The Dissertation Committee for Dong-Young Lee
certifies that this is the approved version of the following dissertation:

Protocol Design for Dynamic Delaunay Triangulation

Committee:

Simon S. Lam, Supervisor

Vijay Garg

Mohamed G. Gouda

Lili Qiu

Yin Zhang

Protocol Design for Dynamic Delaunay Triangulation

by

Dong-Young Lee, B.S.; M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2008

Dedicated to my wife and Mother.

Acknowledgments

First of all, I cannot thank enough my advisor Prof. Simon S. Lam for guiding me through the Ph.D. program. I have learned a lot from his wisdom as well as attitude in doing research and writing papers. I also owe thanks to the other committee members, Profs. Mohamed G. Gouda, Vijay K. Garg, Yin Zhang, and Lili Qiu. Not only their comments and encouragement, but also what I learned in their classes were crucial to build and develop my dissertation. I would like to express gratitude to Prof. Heon Y. Yeom and Dr. Kyung-Oh Lee, who is now a professor in Sunmoon University, for guidance of my first research when I was in the Master's program in Seoul National University.

Having wonderful colleagues is as fortunate as having great professors. Working with my colleagues in NRL was always enjoyable and discussions with them were inspiring. Yang Richard Yang led the first research project that I participated in UT. Collaborating with Min Sik Kim was an important part of my training. I especially thank Xincheng Brian Zhang for helping me finding my dissertation topic and encouraging my research. Huaiyu Liu and Yi Li were also good friends.

Everybody owes thanks to his or her parents, but I owe special thanks to mine. Both of them being professors, I was able to grow up in an academic

atmosphere. My father passed away when I was very young, but he has affected my entire life through his friends and students. My mother has faith in education and classic values, which has founded who I am. Her support and love are the sources of my self-confidence.

I was very lucky to have great friends. Bong-Soo Sohn was a great roommate and helped me a lot from my first day in Austin. I miss the days with Jaehyuk Huh, Young-ri Choi, Eunjin Jung, and Changkyu Kim. I also enjoyed conversation with Yongseok Cheon, Chan-gun Lee, and Honguk Woo.

I was extremely fortunate to meet my wife Jin Eun Yoo and have our little Austin. She has always supported me and he has been the source of happiness.

My study and research have been supported by scholarship from Korea Foundation for Advanced Studies, MCD fellowship, and NSF grants ANI-0319168, CNS-0434515, and CNS-0830939.

Protocol Design for Dynamic Delaunay Triangulation

Publication No. _____

Dong-Young Lee, Ph.D.
The University of Texas at Austin, 2008

Supervisor: Simon S. Lam

Delaunay triangulation (DT) is a useful geometric structure for networking applications. 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 investigate the design of join, leave, failure, and maintenance protocols for a set of nodes in d-dimension ($d > 1$) to construct and maintain a distributed DT in a dynamic environment. The join, leave, and failure protocols in the suite are proved to be correct for a single join, leave, and failure, respectively. For a system under churn, it is impossible to maintain a correct distributed DT continually. We define an accuracy metric such that accuracy is 100% if and only if the distributed DT is correct. The suite also includes a maintenance protocol designed to recover from incorrect system states and to improve accuracy. In designing the protocols, we make use of two novel

observations to substantially improve protocol efficiency. First, in the neighbor discovery process of a node, many replies to the node's queries contain redundant information. Second, the use of a new failure protocol that employs a *proactive* approach to recovery is better than the reactive approaches used in prior work. Experimental results show that our new suite of protocols maintains high accuracy for systems under churn and each system converges to 100% accuracy after churning stopped. They are much more efficient than protocols in prior work.

To illustrate the usefulness of distributed DT for networking applications, we also present several application protocols including greedy routing, finding a closest existing node, clustering, broadcast, and geocast. Bose and Morin proved in 2004 that greedy routing always succeeds to find the destination node on a DT. We prove that greedy routing always finds a closest existing node to a given point, and our broadcast and geocast protocols always deliver a message to every target node. Our broadcast and geocast protocols are also efficient in the sense that very few target nodes receive duplicate messages, and non-target nodes receive no message. Performance characteristics of greedy routing, broadcast, and geocast are investigated using simulation experiments. We also investigate the impact of inaccurate coordinates on the performance of greedy routing, broadcast, and geocast.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Delaunay triangulation [6] and Voronoi diagram [40] have a long history and many applications in different fields of science and engineering including networking applications, such as greedy routing, finding a closest node to a given point, broadcast, geocast, etc.[1, 20, 23, 27]. A triangulation for a given set S of nodes in a 2D space is a subdivision of the convex hull of nodes in S into non-overlapping triangles such that the vertexes of each triangle are nodes in S. A Delaunay triangulation in 2D is usually defined as a triangulation such that the circumcircle of each triangle does not include any other node inside the circumcircle. Delaunay triangulation can be similarly generalized to a d-dimensional space[1] ($d > 1$) using simplexes instead of triangles [11]. We will use DT as abbreviation for "Delaunay triangulation."

An important property of DT in the networking context is that greedy routing always succeeds on a DT [1]. 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

[1]Delaunay triangulation is defined in a Euclidean space. When we say a d-dimensional space in this dissertation, we mean a d-dimensional Euclidean space.

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, as is shown by experimental results in Section 3.7, since the length of an optimal route between a pair of nodes on a DT is within a constant time of the direct distance [3, 7, 13].

Another 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 a source node, which will be useful for many simulation-type applications such as multiplayer on-line games.

To demonstrate usefulness of a distributed DT for networking, we present several application protocols, including greedy routing, clustering, broadcast, and geocast. 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 a 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 [46] or a complete graph of clusters [45]. 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). Since greedy routing always succeeds on a DT, there exists a greedy-routing path from every node u to s. Therefore if GRPB forwards a message from s to all the reverse-paths of greedy-routing paths to s, the message will be delivered to every node u. 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 of the message.

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. Multicast to nodes within a target region is called *geocast* in the wireless networks literature [30]. 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. RadGRPM provides a special kind of geocast, which we call *radius geocast*, because the target nodes are all within a spherical region centered at the source. For distributed virtual environments, since an event usually affects nodes within a circle or sphere centered at an entity, RadGRPM is well suited for propagating events. For other potential applications, such as wireless networks and sensor networks, we will show that RadGRPM can be combined with unicast greedy routing from the source node to the center of a spherical region to provide a general geocast service.

We define a distributed DT and correctness for a distributed DT, and present a *necessary and sufficient condition* for a distributed DT to be correct. In our system model, each node maintains a set of nearby nodes, which is called its *candidate set*, and determines its set of neighbor nodes, which is called its *neighbor set*, based on its candidate set. We say that a distributed DT is *correct* when the neighbor set of each node is the same as the set of its neighbors on the global DT. Note that a candidate set is local information at a node. Our correctness condition identifies how much local information is required at each node to achieve a correct distributed DT.

This condition is used as a guide for designing a new suite of protocols for a set of nodes in a d-dimensional space ($d > 1$) to construct and maintain a distributed DT. In designing these protocols, we allow the set of nodes to

change with time. New nodes join the set (*system*) and existing nodes leave (gracefully) or fail[2] over time. The system is said to be *under churn* and the rate at which changes occur said to be the *churn rate*. In Chapter 4, we present a suite of four protocols for *join*, *leave*, *failure*, and *maintenance*.

In a distributed DT, each node maintains a set of its neighbors. By definition, a distributed DT of a set of nodes S is *correct* if and only if, for every node $u \in S$ on the distributed DT, u's neighbor set is the same as the set of u's neighbors on the (centralized) DT of S. For convenience, we will sometimes say "the system state is correct" to mean "the distributed DT is correct."

In designing the suite of protocols in this dissertation, we aim to achieve three properties: *accuracy*, *correctness*, and *efficiency*. The protocol suite is named ACE.

- Correctness – We prove the join, leave, and failure protocols to be correct for a single join, leave, and failure, respectively. For the join protocol, we prove that if the system state is correct before a new node joins, and no other node joins, leaves, or fails during the join protocol execution, then the system state is correct after join protocol execution. A similar correctness property is proved for the leave and failure protocols. Note that these three protocols are adequate for a system whose churn rate is so low that joins, leaves, and failures occur *serially*, i.e., protocol

[2]When a node fails, it becomes silent. We do not consider Byzantine failures.

execution finishes for each event (join, leave, or failure) before another event occurs. In general, for systems with a higher churn rate, we also provide a maintenance protocol, which is run periodically by each node.

- Accuracy – It is impossible to maintain a correct distributed DT continually for a system under churn. Note that correctness of a distributed DT is broken as soon as a join/leave/failure occurs and is recovered only after the join/leave/failure protocol finishes execution. Fortunately, some applications, such as greedy routing, can work well on a reasonably "accurate" distributed DT. We define an accuracy metric such that accuracy is 1 if and only if the distributed DT is correct. The maintenance protocol is designed to recover from incorrect system states due to concurrent protocol processing and to improve accuracy. We found that in all of our experiments conducted to date with the maintenance protocol, each system that had been under churn would converge to 100% accuracy some time after churning stopped.

- Efficiency – We use the total number of messages sent during protocol execution as the measure of efficiency. Protocols are said to be more efficient when their execution requires the use of fewer messages.

We previously presented three DT protocols in [23]: join and leave protocols that were proved correct and a maintenance protocol that was shown to converge to 100% accuracy after system churn. However, these protocols

(to be referred to as our *old* protocols) were designed with correctness as the main goal and their execution requires the use of a large number of messages.

To make the join and maintenance protocols in ACE much more efficient than our old ones, we have two novel observations. First, the objective of any join protocol is for a new node n to identify its neighbors (on the global DT), and for n's neighbors to detect n's join. In our old join protocol, n sends a request to an existing node u for n's neighbors in u's local information. When n receives a reply, it learns new neighbors and sends requests to those newly-learned neighbors. This process is recursively repeated until n does not find any more new neighbor. Whereas it is necessary to send messages to all neighbors, since the neighbors need to be notified that n has joined, *we discovered that to ensure correctness it is sufficient for n to hear back from just one neighbor in each simplex that includes*[3] *n rather than from all neighbors.* Furthermore, queries as well as replies for some simplexes can be combined so that just one query-reply between n and one neighbor is enough for multiple simplexes. Based on this observation, we designed a new join protocol for ACE. We found that the ACE join protocol is much more efficient than our old join protocol. We have proved the ACE join protocol to be correct for a single join.

We also apply the above observation to substantially reduce the number

[3]When we say a simplex includes a node, we mean that the set of vertexes of the simplex includes the node. Also, when we say a node is in a simplex, we mean that the node is a vertex of the simplex.

of messages used by the ACE maintenance protocol. Furthermore, we make a second observation (described below) to greatly reduce the total number of all protocol messages per unit time by reducing the frequency at which the ACE maintenance protocol runs.

In the old suite of protocols, it is the old maintenance protocol's job to detect node failures and repair the resulting distributed DT. To detect a node failure, the node was probed by all of its neighbors. Furthermore, the distributed DT was repaired in a reactive fashion. The process of reactively repairing a distributed DT after a failure is inevitably costly, because the information needed for the repair was at the failed node and lost after failure.

To improve overall efficiency, we added a new failure protocol to ACE specifically to handle node failures. The ACE failure protocol employs a *proactive* approach. Each node designates one of its neighbors as its *monitor node*. In the ACE failure protocol, a node is probed only by its monitor node, eliminating duplicate probes. In addition, each node prepares a *contingency plan* and gives the contingency plan to its monitor node. The contingency plan includes all information to correctly update the distributed DT after its failure. Once the failure of a node is detected by its monitor node, the monitor node initiates failure recovery. That is, each neighbor of the failed node is notified of the failure as well as any new neighbor(s) that it should have after the failure. In this way, node failures are handled almost as efficiently as graceful node leaves in the ACE leave protocol (which is the same as our old leave protocol). We have proved the ACE failure protocol to be correct for a single failure.

Each node runs the maintenance protocol (ACE or old) periodically. The communication cost of the maintenance protocol increases as the period decreases (or frequency increases). Generally, as the churn rate increases, the maintenance protocol needs to be run more frequently. In the old protocol suite, moreover, the old maintenance protocol needs to be run at the probing frequency because one of its functions is to recover from node failures. With the inclusion of an efficient failure protocol in the ACE protocol suite to handle failures separately, the ACE maintenance protocol can be run less often. We found that the overall efficiency of the ACE protocols is greatly improved as a result.

