates may be given by an application, a GPS device[18], or topology-aware virtual coordinates[19].[4] 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. 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 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. Before we present Theorem 4, which identifies a necessary and sufficient condition for a distributed DT is correct, we present several lemmas for convenience of proof.

Lemma 1. *Let S be a set of nodes. Let $v \in S$ be a neighbor node of $u \in S$ on $DT(S)$. Then there exists a point p in $VC_S(u)$ such that $D(p,u) < D(p,v) < D(p,w)$ for any other node $w \in S, w \neq u, w \neq v$.*

Proof. Consider a point p' on the shared facet of $VC_S(u)$ and $VC_S(v)$. Then $D(p',u) = D(p',v) < D(p',w)$ for any other node $w \in S, w \neq u, w \neq v$. Let w_1 be the third closest node from p' in S and let $\Delta = D(p',w_1) - D(p',v)$. Let p be the point that is $\frac{\Delta}{4}$ away from p' toward u. Then $D(p,u) < D(p,v) <$

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

$D(p,w)$ for any other node $w \in S, w \neq u, w \neq v$. □

Lemma 2. *Let S be a set of nodes. If there exists a point p in $VC_S(u)$ such that $D(p,u) < D(p,v) < D(p,w)$ for any other node $w \in S, w \neq u, w \neq v$, Then $u, v \in S$ are neighbors of each other on $DT(S)$.*

Proof. Consider a line from p toward v. As a point p' moves along the line, $D(p',v)$ will decrease toward 0 while $D(p',u) \geq 0$. In addition, $D(p',v)$ decreases faster than $D(p',w)$ decreases for any other node $w \in S, w \neq u, w \neq v$. Therefore there must be a point where $D(p',u) = D(p',v) < D(p',w)$ for any other node $w \in S, w \neq u, w \neq v$, which means that p' belongs to exactly two Voronoi cells $VC_S(u)$ and $VC_S(v)$, but not other Voronoi cells. By Observation **??**, p' is on the shared facet of $VC_S(u)$ and $VC_S(v)$. □

Lemma 3. *Let S be a set of nodes. Let $u \in C$, $v \in C$, and $C \subset S$. If v is a neighbor of u on $DT(S)$, v is also a neighbor of u on $DT(C)$.*

Proof. By Lemma 1, there exists a point p where $D(p,u) < D(p,v) < D(p,w)$ for any other node $w \in S, w \neq u, w \neq v$. Since $C \subset S$, $D(p,u) < D(p,v) < D(p,w)$ for any other node $w \in C, w \neq u, w \neq v$. Therefore by Lemma 2, v is a neighbor of u on $DT(C)$. □

Lemma 4. *Let S be a set of nodes. Let $u \in S$ and $C_u \subset S$ include all the neighbor nodes of u on $DT(S)$. If $v \in C_u$ is a neighbor of u on $DT(C_u)$, then v is also a neighbor of u on $DT(S)$.*

Proof. When $v \in S$ is a neighbor of u on $DT(C_u)$, by Lemma 1, there exists a point p in $VC_{C_u}(u)$ such that $D(p,u) < D(p,v) < D(p,w)$ for any other node $w \in C_u, w \neq u, w \neq v$. Now, suppose that v is not a neighbor of u on $DT(S)$. Then there must be a node $x \in S, x \notin C_u, x \neq u, x \neq v$ that satisfies $D(p,v) \geq D(p,x)$. Let $x_1, ..., x_k$ be those nodes which satisfy such condition. That is, $D(p,u), D(p,x_1), ..., D(p,x_k) < D(p,w)$ for any other node $w \in S, w \neq u, w \neq x_i, 1 \leq i \leq k$. We show below that there exists a node $x_i, 1 \leq i \leq k$ which is a neighbor of u on $DT(S)$. Since $x_i \notin C_u$, it is contradictory to the assumption that C_u includes all the neighbor nodes of u on $DT(S)$. Therefore v is a neighbor of u on $DT(S)$.