To the best of our knowledge, the only other previous work for a dynamic distributed DT in a d-dimensional space is by Simon *et al.* [36]. They proposed two sets of distributed algorithms: basic generalized algorithms and improved generalized algorithms. Each set consists of an entity insertion (node join) algorithm and an entity deletion (node failure) algorithm. Their basic entity insertion algorithm is similar to our old join protocol. Their improved entity insertion algorithm is based on a centralized flip algorithm [42] whereas our join protocols are based on a "candidate-set approach" and our correctness condition for a distributed DT. The two approaches are fundamentally different. Their entity deletion algorithm and our ACE failure protocols are also different. Our ACE failure protocol is substantially more efficient than their improved entity deletion algorithm, which uses a reactive approach and allows duplicate probes. The centralized flip algorithm is known to be correct [8].

Table 1.1: A comparison of our old and ACE protocols with Simon *et al.*'s basic and improved algorithms.

	efficiency	convergence to 100% accuracy after system churn
Simon *et al.*'s basic algorithms	medium	No
Simon *et al.*'s improved algorithms	high	No
Our old protocols	low	Yes
ACE protocols	very high	Yes

However, correctness of their distributed algorithms is not explicitly proved. Lastly, they do not have any algorithm, like our maintenance protocols, for recovery from concurrent processing of joins and failures due to system churn. As a result, their algorithms consistently failed to converge to 100% accuracy after system churn in simulation experiments.

A quick comparison of the four sets of protocols/algorithms is shown in Table 1.1. More detailed experimental results, presented in Section 7, show that ACE protocols are *an order of magnitude more efficient* than our old protocols. There is a tradeoff, however. During a churn period, the average accuracy of ACE protocols is slightly (a fraction of 1%) lower than the average accuracy of our old protocols.

DT is defined in a Euclidean space. That is, each node has its coordinates in the Euclidean space. Thus in our system model, we assume that each

node is located in a Euclidean space and knows its coordinates. Given such assumptions, DT has desirable properties such that application protocols run correctly and efficiently on a DT. For example, greedy routing always succeeds and has a short routing path. GRPB and RadGRPM deliver a message to all target nodes using minimal number of messages. However, the assumptions may not always be satisfied in practice. For example, a node may not accurately determine its coordinates. Then overall performance of applications may be affected by the inaccuracy of coordinates. We investigate impact of coordinate inaccuracy on quality of greedy routing, which is the basis of our application protocols, and correctness and efficiency of RadGRPM.

The organization of this dissertation is as follows. In Chapter 2, we introduce the concepts and definitions of a distributed DT, present our system model, and a correctness condition for a distributed DT. We present examples of application protocols and performance of the application protocols in Chapter 3. In Chapter 4, design of ACE protocol suite and its performance are presented. We also define the accuracy metric of a distributed DT in the same chapter. The impact of inaccurate coordinates on application protocols is investigated in Chapter 5. We conclude in Chapter 6.

Chapter 2

Distributed Delaunay triangulation

Consider a set of nodes. Conceptually, nodes are points in a Euclidean space. The results and protocols in this dissertation are for d-dimensional spaces, where $d > 1$. 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 circumhyperspheres of simplexes in higher dimensions), as we introduced DT at the beginning of the previous chapter. Since the DT properties of interest to us come from Voronoi diagrams, we believe that our approach is appropriate in the context of this dissertation. Then we define distributed DT and correctness for a distributed DT.

In our system model, the set of nodes that a node knows is referred to as its candidate set. Each node determines its neighbors based on its candidate set. Then we identify a necessary and sufficient condition of candidate sets for a distributed DT to be correct. This correctness condition is used as a guide to design our 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, which is denoted by $VF_S(u,v)$.*

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 2.1 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. $VC_S(u)$ 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. Note that facets of a Voronoi cell perpendicularly bisect edges of a DT. Therefore, a DT

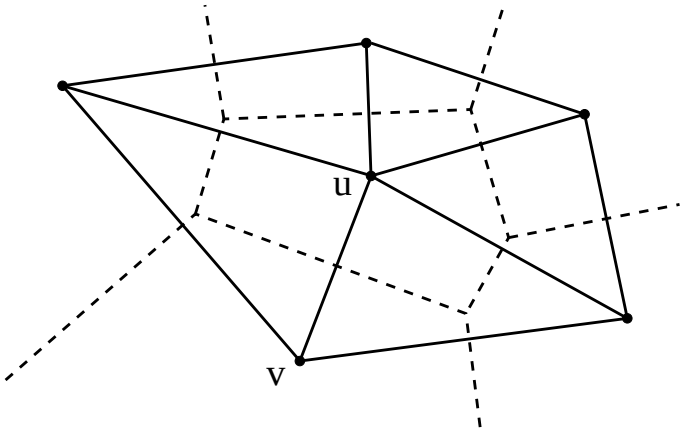

Figure 2.1: A Voronoi diagram (dashed lines) and the corresponding DT (solid lines) in a 2-dimensional space.

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 $\{< u, N_u > |\ u \in S\}$, 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,

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

it is not straightforward to achieve correctness.

2.2 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 [27, 32, 36, 37]. The mechanism to obtain coordinates is beyond the scope of this dissertation. Coordinates may be given by an application, a GPS device, or topology-aware virtual coordinates [31].[2] Also when we say a node u *knows* another node v, we assume that u knows v's coordinates as well. We also assume that nodes are in *general position*, namely: no $d+1$ nodes are on the same hyperplane and no $d+2$ nodes are on the same hypersphere [11].

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 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 $DT(C_u)$.

To simplify protocol descriptions, we assume that message delivery is reliable. In a real implementation, additional mechanisms such as ARQ may be used to ensure reliable message delivery.

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

2.3 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 has enough information 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.

Theorem 1 (Correctness Condition)**.** *Let S be a set of nodes and for each node $u \in S$, u knows C_u, such that $u \in C_u \subset S$. The distributed DT of S is correct if and only if, for every $u \in S$, C_u includes all neighbor nodes of u on $DT(S)$.*

Theorem 1, previously presented in [21, 23], identifies a *necessary and sufficient condition* for a distributed DT to be correct, namely: the candidate set of each node contains all of its global neighbors. We use the above correctness condition as a guide to design our protocols. A proof of Theorem 1 is presented below.

To prove Theorem 1, we first prove Lemmas 1 – 4.

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 all $w \in S, w \neq u, w \neq v$.*

Proof. (1) Consider a point p' on the shared facet of $VC_S(u)$ and $VC_S(v)$.

(2) $D(p',u) = D(p',v) < D(p',w)$ for all $w \in S, w \neq u, w \neq v$.

(3) 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.

(4) $D(p,u) < D(p,v) < D(p,w)$ for all $w \in S, w \neq u, w \neq v$.

□

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

Proof. (1) Consider a point p' that moves from p toward v.

(2) $D(p',v)$ decreases faster than, or as fast as, $D(p',w)$ for all $w \in S, w \neq u, w \neq v$.

(3) In case $D(p',v)$ decreases faster than $D(p',w)$, $D(p',v) < D(p',w)$ after p' moves from p toward v.

(4) In the other case where $D(p',w)$ decreases as fast as $D(p',v)$, w must be in the same direction as v from p. In that case, $D(p,v) < D(p,w)$. (For

p, v, and w that are on the same line, $D(p,v) = D(p,w)$ implies $v = w$.) Subsequently, $D(p',v) < D(p',w)$ holds as p' moves toward v.

(5) From (3) and (4), $D(p',v) < D(p',w)$ after p' moves from p toward v.

(6) As p' moves from p toward v, $D(p',v)$ will decrease toward 0 while $D(p',u) \geq 0$.

(7) There must be a point where $D(p',u) = D(p',v)$.

(8) From (5) and (7), $D(p',u) = D(p',v) < D(p',w)$ for all $w \in S, w \neq u, w \neq v$.

(9) Let w_1 be the third closest node from p' in S and let $\Delta = D(p',w_1) - D(p',v)$. Consider the hyperplane F that includes p' and is perpendicular to the edge $\overline{uv}$. For all p'' that is on F and $D(p',p'') < \frac{\Delta}{4}$, $D(p'',u) = D(p'',v) < D(p'',w)$ for all $w \in S, w \neq u, w \neq v$.

(10) $VC_S(u)$ and $VC_S(v)$ share a facet that includes p' in (8) and p'' in (9).

(11) From (10), u and v are neighbors on $DT(S)$.

□

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

Proof. (1) Since v is a neighbor of u on $DT(S)$, by Lemma 1, there exists a point p where $D(p,u) < D(p,v) < D(p,w)$ for all $w \in S, w \neq u, w \neq v$.

(2) Since $C \subset S$, $D(p,u) < D(p,v) < D(p,w)$ for all $w \in C, w \neq u, w \neq v$.

(3) By Lemma 2, v is a neighbor of u on $DT(C)$.

□

Lemma 4. *Let S be a set of nodes, $u \in S$ and $u \in C_u \subset S$. Assume that C_u includes all 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. Our proof is by contradiction.

(1) $v \in C_u$ is a neighbor of u on $DT(C_u)$.

(2) Suppose that v is not a neighbor of u on $DT(S)$.

(3) From (1) and Lemma 1, there exists a point p in $VC_{C_u}(u)$ such that $D(p,u) < D(p,v) < D(p,w)$ for all $w \in C_u, w \neq u, w \neq v$.

(4) From (2), (3), and Lemma 2, there exists at least one node $x \in S, x \notin C_u, x \neq u, x \neq v$ that satisfies $D(p,x) < D(p,v)$.

(5) Let $x_1, ..., x_k, k \geq 1$ be the nodes each of which satisfies the condition in (4). Without loss of generality, let $D(p,x_1) \leq D(p,x_2) \leq ... \leq D(p,x_k)$.

(6) From (3) – (5), we have $D(p,x_1) \leq D(p,x_2) \leq ... \leq D(p,x_k) < D(p,v) \leq D(p,w)$ for all $w \in S, w \neq u, w \neq v, w \neq x_i, 1 \leq i \leq k$.

(7) Consider a node $w \in S, w \neq u, w \neq v, w \neq x_i, 1 \leq i \leq k$. From (6), $D(p,v) \leq D(p,w)$. From (3), $D(p,u) < D(p,v)$. Thus, for all $w \in S, w \neq u, w \neq x_i, 1 \leq i \leq k$, $D(p,u) < D(p,w)$.

(8) We show below that in all possible cases, there exists a node $x_i, 1 \leq i \leq k$ that is a neighbor of u on $DT(S)$.

(9) Since $x_i \notin C_u$, it is contradictory to the assumption that C_u includes all neighbor nodes of u on $DT(S)$. Therefore v is a neighbor of u on $DT(S)$.

Justification of step (8) in above proof: Recall that, from (6) and (7), for all $w \in S, w \neq u, w \neq x_i, 1 \leq i \leq k$, $D(p,u) < D(p,w)$, and $D(p,x_i) < D(p,w)$ for $1 \leq i \leq k$.

Comparing $D(p,u)$ and $D(p,x_1)$, there can be three cases: $D(p,u) < D(p,x_1)$ (case A), $D(p,u) = D(p,x_1)$ (case B), and $D(p,x_1) < D(p,u)$ (case C).

Case A. $D(p,u) < D(p,x_1)$.

From (6) above, we have $D(p,u) < D(p,x_1) \leq D(p,w)$ for all $w \in S, w \neq u, w \neq x_1$. By Lemma 2, x_1 is a neighbor of u on $DT(S)$.

Case B. $D(p,u) = D(p,x_1)$.

Let h be the largest integer such that $D(p,x_1) = D(p,x_i), 1 \leq i \leq h \leq k$. Let w_1 be a node such that $w_1 \in S, w_1 \neq u, w_1 \neq x_i, 1 \leq i \leq h$, $D(p,w_1) \leq D(p,w)$ for all $w \in S, w \neq u, w \neq w_1, w \neq x_i, 1 \leq i \leq h$.

From (6), we have $D(p,u) = D(p,x_1) = ... = D(p,x_h) < D(p,w_1) \leq D(p,w)$ for all $w \in S, w \neq u, w \neq w_1, w \neq x_i, 1 \leq i \leq h$.

(1) Let $\Delta = D(p,w_1) - D(p,x_1)$. Consider a point p' that is $\frac{\Delta}{4}$ away from p toward u.

(2) Then $D(p',u) < D(p',x_i) < D(p',w), 1 \leq i \leq h$, for all $w \in S, w \neq u, w \neq x_i, 1 \leq i \leq h$.

(3) Let x' be x_i with smallest $D(p',x_i), 1 \leq i \leq h$. Then we have $D(p',u) < D(p',x') \leq D(p',w)$ for all $w \in S, w \neq u, w \neq x'$. This is case A with p' replacing p and x' replacing x_1, which has been proved.