Case A-1. Suppose that $D(p,u) < D(p,x_1) < D(p,w)$ for any other node $w \in S, w \neq u, w \neq x_1$. Then by Lemma 2, x_1 is a neighbor of u on $DT(S)$.

Case A-2. Suppose that $D(p,u) < D(p,x_1) = ... = D(p,x_h) < D(p,w_1) \leq D(p,w)$ for any other node $w \in S, w \neq u, w \neq w_1, w \neq x_i, 1 \leq i \leq h$. Let $\Delta = D(p,w_1) - D(p,x_1)$. Consider a point p' which is $\Delta/4$ away from p toward x_1. Then $D(p',u) < D(p,x_1) < D(p,w)$, where $w \in S, w \neq u, w \neq x_1$. Then by Lemma 2, x_1 is a neighbor of u on $DT(S)$.

Case B. Suppose that $D(p,u) = D(p,x_1) = ... = D(p,x_h) < D(p,w_1) \leq D(p,w)$ for any other node $w \in S, w \neq u, w \neq w_1, w \neq x_i, 1 \leq i \leq h$. Let $\Delta = D(p,w_1) - D(p,x_1)$. Consider a point p' which is $\Delta/4$ away from p toward u. Then $D(p',u) < D(p,x_i) < D(p,w), 1 \leq i \leq h$, where $w \in S, w \neq u, w \neq x_i, 1 \leq i \leq h$. Let x' be x_i with smallest $D(p,x_i), 1 \leq i \leq h$. Then x' is a neighbor of u on $DT(S)$, similarly to in the cases A-1 or A-2.

Case C. Suppose that $D(p,x_1), ..., D(p,x_h) < D(p,u) \leq D(p,w)$ for any other node $w \in S, w \neq u, w \neq x_i, 1 \leq i \leq h$. Consider a point p' which moves from p toward u. Since $D(p',u)$ decreases the fastest, $D(p',u) \leq D(p',w)$ for any other nodes $w \in S, w \neq u, w \neq x_i, 1 \leq i \leq h$ is preserved. Moreover, there must be a point where $D(p',x') < D(p',u) \leq D(p',w)$ for any other node $w \in S, w \neq u, w \neq x'$, where x' is one of $x_i, 1 \leq i \leq h$. Then x' is a neighbor of u on $DT(S)$, similarly to in the cases A-1 or A-2. □

Theorem 4 (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 $DT(S)$.*

Proof. (only if) Suppose that C_u does not include a node v that is a neighbor node of u on $DT(S)$. Clearly, N_u cannot include v and the distributed Delaunay triangulation is not correct.

(if) Suppose that for every $u \in S$, C_u includes all the neighbor nodes of u on $DT(S)$. We show that $v \in S$ is a neighbor of u on $DT(C_u)$ if and only if v is a neighbor of u on $DT(S)$. i) Consider a neighbor v of u on $DT(S)$. Since $C_u \subset S$, by Lemma 3, v is a neighbor of u on $DT(C_u)$. ii) Consider a neighbor v of u on $DT(C_u)$. By Lemma 4, v is a neighbor of u of $DT(S)$. □

Theorem 4 identifies a necessary and sufficient condition for a distributed DT is correct, namely: the candidate set of each node must contain all of its global neighbors. 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, 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 by Theorem 1. 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 given in Figure 5.

```
Join(v) of node u
; Input: u is the joining node, if u is the only node in the system, v = NULL;
otherwise v is the closest existing node to u.
C_u ← {u}
N_u ← ∅
if v ≠ NULL then
  Send(v, NEIGHBOR_SET_REQUEST)
end if

On u's receiving NEIGHBOR_SET_REQUEST from w
if w ∉ C_u then
  C_u ← C_u ∪ {w}
  Update_Neighbors(C_u, N_u)
end if
N_w^u ← {x | x is a neighbor of w on DT(C_u)}
Send(w, NEIGHBOR_SET_REPLY(N_w^u))

On u's receiving NEIGHBOR_SET_REPLY(N_u^w) from w
C_u ← C_u ∪ N_u^w
Update_Neighbors(C_u, N_u)

Update_Neighbors(C_u, N_u) of node u
N_u^old ← N_u
N_u ← neighbor nodes of u on DT(C_u)
N_u^new ← N_u − N_u^old
for all v ∈ N_u^new do
  Send(v, NEIGHBOR_SET_REQUEST)
end for
```