Case C. $D(p,u) < D(p,x_1)$.

Let h be the largest integer such that $D(p,x_i) < D(p,u), 1 \leq i \leq h \leq k$. From (6), we have $D(p,x_i) < D(p,u) \leq D(p,w)$ for all $w \in S, w \neq u, w \neq x_i, 1 \leq i \leq h$.

(1) Consider a point p' that moves from p toward u.

(2) $D(p',u)$ decreases toward 0 faster than or as fast as $D(p',w)$ for all $w \in S, w \neq u$.

(3) We still have $D(p',u) \leq D(p',w)$ for all $w \in S, w \neq u, w \neq x_i, 1 \leq i \leq h$.

(4) $D(p',x_i) > 0$ for all $x_i, 1 \leq i \leq h$.

(5) There must be a point where for some $x' = x_i, 1 \leq i \leq h$, $D(p', u) = D(p', x') \leq D(p', w)$ for all $w \in S, w \neq u, w \neq x'$. This is case B with p' replacing p and x' replacing x_1, which has been proved.

□

Theorem 1 (Correctness Condition)**.** *Let S be a set of nodes and for each node $u \in S$, u knows C_u, such that $u \in C_u \subset S$. The distributed DT of S is correct if and only if, for every $u \in S$, C_u includes all neighbor nodes of u on $DT(S)$.*

Proof. Let N_u, $u \in S$ be the set of u's neighbor nodes on $DT(C_u)$.

(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 DT is not correct.

(if) Suppose that for every $u \in S$, C_u includes all 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 $DT(S)$.

(1) (if) Consider a neighbor v of u on $DT(S)$. Since $C_u \subset S$, by Lemma 3, v is a neighbor of u on $DT(C_u)$.

(2) (only if) Consider a neighbor v of u on $DT(C_u)$. By Lemma 4, v is a neighbor of u on $DT(S)$.

□

Chapter 3

Applications of distributed Delaunay triangulation

In this chapter 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 correct distributed DT. Our protocols to construct and maintain a distributed DT are deferred to Chapter 4. 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, coordinates of its neighbors, and the coordinates of any other node 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.

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 [1]. We prove a similar property that greedy routing can also find a 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 geocast, and prove correctness for the protocols.

3.1 Greedy routing

A well-known property of DT is that greedy routing always succeeds on a DT [1]. In greedy routing, a node forwards a message to a neighbor that is closest to the destination. 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 [3, 7, 13].

3.2 Finding a closest existing node

Similar to the previous application of greedy routing, a DT may be utilized in finding a closest existing node to a given point. (Note that the given point may not be a node in the DT.) Finding a closest existing node is a common operation in many Internet applications, including server selection, node clustering, and peer-to-peer overlay networks.[1]

Consider the problem of finding a 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.

[1] If topology-aware virtual coordinates (e.g. [31]) are used for Internet applications, application performance would be affected by accuracy of the virtual coordinates.

Let v_0 be s. At v_i, the greedy routing algorithm selects the next-hop node v_{i+1} that 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 to find a closest existing node to a given point as long as it is run on a DT. Bose and Morin [1] proved a similar theorem that greedy routing always succeeds to arrive at a given destination node on a DT. We use an approach similar to theirs to prove the following theorem.

Theorem 2. *Finding a closest node $d \in S$ to a given point $n \notin S$ using greedy routing always succeeds on $DT(S)$.*

Proof. We first show that every node $v \neq d$ on $DT(S)$ has a next-hop node in greedy routing toward n. (See Figure 3.1.)

(1) Suppose that $v \neq d$.

(2) Draw a straight line L from v to n, and let P be the hyperplane that includes the first Voronoi facet that L crosses.

(3) Let u be the node in the adjacent Voronoi cell that shares P with v.

(4) u and v are neighbors of each other on DT(S).

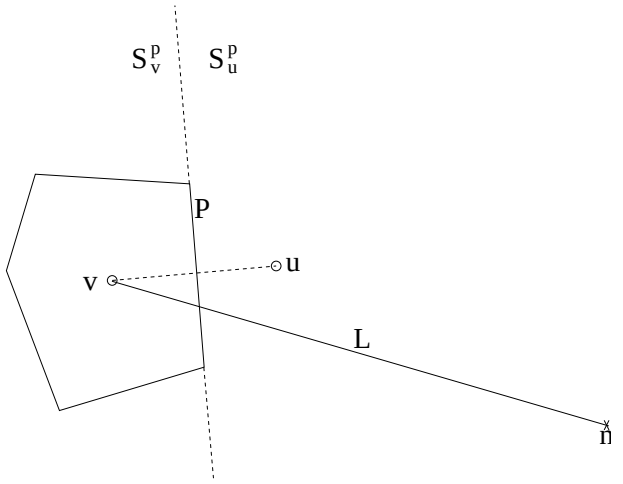

Figure 3.1: Figure for Theorem 2 proof.

(5) P divides the entire space into two regions S_u^p and S_v^p such that points in S_u^p are closer to u than to v.

(6) Since n belongs to S_u^p, n is closer to u than v.

(7) When $v \neq d$, v has a neighbor that is closer to n.

(8) On the other hand, if $v = d$, the routing stops at v.

(9) Since there are a finite number of nodes, eventually a closest node d is found in a finite number of steps.

□

3.3 Clustering of network nodes

To illustrate an application of finding a 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 [15].[2] 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 [15].

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

[2]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.

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 [46], clusters form a random graph and a joining node may fail to find the closest existing cluster. In [45], every node maintains links to every other cluster, limiting scalability. (The scalability issue is addressed in [45] by introducing a hierarchy of clusters.) Our protocol finds the closest cluster for a joining node and is scalable.

3.4 Broadcast using reverse greedy paths

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. Figure 3.2 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 a reverse-path broadcast from the source node s, u should forward a message to v, if u knows that u is the next hop of v in the forward route. Note that s is the destination in the forward

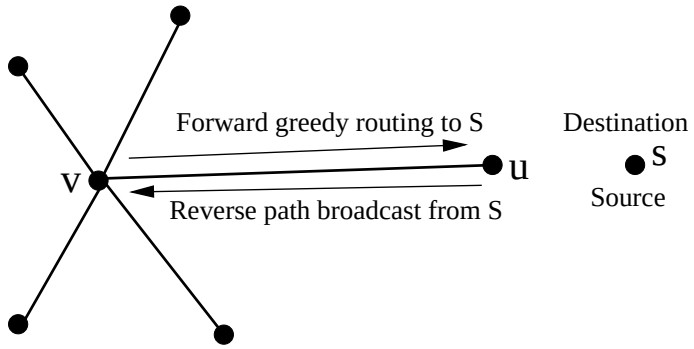

Figure 3.2: Forward path and reverse path.

greedy routing and the source in the reverse-path broadcast. 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 [5]. In the context of DT, HyperCast [26] 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 determine next-hop nodes based on the coordinates of its neighbors, itself, and the source node.

We name our broadcast protocol as GRPB (greedy reverse-path broadcast). In GRPB, a node u maintains *a local DT* of 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 of 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 3.3 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

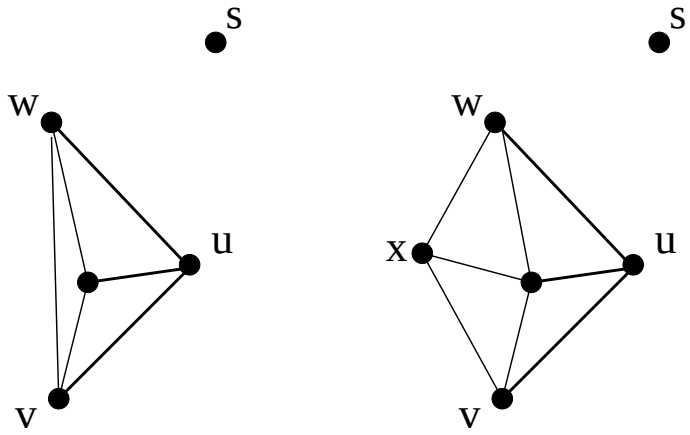

Figure 3.3: An ambiguous situation due to limited knowledge in GRPB.

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 the experiments, the number of duplicate messages was from 3% to 10% of the number of nodes. For further experiments on message duplication, refer to Section 5.2. The protocol pseudocode is presented in Figure 3.4.

The following theorem guarantees the correctness of GRPB, namely it delivers a message to all nodes in the system.

Theorem 3. *Let a set S of nodes 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.*

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

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

Figure 3.4: GRPB protocol at a node u.

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.

Our proof is by contradiction.

(1) Assume that the theorem is not true. Suppose a node u fails to forward to its neighbor v, while there exists an edge from u to v in the global reverse-path tree.

(2) v is a neighbor of u on the local DT of u, since the distributed DT is correct. [From Definition 5.]

(3) u is closer to s than v is, since there is an edge from u to v in the global reverse-path tree.

(4) On the local DT of u, there exists a node w that is a mutual neighbor of

u and v, and the distance between w and s is shorter than the distance between u and s. [From (1), (2), (3), and conditions C1 and C2.]

(5) w is not a neighbor of v on the global DT. [If w were a neighbor of v, the next hop of v in the forward path should not be u since, from (4) , w is closer than u to s.]

(6) On the local DT of u, there exists a simplex that includes u, v and w. Let the simplex be denoted by p. [From condition C2.2.]

(7) p does not exist on the global DT, since w is not a neighbor of v. [From (5).]

(8) On the global DT, the space of p is occupied by other simplexes.

(9) Let x be one such simplex that includes u and v. Let $x_1, ..., x_k$ be the other nodes of x other than u or v.

(10) $x_1, ..., x_k$ are neighbors of u on the global DT.

(11) $x_1, ..., x_k$ are neighbors of u on the local DT of u, since the distributed DT is correct. [From Definition 5.]

(12) There exists the same simplex x on the local DT of u, since v and $x_1, ..., x_k$ are neighbors of u. [From (2) and (11).]

(13) It is impossible that x and p co-exist on the local DT of u, since they overlap.

□

3.5 Radius geocast

Geocast is a special case of multicast in which a message is delivered to all nodes in a given region. Our radius geocast protocol, RadGRPM, is designed to deliver a message to all nodes within a given radius from a source node.

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 forward greedy routing. We utilize this observation in designing RadGRPM.

In RadGRPM 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.

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 (condition C3). Note that no node outside the given radius receives any message, which is one reason that RadGRPM is efficient. Pseudocode of the protocol is presented in Figure 3.5. Theorem 4 guarantees that RadGRPM delivers the message to all nodes within a given radius.

Theorem 4. *Let a set S of nodes form a correct distributed DT. The RadGRPM protocol delivers a message from a source node $s \in S$ to all nodes*

```
Start_radius_geocast(msg, rad) of node u
; u is a source node, loc is location of u
for all v ∈ N_u within rad from loc do
  send GEOCAST(msg, rad, loc) to v
end for

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

Figure 3.5: RadGRPM protocol at a node u.

within a radius r from s.

Proof. By Theorem 3, the GRPB protocol delivers a message to all 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 nodes along the reverse greedy paths after stopping have distances from s longer than r. Therefore the RadGRPM protocol delivers the message to all nodes within the radius r. □

3.6 General geocast

Note that RadGRPM by itself is a special kind of geocast in the sense that a source node is at the center of a spherical target region. RadGRPM can be combined with unicast greedy routing to provide general geocast. That

is, in case a source node is not at the center of a spherical target region, the source sends a unicast message to the center location using greedy routing; the message is then propagated within the target region using RadGRPM, as described below.

If no node exists at the center of a spherical target region, greedy routing towards the center (specified by its coordinates) will succeed to forward the unicast message to a node closest to the center (see Theorem 2 and its proof). For clarity of explanation, let us assume for now that there is only one node that is closest to the center point. The case where there are two or more closest nodes is addressed below. Let c denote the center point and c' the closest node to c. As soon as c' receives a unicast message by greedy routing towards c, c' determines that it is a closest node to c among its neighbors and starts RadGRPM. More specifically, it executes the Start_radius_geocast() function in Figure 3.5 using the center's location for parameter *loc* instead of its own location. Starting RadGRPM from c' does not affect correct execution of RadGRPM. Correctness of GRPB and RadGRPM is based on the fact that greedy routing to a node always succeeds on a DT. Since greedy routing towards c always succeeds to reach the closest node c', c' is the root of the reverse-path tree of greedy routing from every node towards c. Note that the same greedy routing towards c is used, whether a node exists at c or not. Therefore the same reverse-path conditions (C1 and C2) can be used even if RadGRPM is started from c'. Also, the distance from c monotonically increases in the reverse greedy paths, allowing use of the same stopping condition (C3).

It is possible that there are two or more nodes closest to the center. (These nodes are equidistant from the center.) Greedy routing towards the center will forward the unicast message to one of the closest nodes. Let c'_1 denote the closest node that receives the unicast message. Let $c'_2, ..., c'_k$ denote the other closest nodes, where $k > 1$. Note that the greedy paths from all nodes form a forest of trees, the root nodes of which are the closest nodes, $c'_i, 1 \leq i \leq k$. When the unicast message arrives at c'_1 by greedy routing, c'_1 determines that none of its neighbors is closer to the center than itself, which means c'_1 is one of the closest nodes, and it executes the Start_radius_geocast() function in Figure 3.5 using the center's location for paramter *loc* instead of its own. In the function, c'_1 sends a geocast message to each of its neighbors within the radius of the center. Therefore if another closest node is a neighbor of c'_1, it will receive the geocast message; it also determines that it is a closest node to the center and it executes the Start_radius_geocast() function in Figure 3.5 using the center's location for parameter *loc* instead of its own. Thus, if the set of closest nodes to the center and DT edges between them form a connected graph, then all of the other closest nodes are guaranteed to receive the geocast message. Subsequently, the geocast message will be propagated in all reverse-path trees of the forest and it will be delivered to all nodes within the radius of the center location. Pseudocode of the protocol is presented in Figure 3.6.

In the following lemma, we prove that the set of nodes closest to the center and DT edges between them form a connected graph, which is sufficient to prove correctness of our general geocast protocol described above.

```
Start_geocast(msg, c, rad) of node u
; u is a source node
Route_geocast(msg, c, rad)

Route_geocast(msg, c, rad) of node u
let v ∈ N_u be the closest node to c
if D(u,c) ≤ D(v,c) and D(u,c) ≤ rad then
  ; u is the closest node to c
  Deliver(msg)
  for all v ∈ N_u within rad from c do
    send GEOCAST(msg, rad, c) to v
  end for
else
  send GEOCAST(msg, rad, c) to v
end if

On u's receiving GEOCAST(msg, rad, c) from w
if D(u,c) > D(w,c) then
  ; RadGRPM phase
  Deliver(msg)
  for all v ∈ N_u do
    if v satisfies conditions C1, C2, and C3 from c then
      send GEOCAST(msg, rad, c) to v
    end if
  end for
else
  ; unicast phase
  Route_geocast(msg, c, rad)
end if
```

Figure 3.6: General geocast protocol at a node u.

Lemma 5. *Let a set S of nodes form a correct distributed DT. Let p denote a point in the space. Let $c'_1, c'_2, ..., c'_k$ denote the closest nodes to p in S, , $k > 1$. Then the subgraph of DT that includes $c'_1, c'_2, ..., c'_k$ and edges between them is*

a connected graph.

Proof. Our proof is by contradiction.

(1) Suppose the subgraph of DT is not a connected graph. Without loss of generality, suppose that $c'_1, c'_2, ..., c'_h$ are connected, $h < k$, but they are not connected to c'_k.

(2) Let $N_c = N_{c'_1} \cup N_{c'_2} \cup ... \cup N_{c'_h} - \{c'_1, c'_2, ..., c'_h\}$. Let n_c be the closest node in N_c to p. Let $\Delta = D(n_c, p) - D(c'_1, p)$.

(3) Let p' be a point that is $\frac{\Delta}{4}$ away from p towards c'_k.

(4) $D(c'_k, p') = D(c'_k, p) - \frac{\Delta}{4}$.

(5) $D(c'_i, p') > D(c'_i, p) - \frac{\Delta}{4} = D(c'_k, p'), i \neq k$. [$c'_i, i \neq k$ cannot be in the same direction as c'_k from p.]

(6) c'_k is the only closest node to p'.

(7) $D(c'_i, p') \leq D(c'_1, p) + \frac{\Delta}{4}, 1 \leq i \leq h$. [From (3) and the assumption that $D(c'_1, p) = D(c'_i, p), 1 \leq i \leq k$.]

(8) $D(c'_1, p) + \frac{\Delta}{4} \leq D(n'_c, p) - \Delta + \frac{\Delta}{4}, n'_c \in N_c$. [From (2).]

(9) $D(n'_c, p) - \Delta + \frac{\Delta}{4} < D(n'_c, p) - \frac{\Delta}{4}, n'_c \in N_c$.

(10) $D(n'_c, p) - \frac{\Delta}{4} \leq D(n'_c, p'), n'_c \in N_c$. [From (3).]

(11) $D(c'_i, p') < D(n'_c, p'), 1 \leq i \leq h, n'_c \in N_c$. [From (7) – (10).]

(12) Greedy routing from c'_1 towards p' will be stuck at one of the nodes $c'_1, ..., c'_h$, and cannot reach c'_k.

(13) Greedy routing from any node in S towards p' always succeeds to reach the closest node to p', which is c'_k. [From (6) and Theorem 2.]

(14) (12) and (13) are contradictory to each other.

□

3.7 Performance of application protocols

In this section, we evaluate performance of application protocols assuming that coordinates are accurate and the underlying distributed DT is correct. Impact of inaccurate coordinates on application performance is investigated in Chapter 5.

3.7.1 Performance of greedy routing

We first evaluate the performance of greedy routing, which is the basis of our application protocols. We use **relative path length** as the performance metric, which is defined as the ratio of the overall length of a greedy routing path to the direct distance between a source and a destination node (see Figure 3.7).

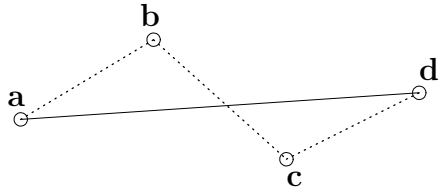

Figure 3.7: An illustration of relative path length. The dotted line represents the greedy routing path from a to d via b and c. Relative path length of this greedy routing path is the ratio of the overall length of the dotted line to the length of the solid line (direct distance between a and d).

Figure 3.8 shows the relative path length of greedy routing on a dis-

tributed DT using accurate coordinates. In an experiment, greedy routing paths between 1000 random pairs of nodes on a distributed DT of 1000 randomly placed nodes are evaluated and the average of the relative path lengths is calculated. The curve represents the average result of 100 such experiments for different dimensionalities. Each vertical bar indicates the range of results from 10th percentile to 90th percentile. It is known that the shortest path on a DT is within a constant time of direct distance [3, 7, 13]. Though greedy routing does not always find the shortest path, Figure 3.8 shows that it performs very well in practice. The average path length is around 1.2 times of direct distance.

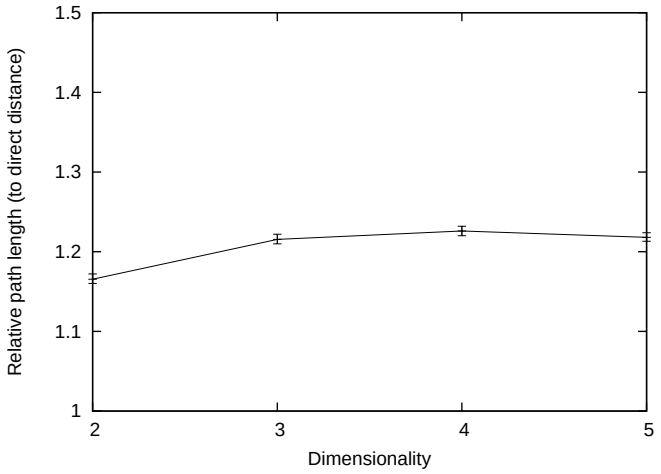

Figure 3.8: Ratio of greedy-routing path length to direct distance on a distributed DT using accurate coordinates.

3.7.2 Performance of broadcast and geocast protocols

Using simulation experiments, we evaluate our broadcast and geocast protocols in terms of correctness and efficiency. Then we investigate characteristics of our protocols in terms of node outdegree and hop count. In our experiments, 1000 nodes are randomly placed in a 2D (or 3D) space, each axis of which has a range of 0 to 9999. We run our protocols from each of the 1000 nodes and the average of the 1000 experiments is shown.

3.7.2.1 Correctness

We say that a broadcast or geocast protocol is *correct* if it delivers a message to every target node. That is, GRPB should deliver a message to all nodes in the system and RadGRPM should deliver a message to all nodes within the given radius. Recall that both GRPB and RadGRPM are proved to be correct by Theorem 3 and Theorem 4, respectively. In every one of many thousands of experiments we conducted, GRPB and RadGRPM always worked correctly, namely, delivered a message to every target node.

3.7.2.2 Efficiency

We define *efficiency* of a broadcast or geocast protocol as the ratio of the number of target nodes to the number of message transmissions. For example, if a protocol uses 100 messages to deliver to 50 target nodes, its efficiency is 50%. Ideally each target node needs to receive exactly one message and non-target nodes should not receive any messages, in which case efficiency is 100%.

There are two sources of inefficiency: (a) a non-target node receives a message and (b) a target node receives a message more than once. Our protocols are carefully designed such that non-target nodes do not receive any message and very few target nodes receive duplicate messages. A small number of duplicate messages are sent due to limitation of local knowledge at some nodes.

Figure 3.9 shows the distribution of number of messages delivered at a node in GRPB. The solid line represents the result in 2D and the dashed line represents the result in 3D. In both results, most of nodes receive the broadcast message exactly once. Most of the other nodes receive the message twice. The efficiency is 96.1% in 2D and 88.3% in 3D.

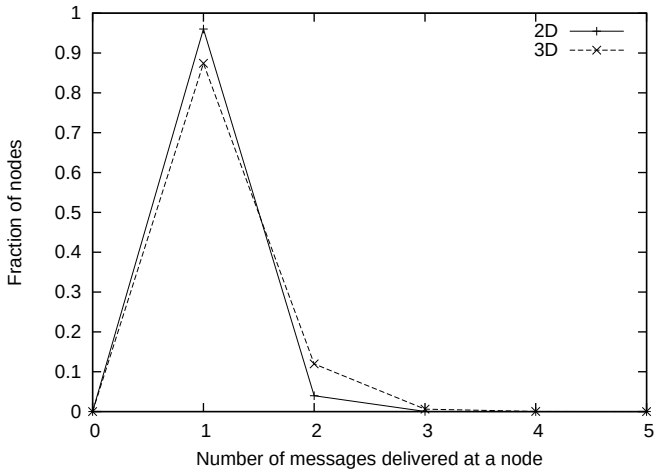

Figure 3.9: Distribution of number of messages delivered at a node (GRPB).

Figure 3.10 shows the distribution of number of messages delivered at a node in RadGRPM. The results of radius 1000 in 2D, radius 3000 in 2D, radius 2000 in 3D, and radius 5000 in 3D are shown. The average number

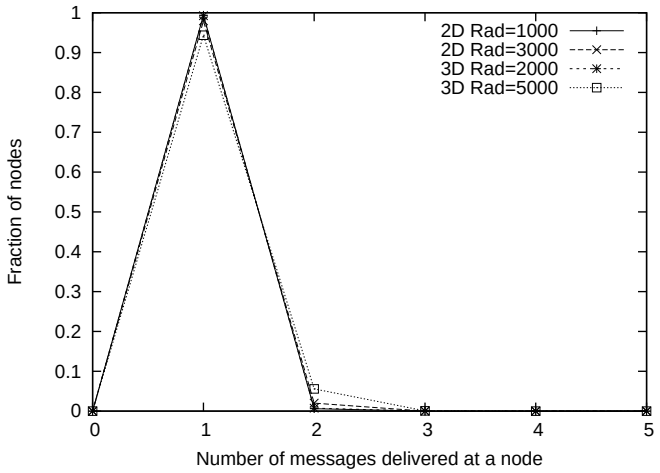

Figure 3.10: Distribution of number of messages delivered at a node (Rad-GRPM).

of target nodes in each case is 28.2, 215.4, 26.3, and 280.2, respectively. The efficiency in each case is 99.4%, 98.0%, 99.3%, and 94.6%, respectively.

Figures 3.11 and 3.11 shows a trend that efficiency decreases as the number of target nodes increases. However, the efficiency is still very high for hundreds of target nodes.

3.7.2.3 Node outdegree and hop count

Node outdegree and hop count are important characteristics of a broadcast/multicast tree.[3] The outdegree of a node is the number of other nodes to which the node sends a message in a broadcast/multicast. A low node outde-

[3]Even though our protocols do not explicitly maintain a broadcast/multicast tree, the graph consisting of all message-forwarding paths is called a tree. Note that, to be strict, the graph may not be a tree due to duplicate messages delivered to a node.

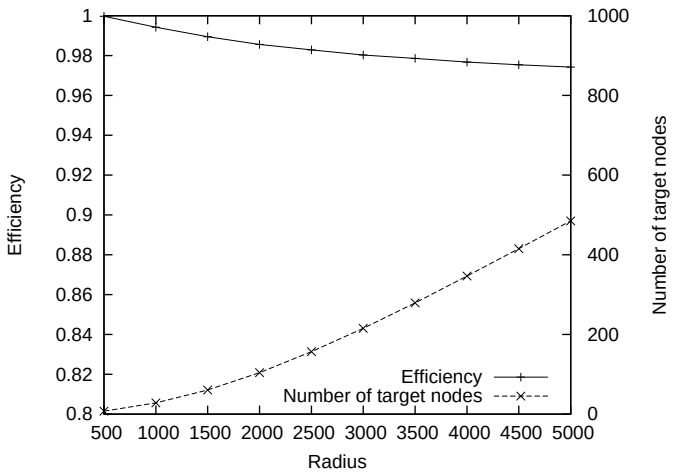

Figure 3.11: Efficiency and number of target nodes (2D).

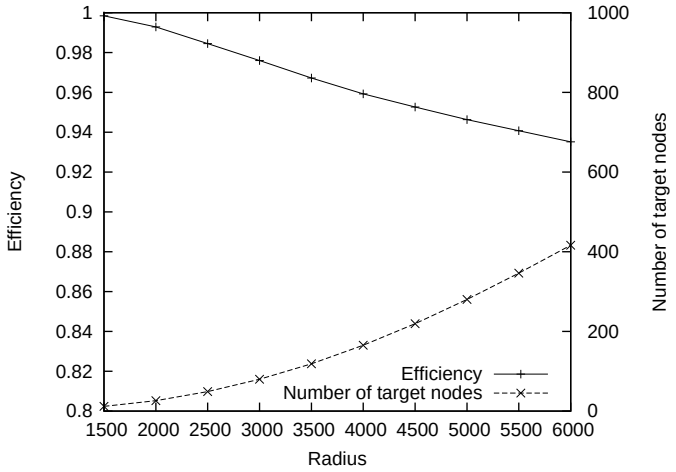

Figure 3.12: Efficiency and number of target nodes (3D).

gree is preferred since a node has limited resources, especially in a peer-to-peer environment. A low hop count is also preferred to reduce delay. There is a trade-off between node outdegree and hop count. That is, a tree cannot have a low node outdegree and a low hop count at the same time.

Figure 3.13 shows the distribution of node outdegree in GRPB. The solid line represents the result in 2D and the dashed line represents the result in 3D. The average node outdegree is higher in 3D than 2D, which is expected since a node has more neighbors in a higher-dimension DT. Both in 2D and 3D, very few nodes have outdegree of four or higher.

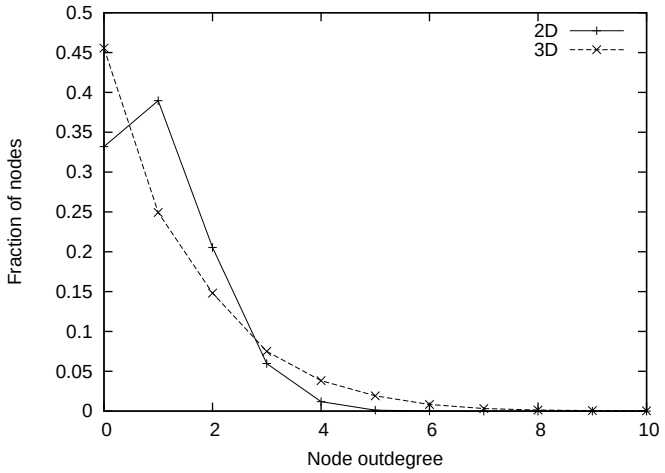

Figure 3.13: Distribution of node outdegree (GRPB).

Figure 3.14 shows the distribution of node outdegree in RadGRPM. The results of radius 1000 in 2D, radius 3000 in 2D, radius 2000 in 3D, and radius 5000 in 3D are shown. In all cases, very few nodes have outdegree of four or higher.

Figure 3.15 shows the distribution of hop count in GRPB. The solid line represents the result in 2D and the dashed line represents the result in 3D. The average hop count is 14.8 in 2D and 5.6 in 3D, which is smaller in 3D since the average node outdegree is higher in 3D.

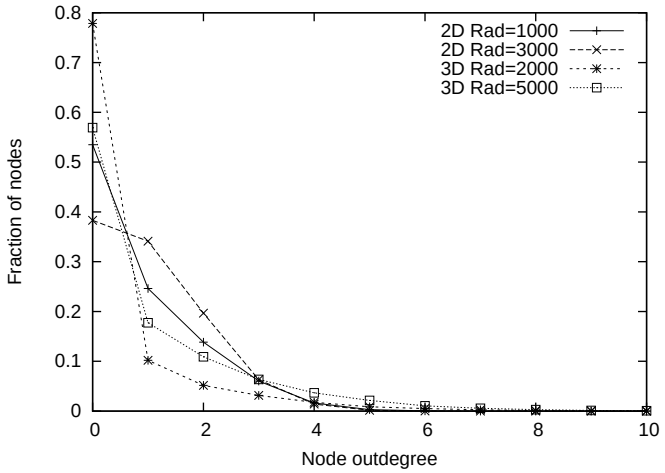

Figure 3.14: Distribution of node outdegree (RadGRPM).

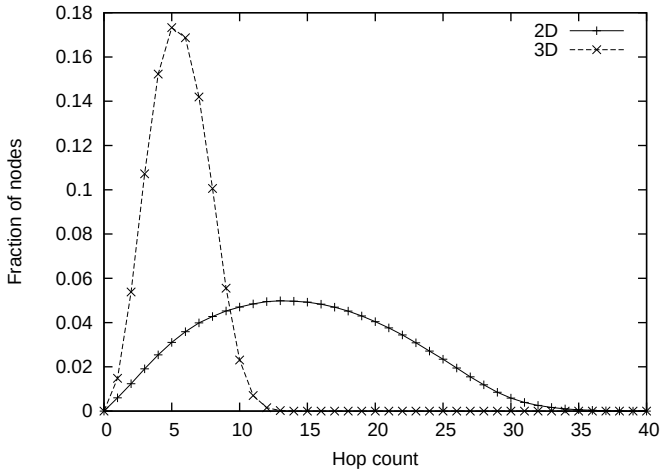