Figure 5: Join protocol at a node u

Theorem 5 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 (Lemma 5), 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 (Lemma 8), and iii) the neighbor nodes of the joining node are also notified of the joining node's addition in the process. Note that this proof is based on Theorem 4, which determines the condition when a distributed DT is correct.

Lemma 5. *Let $S' = S \bigcup \{n\}$ and u be the closest node to n in S. Then u is a neighbor of n on $DT(S')$.*

Proof. Consider n, which is in $VC_{S'}(n)$. $D(n,n) < D(n,u) < D(n,w)$, for any other node $w \in S', w \neq$

$n, w \neq u$. Therefore, by Lemma 2, u is a neighbor of n on $DT(S')$. □

Lemma 6. *Let u' and v' be two points on an n-dimensional convex polytope P_n, $n \geq 2$. Then there exists a path from u' to v' on the surface of P_n, which goes through adjacent facets of P_n.*

Proof. On 2-dimensional convex polygon, there exists a path from u' to v' which goes along the edges of the polygon. Suppose that the lemma holds for k-dimensional convex polytope, $k \geq 2$, and consider two points u' and v' on an $(k+1)$-dimensional convex polytope P_{k+1}. Consider a cross section of the P_{k+1} which contains u' and v'. The cross section is an k-dimensional convex polytope and there exists a path from u' to v' on the surface of the polytope, which goes through adjacent facets. Note that each facet of the cross section is a part of a corresponding facet of P_{k+1}, and two adjacent facets of the cross section correspond to adjacent facets of P_{k+1}. Therefore the path is also on the surface of P_{k+1}, going through its adjacent facets. By induction, the lemma holds for n-dimensional polytopes, $n \geq 2$. □

Lemma 7. *Let $S' = S \bigcup \{n\}$. If $u \in S$ and $v \in S$ are two neighbor nodes of n on $DT(S')$ and $VF_{S'}(n,u)$ and $VF_{S'}(n,v)$ are adjacent, u and v are neighbors on $DT(S)$.*

Proof. Let p' be a point where $VF_{S'}(n,u)$ and $VF_{S'}(n,v)$ meet. That is, $D(p',n) = D(p',u) = D(p',v)$. Let $w_1 \in S$ be the closest node to p' except for n, u and v, and $\Delta = D(p',w_1) - D(p',n)$. Consider a point p which is $\frac{\Delta}{4}$ away from p' toward u. Then $D(p,u) < D(p,v) < D(p,w)$, for any other node $w \in S, w \neq u, w \neq v$. By Lemma 2, v is a neighbor node of u on $DT(S)$. □

Lemma 8. *Let $S' = S \bigcup \{n\}$ and u be a neighbor node of n of $DT(S')$. Then for any neighbor v of n on on $DT(S')$, there exists a series of nodes $< p_0 = u, ..., p_k = v >$, where $p_i, 0 \leq i \leq k$, is a neighbor of n on $DT(S')$, and p_i and p_{i+1}, $0 \leq i \leq k-1$, are neighbors on $DT(S)$.*