Figure 3.15: Distribution of hop count in GRPB.

Figure 3.16 shows the distribution of hop count in RadGRPM. The results of radius 1000 in 2D, radius 3000 in 2D, radius 2000 in 3D, and radius 5000 in 3D are shown. The average hop count in each case is 2.3, 5.8, 1.5, and 3.2, respectively.

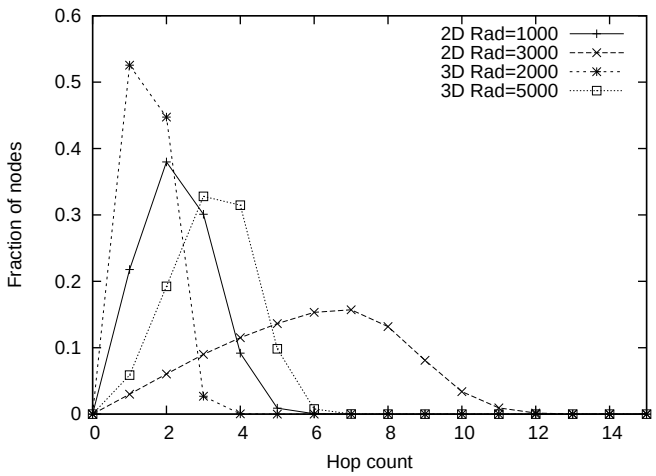

Figure 3.16: Distribution of hop count in RadGRPM.

Figures 3.17 and 3.18 show that the average hop count increases as the number of target nodes increases. Since the average node outdegree is higher in 3D than 2D, the average hop count increases faster in 2D. This is due to the planar nature of a DT. That is, a DT does not have a shortcut that connects a node to a faraway node. If necessary, the average hop count may be reduced by introducing additional shortcut edges to a DT. Correctness GRPB and RadGRPM is not affected by forwarding additional messages over shortcut edges. Efficiency decreases in exchange for a decreased average hop count, since the additional messages are redundant. Tsuboi *et al.* [38] recently proposed Skip Delaunay Network (SDN), which is a hierarchy of DTs and enables a unicast and a geocast protocol with $log(N)$ hop counts. Their geocast protocol (GeoMulticast) delivers a message to a rectangular region. A hierarchy of DTs is also mentioned in [26].

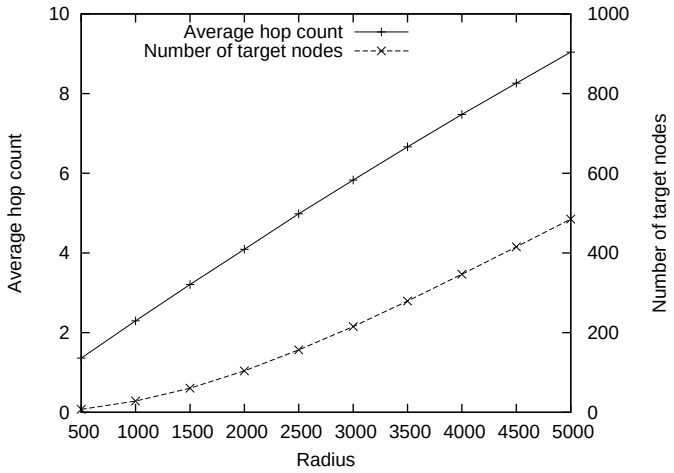

Figure 3.17: Average hop count and number of target nodes in 2D.

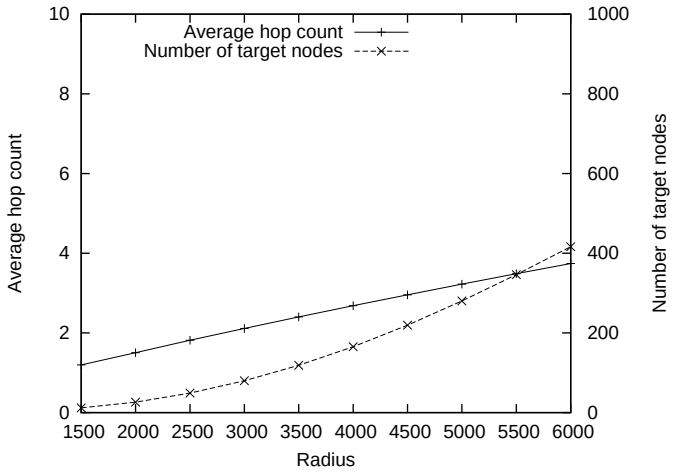

Figure 3.18: Average hop count and number of target nodes in 3D.

Chapter 4

ACE protocol suite

In this chapter we present a new suite of protocols to construct and maintain a distributed DT. In designing the protocols, we aim to achieve three properties: *accuracy*, *correctness*, and *efficiency*. The protocol suite is named ACE [24].

We previously presented three DT protocols in [23]: join and leave protocols that were proved correct and a maintenance protocol that was shown to converge to 100% accuracy after system churn. However, these protocols (to be referred to as our *old* protocols) were designed with correctness as the main goal and their execution requires the use of a large number of messages.

Inspired by the flip algorithm [42], we substantially reduce the number of messages in the ACE join and maintenance protocols. We also introduce a failure protocol in the ACE protocol suite, which uses a proactive approach to efficiently recover from a node failure and a designated monitor node to greatly reduce the number of probe messages. The old leave protocol is very efficient and remains in the ACE protocol suite.

The ACE join, leave, and failure protocols are proved to be correct for a single join, leave, and failure, respectively. We define an accuracy metric such

that accuracy is 100% if and only if the distributed DT is correct. In all of our experiments for systems under churn, the ACE maintenance protocol recovered 100% accuracy after churning stopped. Furthermore, our experimental results show that ACE protocols are *an order of magnitude more efficient* than our old protocols.

4.1 Join protocols

4.1.1 Flip algorithm in a d-dimensional space

Flipping is a well-known and often-used technique to incrementally construct DT in 2D and 3D spaces. A centralized flip algorithm was also proposed to be used for a d-dimensional space [42] and was proved to be correct [8].

Note that two triangles in a 2D space are flipped into two other triangles, and two tetrahedra in a 3D space are flipped into three tetrahedra. In general, two simplexes in a d-dimensional space are flipped into d simplexes. This transformation is called 2-d flipping.

Incremental construction of DT based on flipping is as follows. When a new node is inserted, the simplex that encloses the new node is divided into $(d+1)$ new simplexes. Recall that the circum-hypersphere of a simplex on a DT should not include any other node except for the vertexes of the simplex. Each new simplex is checked whether its circum-hypersphere includes any other node. In case a simplex does include another node, it is flipped to generate new simplexes. The new simplexes are checked, and flipped if necessary. This process continues recursively. The flip algorithm requires a *general position*

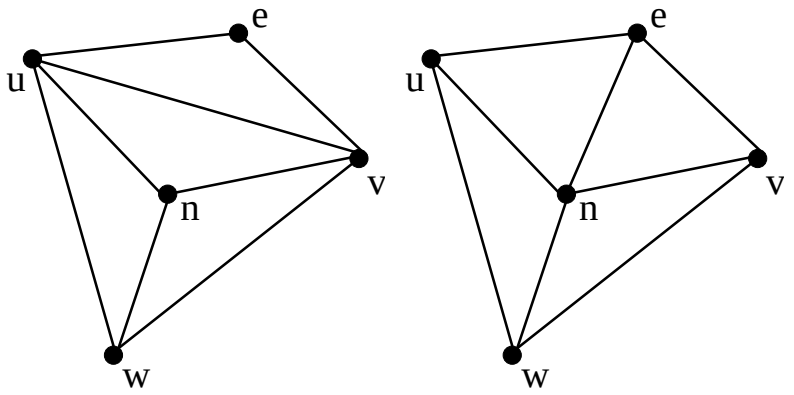

Figure 4.1: An example of flipping in 2D.

assumption, namely: no $d+1$ nodes are on the same hyperplane and no $d+2$ nodes are on the same hypersphere [11].

Figure 4.1 shows an example of flipping in a 2D space. A node n is inserted to a distributed DT. First, the simplex $\triangle uvw$ that encloses n is divided into three new simplexes (left figure). Then each new simplex is checked whether its circum-hypersphere includes any other node. In this example, the circum-hypersphere of $\triangle unv$ includes another node e. Therefore $\triangle unv$ and $\triangle uev$ are flipped into $\triangle une$ and $\triangle vne$ (right figure).

Distributed flip algorithms for joining were proposed for 2D[27], 3D[37], and a d-dimensional space [36]. The centralized flip algorithm is known to be correct (for a single join or serial joins). Since a simplex in d-dimension has $d+1$ nodes, operations at the $d+1$ nodes must be consistent in a distributed algorithm. Correctness has not been explicitly proved for any of the distributed algorithms.

4.1.2 Candidate-set approach

In a previous paper [23], we proposed a join protocol based on the distributed system model using candidate sets and the correctness condition for a distributed DT introduced in Section 2. When a new node n joins a distributed DT, it is first led to the closest existing node z.[1] Then n sends a request to z for mutual neighbors of n and z on $DT(C_z)$. When n receives the reply, n puts the mutual neighbors in its candidate set (C_n) and re-calculates its neighbor set (N_n). If n finds any new neighbors, n sends requests to the new neighbors. This process is repeated recursively. We proved correctness of this protocol for a single join [21].

4.1.3 Novel observation

The flip algorithm and candidate-set approach are fundamentally different. However, it is interesting to note that there is a correspondence between the two. Table 4.1 shows how steps of the two different approaches correspond to each other.

Whereas the two approaches have corresponding steps, the steps are not exactly the same. For example, in step (b), n initially learns $(d + 1)$ neighbors in the flip algorithm. In step (b) of the candidate-set approach, n may be informed of any nodes that z knows. In step (c) of the candidate-set approach, multiple neighbors may send duplicate messages to n to inform n

[1] This can be done using the protocol for finding a closest existing node in [23].

Table 4.1: Correspondence between join protocol in the candidate-set approach and flip algorithm.

	Candidate-set approach	Flip algorithm
(a)	A joining node n is led to a closest existing node z.	A joining node n is led to a closest existing node.
(b)	z calculates local DT using C_z and n, and sends n's neighbors on $DT(C_z)$ to n.	The simplex that encloses n is divided into $(d+1)$ simplexes.
(c)	n contacts each of its new neighbors to see whether there are other potential neighbors.	The new simplexes are checked and flipped if necessary.
(d)	n recursively contacts new neighbors.	New (flipped) simplexes are recursively checked.

of the same new neighbor. In step (c) of the flip algorithm, only one node may reply that a simplex is flipped. This last observation gave us an idea to substantially improve the efficiency of the ACE join protocol.

4.1.4 ACE join protocol

Using the observation described above, we designed the ACE join protocol that is substantially more efficient than our old one. In addition to C_n and N_n, a joining node n maintains a set $N_n^{queried}$, which includes the neighbors that are already queried during its join process. Instead of querying all new neighbors, n queries only one neighbor for each simplex on $DT(C_n)$ that does not include any node in $N_n^{queried}$. Note that only one neighbor in each simplex needs to be queried. If a simplex includes a node $v \in N_n^{queried}$, it means

that the simplex has already been checked by v. Furthermore, queries as well as replies for multiple simplexes may be combined. The ACE join protocol requires the general position assumption, which was not required for the old join protocol.

The ACE join protocol is still based on the candidate-set model and its correctness for a single join is proved using the correctness condition in Theorem 1.

Pseudocode of the ACE join protocol at a node is given in Figures 4.2 and 4.3. The protocol execution loop at a joining node, say n, and the response actions at an existing node, say v, are presented below.[2]

Protocol execution loop at a joining node n

At a joining node n, the ACE join protocol runs as follows with a loop over steps 3-6:

(1) A joining node n is first led to a closest existing node z.

(2) n sends a NEIGHBOR_SET_REQUEST message to z. C_n is set to $\{n, z\}$ and $N_n^{queried}$ is set to $\{z\}$.

Repeat steps 3-6 below until a reply has been received for every NEIGHBOR_SET_REQUEST message sent:

[2]In our current implementation, the joining node processes one NEIGHBOR_SET_REPLY message at a time. We note that if the joining node can process multiple reply messages in step 3 of the loop, the number of query messages may be reduced; this change does not affect the correctness proof for the join protocol.

(3) n receives a NEIGHBOR_SET_REPLY message from a node, say v. The message includes mutual neighbors of n and v on $DT(C_v)$.

(4) n adds the newly learned neighbors (if any) to C_n, and calculates $DT(C_n)$.

(5) Among simplexes that include n on $DT(C_n)$, simplexes that do not include any node in $N_n^{queried}$ are identified as unchecked simplexes. n selects some of its neighbors such that each unchecked simplex includes at least one selected neighbor.

(6) n sends NEIGHBOR_SET_REQUEST messages to the selected neighbors. $N_n^{queried}$ is updated to include the selected neighbors. For the non-selected new neighbors, NEIGHBOR_NOTIFICATION messages are sent.

Response actions at an existing node v

- When v receives NEIGHBOR_SET_REQUEST from n, v puts n into C_v and re-calculates $DT(C_v)$. Then v sends to n NEIGHBOR_SET_REPLY that includes a set of all nodes e such that e, v, and n are in the same simplex on $DT(C_v)$.

- When v receives NEIGHBOR_NOTIFICATION from n, v includes n into C_v and re-calculates $DT(C_v)$. But v does not reply to n.

Join(z) of node u
; Input: u is the joining node, if u is the only node in the system, $z = NULL$; otherwise z is the closest existing node to u.
if $z \neq NULL$ **then**
  Send(z, NEIGHBOR_SET_REQUEST)
  $C_u \leftarrow \{u, z\}$, $N_u \leftarrow \emptyset$, $N_u^{queried} \leftarrow \{z\}$
else
  $C_u \leftarrow \{u\}$, $N_u \leftarrow \emptyset$, $N_u^{queried} \leftarrow \emptyset$
end if

On u's receiving NEIGHBOR_SET_REQUEST from w
if $w \notin C_u$ **then**
  $C_u \leftarrow C_u \cup \{w\}$
  $N_u \leftarrow$ neighbor nodes of u on $DT(C_u)$
end if
$N_w^u \leftarrow \{e \mid e, w$, and u are in the same simplex on $DT(C_u)\}$
Send(w, NEIGHBOR_SET_REPLY(N_w^u))

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

On u's receiving NEIGHBOR_NOTIFICATION from w
if $w \notin C_u$ **then**
  $C_u \leftarrow C_u \cup \{w\}$
  $N_u \leftarrow$ neighbor nodes of u on $DT(C_u)$
end if

Figure 4.2: ACE join protocol at a node u (to be continued).

4.1.5 Correctness of the ACE join protocol

The following theorem states that the ACE join protocol is correct for a single join. Our proof of Theorem 5 is presented below.

Theorem 5. *Let n denote a new joining node, S be a set of existing nodes,*

Update_Neighbors(C_u, N_u) of node u

$N_u^{old} \leftarrow N_u$
$N_u \leftarrow$ neighbor nodes of u on $DT(C_u)$
$N_u^{new} \leftarrow N_u - N_u^{old}$
$T_u^{new} \leftarrow$ set of simplexes that include u on $DT(C_u)$ and
    do not include any node in $N_u^{queried}$
$N_u^{check} \leftarrow Get\_Neighbors\_To\_Check(T_u^{new})$
for all $v \in N_u^{check}$ **do**
  Send(v, NEIGHBOR_SET_REQUEST)
end for
$N_u^{queried} \leftarrow N_u^{queried} \cup N_u^{check}$
$N_u^{notify} \leftarrow N_u^{new} - N_u^{check}$
for all $v \in N_u^{notify}$ **do**
  Send(v, NEIGHBOR_NOTIFICATION)
end for

Get_Neighbors_To_Check(T_u^{new}) of node u

$N'_u \leftarrow \emptyset$
while $T_u^{new} \neq \emptyset$ **do**
  $n \leftarrow$ a vertex of a simplex in T_u^{new}
  $N'_u \leftarrow N'_u \cup n$
  remove all simplexes that include n from T_u^{new}
end while
Return $N'u$

Figure 4.3: ACE join protocol at a node u (continued).

and $S' = S \bigcup \{n\}$. Suppose that the existing distributed DT of S is correct, no other node joins, leaves, or fails, and n joins using the ACE join protocol. Then the ACE join protocol finishes, and the updated distributed DT is correct.

To prove Theorem 5, we first prove Lemmas 6 – 9.

Lemma 6. *Let $S' = S \bigcup \{n\}$ and u be a closest node to n in S. Then u is a neighbor of n on $DT(S')$.*

Proof. (1) n is in $VC_S(u)$, since u is a closest node to n (by Definition 1).

(2) $D(n,n) = 0 < D(n,u) \leq D(n,w)$, for all $w \in S', w \neq n, w \neq u$.

(3) By Lemma 2, u is a neighbor of n on $DT(S')$.

□

Lemma 7. *Let n denote a new joining node, S be a set of existing nodes, and $S' = S \bigcup \{n\}$. Suppose that the existing distributed DT of S is correct and no other node joins, leaves, or fails. Let x be a node to which n sends a NEIGHBOR_SET_REQUEST. Then x is a neighbor of n on $DT(S')$.*

Proof. (1) In the ACE join protocol, x can be either a node in S that is closest to n (in step 1 of join protocol execution loop) or a neighbor of n on $DT(C_n)$ (in step 6 of the loop).

(2) In the former case, by Lemma 6, x is a neighbor of n on $DT(S')$.

(3) In the latter case, a node in $DT(C_n)$ may be n, a closest node to n, or a node received in a NEIGHBOR_SET_REPLY from an existing node. Since n does not send a NEIGHBOR_SET_REQUEST to itself and given Lemma 6, we only need to consider the last case where node x is received in a NEIGHBOR_SET_REPLY from an existing node, say w.

(4) At the beginning of the join process, since the existing distributed DT of S is correct, C_w includes all neighbors of w on $DT(S)$.

(5) After w receives a NEIGHBOR_SET_REQUEST from n, C_w will include n, and thus C_w will include all neighbors of w on $DT(S')$.

(6) w includes x in a NEIGHBOR_SET_REPLY only when x, n, and w are in the same simplex, denoted by T, on $DT(C_w)$.

(7) We next show that T exists on $DT(S')$. Our proof is by contradiction. Suppose T does not exist on $DT(S')$.

(8) Then the space of T on $DT(S')$ is occupied by different simplexes. Let T^* be such a simplex that includes w. Let $y_1, ..., y_d$ be the other nodes in T^*, where d denotes dimensionality of the space. That is, $w, y_1, ..., y_d$ are neighbors of one another on $DT(S')$.

(9) From (5), C_w includes $y_1, ..., y_d$. From (8) and Lemma 3, $w, y_1, ..., y_d$ are neighbors of one another on $DT(C_w)$. Thus T^* also exists on $DT(C_w)$, which contradicts (6) because T and T^* overlap and cannot co-exist on $DT(C_w)$.

(10) From (9), T exists on $DT(S')$, which means that x is a neighbor of n on $DT(S')$.

(11) From (2) and (10), x is a neighbor of n on $DT(S')$ in all cases.

□

Lemma 8. *Let n denote a new joining node, S be a set of existing nodes, and $S' = S \bigcup \{n\}$. Suppose that the existing distributed DT of S is correct and*

no other node joins, leaves, or fails. Let T be a simplex that includes n on $DT(C_n)$ at some time during the ACE join protocol execution and does not exists on $DT(S')$. Let $x \neq n$ be a node in T. Suppose that n sends a NEIGHBOR_SET_REQUEST to x. After n receives a NEIGHBOR_SET_REPLY from x, T is removed from $DT(C_n)$.

Proof. (1) Since the existing distributed DT of S is correct, C_x includes all neighbors of x on $DT(S)$.

(2) After x receives a NEIGHBOR_SET_REQUEST from n, C_x will include n, and thus C_x will include all neighbors of x on $DT(S')$.

(3) Consider the space that T occupies on $DT(C_n)$.

(4) Since T does not exist on $DT(S')$, the space is occupied by two or more different simplexes on $DT(S')$. Let T^* be one of these simplexes that includes both n and x. Such T^* exists because, from Lemma 7, n and x are neighbors on $DT(S')$.

(5) Let d denote dimensionality of the space. There are $d-1$ other nodes in T^*, which are mutual neighbors of n and x on $DT(S')$, and, by Lemma 3, on $DT(C_x)$ as well.

(6) These $d-1$ nodes are included in the NEIGHBOR_SET_REPLY message from x to n.

(7) When n receives the NEIGHBOR_SET_REPLY message, the $d-1$ nodes are included in C_n and, by (5) and Lemma 3, become neighbors of n on $DT(C_n)$.

(8) As a result, T^* is created on $DT(C_n)$. This means T, which overlaps with T^*, is removed from $DT(C_n)$.

□

Lemma 9. *Let n denote a new joining node, S be a set of existing nodes, and $S' = S \bigcup \{n\}$. Suppose that the existing distributed DT of S is correct, no other node joins, leaves, or fails, and n joins using the ACE join protocol. Then the ACE join protocol finishes, and C_n includes all neighbor nodes of n on $DT(S')$.*

Proof. (1) Consider a neighbor v of n on $DT(S')$. We show that v will be included in C_n when the ACE join protocol finishes.

(2) At step 4 of the protocol execution loop, n has some nodes in C_n and calculates $DT(C_n)$.

(3) Suppose that at this time of protocol execution, v is not yet included in C_n. Consider the straight line l from n to v.

(4) Let T be the first simplex on $DT(C_n)$ that l crosses. Such a simplex exists because v is not yet a neighbor of n on $DT(C_n)$. Note that T includes n.

(5) Let the other nodes of T be $x_1, x_2, ..., x_d$, where d denotes dimensionality of the space.

(6) By Lemma 8, the existence of T at this time implies that n has not yet received a NEIGHBOR_SET_REPLY message from any node in T.

(7) Either T includes a node $x_i, 1 \leq i \leq d$ in $N_n^{queried}$ or T does not include any node in $N_n^{queried}$. In the former case, n has sent a NEIGHBOR_SET_REQUEST to x_i and will receive a NEIGHBOR_SET_REPLY message from x_i. In the latter case, at step 5 of the protocol execution loop, n will send a NEIGHBOR_SET_REQUEST to a node $x_j, 1 \leq j \leq d$ in T and will receive a NEIGHBOR_SET_REPLY message from x_j. In each case, when n receives the NEIGHBOR_SET_REPLY message, by Lemma 8, T is removed from $DT(C_n)$ in step 4 of the protocol execution loop.

(8) Afterwards, if v is still not a neighbor of n on $DT(C_n)$ and l crosses another simplex on $DT(C_n)$, protocol execution continues and the above process described in (3) – (7) repeats.

(9) This process finishes in a finite number of iterations since the number of nodes in S is finite and the number of simplexes in S is also finite.