Proof. Note that $VC_{S'}(n)$ is a convex polytope enclosed by facets and each facet corresponds to a neighbor node of n. Also note that two neighbors of n are defined adjacent when their corresponding facets are adjacent on $VC_{S'}(n)$. Let u' be a point on the facet which corresponds to u and v' be a point on the facet which corresponds to v. By Lemma 6, there exists a path from u' to v' which goes through adjacent facets of $VC_{S'}(n)$. That is, there exists a series of nodes $< p_0 = u, ..., p_k = v >$, where $p_i, 0 \leq i \leq k$, is a neighbor of n on $DT(S')$ and $VF_{S'}(n,p_i)$ and $VF_{S'}(n,p_{i+1})$, $0 \leq i \leq k-1$, are adjacent. And by Lemma 7, p_i and p_{i+1} are neighbors on $DT(S)$. □

Lemma 9. *Let n denote a newly joining node, S be the set of existing node, and $S' = S \bigcup \{n\}$. Suppose that the existing distributed Delaunay triangulation for S is correct. Then when the join protocol finishes, C_n includes all the neighbor nodes of n on $DT(S')$.*

Proof. By Lemma 5, u is a neighbor node of n on $DT(S')$. For any neighbor node v of n on $DT(S')$, by Lemma 8, there exists a series of nodes $< p_0 = u, ..., p_k = v >$, where $p_i, 0 \leq i \leq k$, are neighbors of n on $DT(S')$ and p_i and p_{i+1}, $0 \leq i \leq k-1$, are neighbors on $DT(S)$. First u receives NEIGHBOR_SET_REQUEST from n. Since p_1 is a neighbor of u on $DT(S)$, C_u includes p_1, and p_1 is also included in N_n^u by Lemma 3 since p_1 is also a neighbor of n on $DT(S')$. After n receives N_n^u from u, C_n includes p_1.

Suppose that p_i is included in C_n. By Lemma 3, p_i is also included in N_n, and n sends NEIGHBOR_SET_REQUEST to p_i. Since p_{i+1} is a neighbor of p_i on $DT(S)$, C_{p_i} includes p_{i+1}, and p_{i+1} is also included $N_n^{p_i}$ by Lemma 3, since p_{i+1} is also a neighbor of n on $DT(S')$. After n receives $N_n^{p_i}$ from p_i, C_n includes p_{i+1}.

Therefore, within k iterations, C_n will include $p_k = v$. □

Theorem 5. *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.*

Proof. Lemma 9 shows that when the join process finishes, C_n will include all of its neighbor nodes on $DT(S')$. Also, whenever n discovers a neighbor node v of itself during the process, n sends NEIGHBOR_SET_REQUEST to v so that v includes n into C_v. Therefore the candidate sets of all nodes are properly updated, and the updated distributed Delaunay triangulation 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 section 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 section 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 given in Figure 6.

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.

Lemma 10. *Let $S' = S - \{u\}$. Let v be a neighbor node of u on $DT(S)$. If w is a neighbor node of v on $DT(S')$, then w is a neighbor node of u on $DT(S)$ or w is a neighbor node of v on $DT(S)$.*

Proof. Since w is a neighbor of v on $DT(S')$, by Lemma 1, there exists a point p such that $D(p, v) <$

```
Leave() of node u
Calculate DT(N_u) ; Note: u ∉ N_u
for all v ∈ N_u do
  N_v^u ← {w | w is a neighbor of v on DT(N_u)}
  Send(v, LEAVE(N_v^u))
end for

On u's receiving LEAVE(N_u^v) from v
C_u ← C_u − {v} ∪ N_u^v
N_u ← neighbor nodes of u on DT(C_u)
for all w ∈ N_u do
  if w satisfies conditions C1 and C2 from v then
    Send(w, DELETE(v))
  end if
end for

On u's receiving DELETE(w) from v
C_u ← C_u − {w}
for all x ∈ N_u do
  if x satisfies conditions C1 and C2 from w then
    Send(x, DELETE(w))
  end if
end for
```

Figure 6: Leave protocol at a node u

$D(p,w) < D(p,x)$, for any other node $x \in S', x \neq v, x \neq w$.

Case A) $D(p,w) < D(p,u)$. Since $S = S' \cup \{u\}$, $D(p,v) < D(p,w) < D(p,x)$, for any other node $x \in S$. Then by Lemma 2, v and w are neighbors on $DT(S)$.

case B) $D(p,u) \leq D(p,w)$. Then consider a point p' which moves from p toward w. Since $D(p',w)$ decreases faster than $D(p',x)$, $x \in S, x \neq u, x \neq v, x \neq w$, as p' moves and $D(p',u) \geq 0$ and $D(p',v) \geq 0$, there must be a point where $D(p',v) < D(p',w) < D(p',u) < D(p',x)$ or $D(p',u) < D(p',w) < D(p',v) < D(p',x)$, for any other node x. Then by Lemma 2, v and w are neighbors on $DT(S)$ or u and w are neighbors on $DT(S)$. □

Theorem 6. *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.*

Proof. Let $S' = S - \{u\}$. Consider a node $v \in S'$. First, u is removed from C_v by propagation of LEAVE and DELETE messages. Therefore $C_v \subset S'$.

Case A) Suppose that v is not a neighbor of u on $DT(S)$. Consider a node $w \in S', w \neq v$. If w is a neighbor of v on $DT(S')$, w is also a neighbor of v on $DT(S)$ by Lemma 3. If w is a neighbor of v on $DT(S)$, w is also a neighbor of v on $DT(S')$ by Lemma 4. Therefore the neighbors of v on $DT(S)$ are the same as the neighbors of v on $DT(S')$ and v is not affected by leave of u.

Case B) Suppose that v is a neighbor of u on $DT(S)$. Consider a node $w \in S', w \neq v$. If w is a neighbor of v on $DT(S')$, by Lemma 10, either w is already in C_v or v is notified of w by u. Therefore C_v will include all the neighbor nodes of v on $DT(S')$. □

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 the next subsection. 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 application. In the case of a leave, 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 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 given in Figure 7.

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.

We found that our maintenance protocol converged to a correct distributed DT some time after churn and failure have stopped in every one of our experiments. However 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.

On u's expiration of $PERIOD\_TIMER$
for all $v \in N_u$ **do**
  Send(v, NEIGHBOR_SET_REQUEST)
  Set $TIMEOUT\_TIMER_v$ as $T + TO$
  ; T is current time. TO is the timeout value.
end for
Set $PERIOD\_TIMER$ as $T + P$
; T is current time. P is the period of maintenance protocol.

On u's expiration of $TIMEOUT\_TIMER_v$
$C_u \leftarrow C_u - \{v\}$
Update_Neighbors(C_u, N_u)
for all $w \in N_u$ **do**
  Send(w, DELETE(v))
end for

On u's receiving NEIGHBOR_SET_REQUEST from v
if $v \notin C_u$ **then**
  $C_u \leftarrow C_u \cup \{v\}$
  Update_Neighbors(C_u, N_u)
end if
$N_v^u \leftarrow \{w \mid w$ is a neighbor of v on $DT(C_u)\}$
Send(v, NEIGHBOR_SET_REPLY(N_v^u))

On u's receiving NEIGHBOR_SET_REPLY(N_u^v) from v
$C_u \leftarrow C_u \cup N_u^v$
Update_Neighbors(C_u, N_u)

Figure 7: Maintenance protocol at a node u. Update_Neighbors(C_u, N_u) is the same as the one specified in Figure 5.

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 joing 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 neighbor 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 Distributed triangulation are undirectional and thus are counted twice to be compared

with neighbor entries. The accuracy of DDT_S is defined as follows:

$$\text{accuracy}(DDT_S) = \frac{N_{correct}(DDT_S) - N_{wrong}(DDT_S)}{2 \times N(DT(S))}.$$

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. Figure 8 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 is faster in a higher dimension space, since nodes have more neighbors and information is exchanged faster.