(10) When there is no simplex that l crosses on $DT(C_n)$, l is an edge on $DT(C_n)$, and v is included in C_n.

□

Theorem 5. *Let n denote a new joining node, S be a set of existing nodes, and $S' = S \bigcup \{n\}$. Suppose that the existing distributed DT of S is correct, no other node joins, leaves, or fails, and n joins using the ACE join protocol. Then the ACE join protocol finishes, and the updated distributed DT is correct.*

Proof. By Lemma 9, the join process finishes, and C_n will include all of its neighbor nodes on $DT(S')$. In addition, whenever n discovers a neighbor node v during the process, n sends either NEIGHBOR_SET_REQUEST or NEIGHBOR_NOTIFICATION message to v so that v adds n to C_v. Since the candidate sets of all existing nodes as well as the joining node are correctly updated, the updated distributed DT is correct by Theorem 1. □

4.2 Leave and failure protocols

4.2.1 ACE leave protocol

Consider a node u that leaves gracefully. It notifies a neighbor node v which then removes u from C_v and updates N_v. Such notifications and actions for all neighbors of u are not enough to maintain a distributed DT. This is because after u's leave, v may have a new neighbor w that was not a neighbor of v before u's leave and w may not be in C_v. To design a correct leave protocol, we prove that such w is always a neighbor of u prior to u's leave. 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. More specifically, when a node u leaves, u calculates a local DT of its neighbor set N_u (which does not include u). Then u notifies each of its neighbors, say v, that u is leaving as well as a list

of the neighbors of v on $DT(N_u)$. Upon receiving such notification, v updates its candidate set and neighbor set. In addition, a DELETE(u) message is propagated using the GRPB (greedy reverse-path broadcast) protocol [23]. 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 Figure 4.4. It is essentially the same as our old leave protocol [23], which is very efficient.

4.2.2 ACE failure protocol

We propose a proactive approach to address node failures instead of the reactive approaches used in previous work. The ACE failure protocol is almost as efficient as the ACE leave protocol. It is proved to be correct for a single failure. The main idea is that every node u prepares a contingency plan in case it fails. That is, u calculates a local DT of u's neighbor set N_u. The contingency plan includes, for each neighbor v of u, new neighbor nodes of v after deleting u. Node u selects one of its neighbors, say m, and sends the contingency plan to m, which is called the *monitor node* of u. Then m periodically probes u to check whether u is alive. When m detects failure of u, m sends to each of u's former neighbors its portion of the contingency plan. The protocol pseudocode is given in Figures 4.5 and 4.6. The pseudocode for receiving a DELETE message and pseudocode for GRPB are given in Figure 4.4.

The ACE failure protocol takes over one of the functions of the old

Leave() of node u
Calculate $DT(N_u)$; Note: $u \notin N_u$
for all $v \in N_u$ **do**
  $N_v^u \leftarrow \{w \mid w$ 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
$C_u \leftarrow (C_u \cup N_u^v) - \{v\}$
$N_u \leftarrow$ neighbor nodes of u on $DT(C_u)$
GRPB(DELETE(v), v)

On u's receiving DELETE(w) from v
; w is a deleted node
$C_u \leftarrow C_u - \{w\}$
GRPB(DELETE(w), w)

GRPB(m, s) of node u
; m is a message, s is the source node of broadcast
for all $x \in N_u$, $D(x,s) > D(u,s)$ **do**
  $N_{ux} \leftarrow \{z \in N_u \mid z, u, x$ are in the same simplex
      on $DT(C_u)\}$
  if $D(u,s) \leq D(z,s)$ for all $z \in N_{ux}$ **then**
    Send(x, m)
  end if
end for

Figure 4.4: ACE leave protocol at a node u

maintenance protocol. As a result, the ACE maintenance protocol may be run much less frequently, reducing overall cost of the system. As will be demonstrated by experiments for a system of nodes under churn, the ACE maintenance protocol is still necessary to recover from incorrect system states resulting from concurrent event occurrences.

On change in N_u
$m_u \leftarrow$ the neighbor in N_u with the least ID
; m_u is the monitor node of u
Calculate $DT(N_u)$; Note: $u \notin N_u$
for all $v \in N_u$ **do**
  $N_v^u \leftarrow \{w \mid w$ is a neighbor of v on $DT(N_u)\}$
end for
Send(m_u, CONTINGENCY_PLAN($\{<v, N_v^u> \mid v \in N_u\}$)

On u's receiving CONTINGENCY_PLAN(CP_v) from v
Set $FAILURE\_TIMER_v$ to $T + F$
; T is current time, F is the period of failure probe.

On u's expiration of $FAILURE\_TIMER_v$
Send(v, PING)
Set $PING\_TIMEOUT\_TIMER_v$ to $T + TO$
; T is current time, TO is the timeout value.

On u's receiving PING from v
if $v = m_u$ **then**
  Send(v, PONG($true$))
else
  Send(v, PONG($false$))
end if

Figure 4.5: ACE failure protocol at a node u (to be continued).

Unlike the old maintenance protocol, probes are not duplicated in the ACE failure protocol, since u is probed only by its monitor node. Furthermore, each former neighbor of u receives exactly 1 message upon u's failure. On the other hand, the ACE failure protocol has the overhead of updating a contingency plan whenever a neighbor is added or deleted.

On u's receiving PONG($flag$) from v
Cancel $PING\_TIMEOUT\_TIMER_v$
if $flag = true$ **then**
  Set $FAILURE\_TIMER_v$ to $T + F$
  ; T is current time, F is the period of failure probe.
else
  Cancel $FAILURE\_TIMER_v$
end if

On u's expiration of $PING\_TIMEOUT\_TIMER_v$
Cancel $FAILURE\_TIMER_v$
for all w that CP_v contains $< w, N_w^v >$ **do**
  Send(w, FAILURE(v, N_w^v))
end for
$C_u \leftarrow (C_u \cup N_u^v) - \{v\}$
$N_u \leftarrow$ neighbor nodes of u on $DT(C_u)$
GRPB(DELETE(v), v)

On u's receiving FAILURE(v, N_u^v) from w
$C_u \leftarrow (C_u \cup N_u^v) - \{v\}$
$N_u \leftarrow$ neighbor nodes of u on $DT(C_u)$
GRPB(DELETE(v), v)

Figure 4.6: ACE failure protocol at a node u (continued).

4.2.3 Correctness of the ACE leave and failure protocols

Theorem 6 and Theorem 7 below state that the ACE leave and failure protocols are correct for a single leave and a single failure, respectively. Our proof of Theorem 7 is presented below. A proof of Theorem 6 is provided in [21]. Proof of Theorem 6 is omitted herein because it is very similar to that of Theorem 7.

Theorem 6. *Let S be a set of nodes with a correct distributed DT. Suppose*

that a node $u \in S$ leaves using the ACE leave protocol. Assume that there is no other join, leave, or failure. Then the ACE leave protocol finishes, and the updated distributed DT is correct.

Theorem 7. *Let S be a set of nodes with a correct distributed DT. Suppose that a node $u \in S$ fails and its failure is detected by its monitor node $m_u \in S$, which then executes the ACE failure protocol. Assume that there is no other join, leave, or failure. Then the ACE failure protocol finishes, and the updated distributed DT is correct.*

Lemma 10 below is an important step to prove Theorem 6 and Theorem 7.

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

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

Case A) $D(p,w) < D(p,u)$.

(1) Since $S = S' \cup \{u\}$, we have $D(p,v) < D(p,w) < D(p,x)$ for all $x \in S, x \neq v, x \neq w$.

(2) By Lemma 2, v and w are neighbors on $DT(S)$.

Case B) $D(p,u) \leq D(p,w)$.

(1) Consider a point p' that moves from p toward w.

(2) $D(p', w)$ decreases toward 0 faster than or as fast as $D(p', x)$, for all $x \in S, x \neq w$, as p' moves toward w.

(3) $D(p', u) \geq 0$ and $D(p', v) \geq 0$.

(4) There must be a point where $D(p', w)$ becomes smaller than either $D(p', u)$ or $D(p', v)$. That is, $D(p', v) < D(p', w) < D(p', u) < D(p', x)$ or $D(p', u) < D(p', w) < D(p', v) < D(p', x)$, for all $x \in S, x \neq u, x \neq v, x \neq w$.

(5) From (4) and Lemma 2, v and w are neighbors on $DT(S)$ or u and w are neighbors on $DT(S)$.

□

Theorem 7. *Let S be a set of nodes with a correct distributed DT. Suppose that a node $u \in S$ fails and its failure is detected by its monitor node $m_u \in S$, which then executes the ACE failure protocol. Assume that there is no other join, leave, or failure. Then the ACE failure protocol finishes, and the updated distributed DT is correct.*

Proof. The ACE failure protocol finishes since it does not contain any loop after detection of a failure. In the monitor node, $PING\_TIMEOUT\_TIMER_u$ has expired and $FAILURE\_TIMER_u$ is canceled. The monitor node sends out a FAILURE message only once to each node in the contingency plan of

u. The DELETE(u) message is forwarded by a node x to another node y only if distance $D(y, u)$ is larger than distance $D(x, u)$. Thus the DELETE(u) message is not forwarded in a cycle, and its propagation finishes.

We next show that the updated distributed DT is correct. Let $S' = S - \{u\}$. Consider a node $v \in S'$ and its candidate set C_v. The following case A shows that if v is not a neighbor of u on $DT(S)$, then v is not affected by the failure of u. Case B shows that if v is a neighbor of u on $DT(S)$, v will receive enough information from m_u to correctly update its candidate set.

Case A) v is not a neighbor of u on $DT(S)$. Consider a node $w \in S', w \neq v$. Since $S' = S - \{u\}$ and u is not a neighbor of v, S' includes all neighbors of v on $DT(S)$. If w is a neighbor of v on $DT(S')$, w is also a neighbor of v on $DT(S)$ by Lemma 4. If w is a neighbor of v on $DT(S)$, w is also a neighbor of v on $DT(S')$ by Lemma 3. Therefore the neighbors of v on $DT(S)$ are the same as the neighbors of v on $DT(S')$. Since only u is removed from C_v by the ACE failure protocol, C_v has all neighbors of v on $DT(S')$, and v is not affected by failure of u.