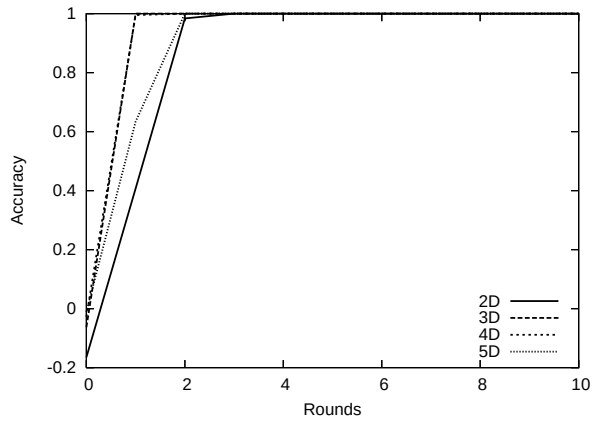


Figure 8: Accuracy of the maintenance protocol in a "ring" scenario, where each node initially knows only one other node. The number of nodes is 200.

3.6 Timestamp

In a dynamic environment, it is possible that a node receives messages with conflicting information. For example, some nodes may keep in their candidate set a node that has already left or failed, and disperse the information later. Then other nodes may consider the node as new, adding it to their candidate sets. Although the maintenance protocol will later detect that the node does not exist and delete it, such wrong information may again be forwarded to other nodes before the detection. In this way, wrong information may linger in the system unless the wrong information is detected and discarded as soon as it is received.

To address this issue, we introduce a clock T_u at each node u. The clock value is incremented whenever u sends out a message. Any information regarding a node is timestamped with its clock. In addition, a node u maintains the latest timestamp T_v^u of the information regarding another node v it knows of. When a node u receives any information regarding a node v from another node w, the timestamp of the information T_v^w is compared with the latest timestamp T_v^u at node u. If $T_v^w < T_v^u$ (that is, the received information is older), the information is discarded; otherwise it is accepted and the latest timestamp is updated ($T_v^u = T_v^w$).

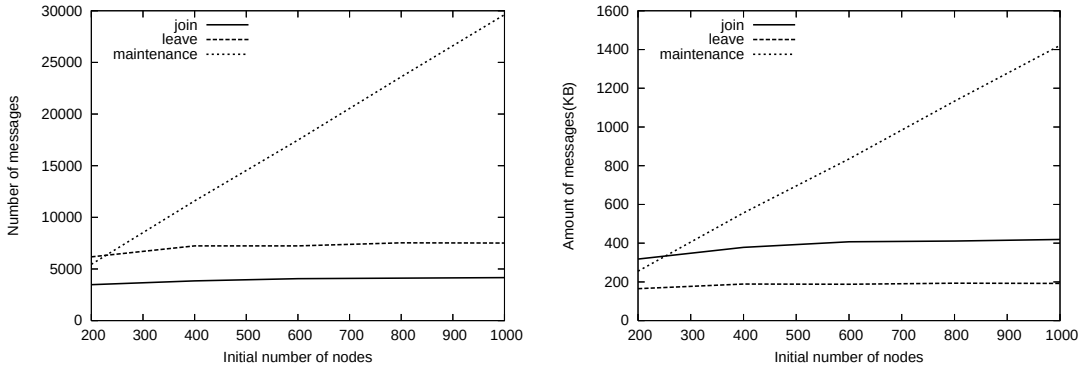
4 Experimental results

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 number of neighbors of a node on a DT is independent of the number of nodes in the system, 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. When the network size is smaller, the fraction of boundary nodes is larger, making the average number of neighbors smaller. Figure 9(a) and Figure 9(b) show the number of messages and the amount of messages (in Kbytes) versus system size in 3D. The join and leave curves represent the costs of 100 joins and 100 leaves respectively, and are more or less independent of system size showing very good scalability. The per-round cost of the maintenance protocol *for all nodes* increases linearly with system size; thus the average cost per node is constant versus system size.

4.2 Performance under churn

Figure 10(a) 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 sytem has 200, 400, or 800 nodes with a correct distributed DT. Then 1 node joins and 1 node gracefully leaves once every second on the average until time 110 second, using our join and leave protocols. Our maintenance



(a) The number of messages.

(b) The amount of messages.

Figure 9: The communication cost of protocols versus number of nodes in 3D. The join and leave curves represent the total costs of 100 serial joins and 100 serial leaves, respectively. The maintenance curve represents the per-round cost for all nodes to run the maintenance protocol.

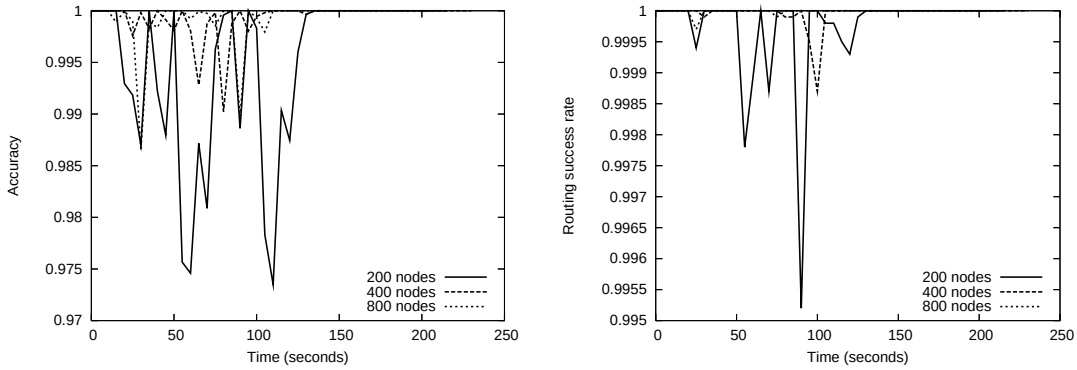
protocol is run once every 10 seconds.[5] In spite of the churn, accuracy of the distributed DT remains very high. The small error is due to concurrent joins and leaves, and is repaired by our maintenance protocol periodically. Figure 10(b) shows the success rate of a greedy routing protocol for a system 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 11(a) and Figure 11(b) show accuracy of distributed DT and greedy routing success rate for a system in which nodes join and fail concurrently. Except for nodes failing instead of leaving, the simulation parameters are the same as in the previous set of system churn experiments. Initially, the system has 200, 400, or 800 nodes with a correct distributed DT. Then, 1 node joins and 1 node fails once every second on the average until time 110 second, and our maintenance protocol is run once every 10 seconds.

Both accuracy and greedy routing success rate are much worse than in the previous case of system

[5] 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[16]. Note that accuracy in Figure 10(a) improves as the system size increases.



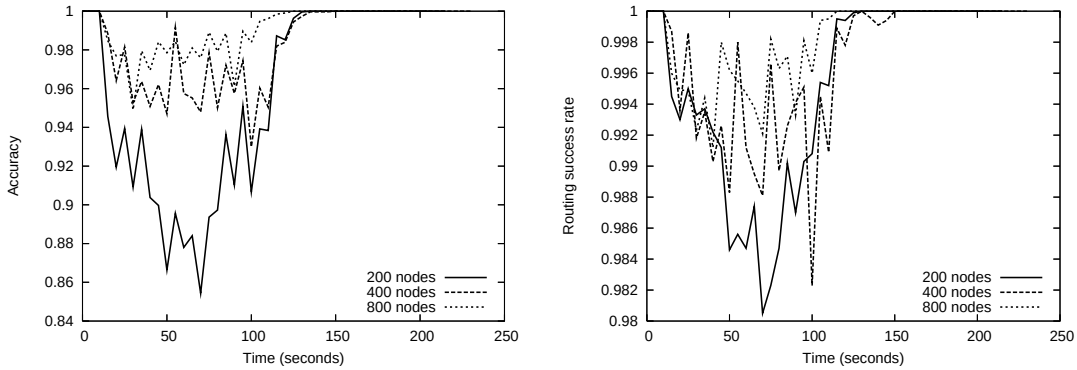
(a) Accuracy.

(b) Greedy routing success rate.

Figure 10: Performance of a system in 3D under churn versus time. 1 node joins and 1 node leaves per second on the average until 110 second. The maintenance protocol is run every 10 seconds.

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.



(a) Accuracy.

(b) Greedy routing success rate.

Figure 11: Performance of a system in 3D with concurrent joins and failures versus time. 1 node joins and 1 node fails per second on the average until 110 second. The maintenance protocol is run every 10 seconds.

5 Related work

The first protocol to construct DT was proposed by Liebeherr and Nahas [7]. The protocol utilizes the locally equiangular property of DT in 2D space. Nodes are assumed to have pre-assigned logical coordinates in 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 DT is guaranteed to succeed, a multicast tree can be implicitly determined for a given source using reverse path.

Steiner and Biersack [8] proposed a distributed approach to construct DT in 3D space. In their work, 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. [9] proposed a similar approach in d-dimensional space. They also addressed the case of node departures as well as arrivals. 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.

While DT has been extensively studied in computational geometry, most work in the field focuses on centralized algorithms. Ohnishi et al. [10] proposed an incremental algorithm to construct a distributed DT in 2D space. Yoo et al. [11] proposed a distributed algorithm to maintain DT for moving nodes in 2D space.

Locating the closest node to a given point is a common problem in many applications. Wong et al. [17] proposed a solution called Meridian, which uses multi-resolution rings. While Meridian is efficient since it requires $O(\log N)$ steps, where N is number of nodes in the system, it does not guarantee to find the closest node.

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 some application-level protocols to support DT applications.

We define a distributed DT and present a necessary and sufficient condition for a distributed DT to

be correct. This condition was used as a guide to design our join, leave, and maintenance protocols. Our join and leave protocols are proved correct for serial joins and leaves. For a system under churn and with node failures, we define an accuracy metric for a distributed DT. Experimental results show that our protocols are scalable and they achieve high accuracy for systems under churn and with node failures. In every one of our experiments conducted to date, the system converged to a correct distributed DT some time after churn and failures stopped. Typically convergence was achieved after running the maintenance protocol for a few rounds.

To support applications of distributed DT, we present application-level protocols for greedy routing, network node clustering, broadcast, and multicast within a radius. Each node in our greedy routing, broadcast and multicast protocols does not maintain any per-session state. We also discuss and prove correctness for the application protocols.

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.

[4] P.L. Chew, There is a planar graph as good as the complete graph, in Proceedings of the 2nd Symposium on Computational Geometry, pp. 169–177, 1986.

[5] J.M. Keil and C.A. Gutwin, The Delaunay triangulation closely approximates the complete Euclidean graph, in Proceedings of the 1st Workshop Algorithms Data Structure (LNCS 382), 1989.

[6] 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.

[7] Jorg Liebeherr, Michael Nahas, Application-Layer Multicasting With Delaunay triangulation Overlays, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 20, NO. 8, OCTOBER 2002.

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

[9] Gwendal Simon, Moritz Steiner, Ernst Biersack, Distributed Dynamic Delaunay Triangulation in d-Dimensional Spaces, submitted for publication, 2005.

[10] 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.

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

[12] 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.

[13] 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.

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

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

[16] 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.

[17] Bernard Wong, Aleksandrs Slivkins, Emin Gun Sirer, Meridian: A Lightweight Network Location Service without Virtual Coordinates, Proceedings ACM SIGCOMM, 2005.

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

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