Case B) 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 w was a neighbor of u on DT(S). In the latter case, u's monitor node will notify v that w is its neighbor. In each case, C_v will include w. Therefore C_v will include all neighbor nodes of v on $DT(S')$.

From cases A and B, for each node $v \in S'$, C_v includes all neighbor

nodes of v on $DT(S')$. In addition, $C_v \subset S'$ since u is removed from C_v by propagation of FAILURE and DELETE messages. Therefore by Theorem 1, the updated distributed DT is correct. □

4.3 ACE maintenance protocol

The last member of our protocol suite is the ACE maintenance protocol. Even though the other protocols in the suite – the ACE join protocol, the ACE leave protocol, the ACE failure protocol – are proved to be correct for a single join, leave, and failure, respectively, nodes may join, leave, and fail concurrently for a system under churn. As to be shown by experimental results in Figure 4.11, neither our protocols without a maintenance protocol nor Simon *et al.*'s algorithms can recover a correct distributed DT after system churn. In that sense, our protocol suite is incomplete without a maintenance protocol, and so is Simon *et al.*'s set of insertion and deletion algorithms.

By Theorem 1, for a distributed DT to be correct, each node u must include in its neighbor set C_u all of its neighbor nodes on the global DT. This was one goal that our old maintenance protocol was designed to achieve. To that end, each node u periodically queries each of its neighbors to find any new neighbor of u that u is not aware of.

We found that running the maintenance protocol frequently requires a large communication cost. Note that the goal of a maintenance protocol is similar to that of a join protocol, namely, finding new neighbors. Therefore, we use the same technique as in the design of our ACE join protocol. That

is, we reduce communication cost of the maintenance protocol by eliminating messages with redundant information. Instead of querying all neighbors, a node u queries only one node for each simplex that includes u. Since a neighbor node may be included in multiple simplexes, the number of queried neighbors is much less than the number of all neighbors.

Another goal of the old maintenance protocol was failure detection and recovery. In the old maintenance protocol, probing a node u was carried out by all neighbors of u. In the ACE suite, the ACE failure protocol takes over the task of failure detection and recovery, where a node is probed by only one of its neighbor nodes. Thus the overall cost of the ACE maintenance and failure protocols is much less than the cost of the old maintenance protocol.

Although failure recovery is not a primary goal of the ACE maintenance protocol, if a failure is detected by a message timeout, this information is propagated via REMOVE messages. This may be necessary in case of concurrent failures. REMOVE messages are propagated using the GRPB (greedy reverse-path broadcast) protocol [23]. The ACE maintenance protocol pseudocode is given in Figure 4.7. The pseudocode for GRPB is given in Figure 4.4. Actions for receiving NEIGHBOR_SET_REQUEST and NEIGHBOR_SET_REPLY messages and the functions, Update_Neighbors and Get_Neighbors_To_Check, are the same as given in Figures 4.2 and 4.3, *with the addition* of one line of code to set $NS\_TIMEOUT\_TIMER_v$ when node u sends a NEIGHBOR_SET_REQUEST message to node v and one line of code to cancel $NS\_TIMEOUT\_TIMER_v$ when u receives a NEIGHBOR_SET_REPLY

message from v.

Note that NEIGHBOR_SET_REQUEST and NEIGHBOR_SET_REPLY messages are used in both ACE join and maintenance protocols. The timeout mechanism to detect node failures may also be utilized in the ACE join protocol, but we did not enable it in the join protocol when we ran the experiments presented in the next section.

In our current implementation of the ACE leave and failure protocols, we have one modification to their pseudocode in Figure 4.4 that greatly reduces communication cost. More specifically, when a node u receives a DELETE(v), u forwards it by GRPB only if v is in C_u. We found that if v is not in C_u, it is very rare for v to be present in the candidate sets of nodes one or more hops further away from the source node than u. For a system under churn and running the maintenance protocol, these rare cases can be repaired by the maintenance protocol.

4.4 Accuracy metric for a system under churn

We define an accuracy metric as in [23], which we will use to evaluate experiments for a system of nodes under churn. We consider a node to be *in-system* from when it finishes joining to when it starts leaving. Let DDT_S be a distributed DT of a set of in-system nodes S. (Note that some nodes may be in the process of joining or leaving and not included.) Let $N_{correct}(DDT_S)$ be the total number of correct neighbor entries of all nodes and $N_{wrong}(DDT_S)$ be the total number of wrong neighbor entries of all nodes on DDT_S. A neighbor

On u's expiration of $PERIOD\_TIMER$
$N_u^{queried} \leftarrow \emptyset$
$T_u \leftarrow$ set of simplexes that include u on $DT(N_u \cup \{u\})$
$N_u^{check} \leftarrow Get\_Neighbors\_To\_Check(T_u)$
for all $v \in N_u^{check}$ **do**
  Send(v, NEIGHBOR_SET_REQUEST)
  Set $NS\_TIMEOUT\_TIMER_v$ to $T + TO$
  ; T is current time, TO is the timeout value.
end for
Set $PERIOD\_TIMER$ to $T + P$
; P is the period of maintenance protocol.

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

On u's receiving REMOVE(v, s) from x
; v is a removed node, s is the source node of broadcast
if $v \in C_u$ **then**
  $C_u \leftarrow C_u - \{v\}$
  Update_Neighbors(C_u, N_u)
  GRPB(REMOVE(v, s), s)
  ; s in the REMOVE message is passed to next-hop nodes
  ; s is also given to GRPB function to be used at this hop
end if

Figure 4.7: ACE maintenance protocol at u.

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 undirected and are thus counted twice when compared to

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

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

Observation 1. *The accuracy of a distributed DT is 1 if and only if the distributed DT is correct.*

Proof. (if) If the distributed DT is correct, $N_{correct}(DDT_S) = 2 \times N(DT(S))$ and $N_{wrong}(DDT_S) = 0$, resulting in accuracy of 1.

(only if) When accuracy is 1, we have $N_{correct}(DDT_S) - N_{wrong}(DDT_S) = 2{\times}N(DT(S))$. Since $N_{wrong}(DDT_S) \geq 0$, we get $N_{correct}(DDT_S) \geq 2{\times}N(DT(S))$. Also, $N_{correct}(DDT_S) \leq 2{\times}N(DT(S))$. It then follows that $N_{correct}(DDT_S) = 2{\times}N(DT(S))$ and $N_{wrong}(DDT_S) = 0$. That means the distributed DT is correct. □

4.5 Performance of ACE protocol suite

In all experiments presented in this chapter, each node has randomly generated coordinates. First, to demonstrate effectiveness of the ACE maintenance protocol, we designed an experiment for a system with an initial unidirectional ring configuration. The system begins with a barely connected graph of 100 nodes, in which each node initially knows only one other node. That is, node p_i, $1 \leq i \leq 99$, initially has only p_{i-1} in its candidate set and its neighbor set; node p_0 knows p_{99}. Figure 4.8 shows change in accuracy of

the distributed DT as the ACE maintenance protocol runs. Each curve represents the average accuracy from 100 runs of simulation. Each vertical bar represents the range of accuracy values from 10th percentile to 90th percentile. Note that the ACE maintenance protocol achieved a correct distributed DT within a few rounds of protocol execution except in 2D.[3] In 2D, eight out of 100 runs of simulation resulted in network partitioning, decreasing the average accuracy value. To see why network partitioning occurs, consider an initial configuration where the 100 nodes exist as two clusters on the left and right sides of a space. Suppose that nodes a and b are the leftmost nodes, nodes x and y are the rightmost nodes, and the left and right clusters are connected by only two directed edges; namely, initially a knows x and y knows b. After the maintenance protocol runs and a knows some nearby nodes, x is no longer a neighbor of a on $DT(C_a)$. Similarly, b is no longer a neighbor of y on $DT(C_y)$. Although x is still in C_a and b still in C_y, x and b are not used any longer. Thus the network is partitioned into the left and right clusters. Network partitioning did not occur in 3D or higher dimensions, in which nodes are more densely connected after the first round than in 2D.

Figure 4.9 shows communication costs of join protocols. Each curve shows the number of messages for 100 serial joins, increasing the system size from 200 nodes to 300 nodes, for different dimensionalities. The ACE join protocol has much less cost than our old join protocol, and is slightly better

[3]Each round corresponds to a 10-second period during which each node executes the maintenance protocol once.

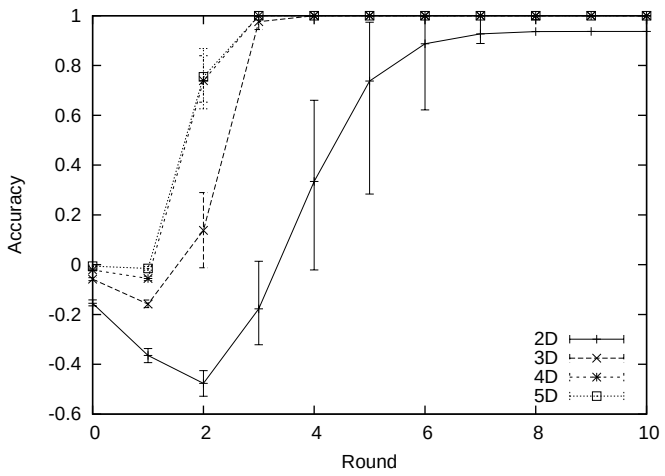

Figure 4.8: Accuracy of the ACE maintenance protocol for a system with an initial unidirectional ring configuration.

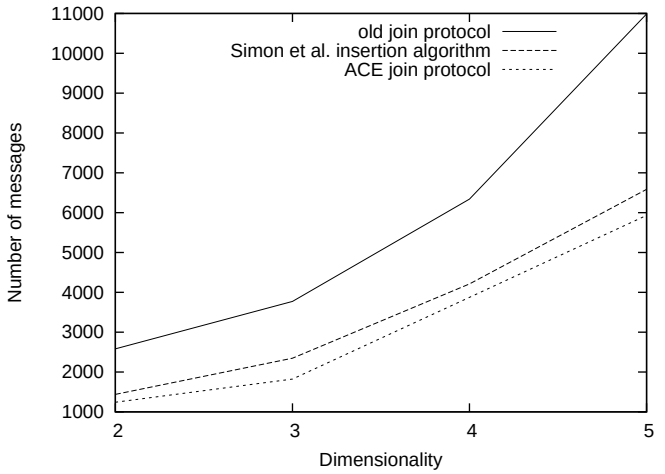

Figure 4.9: Costs of join protocols for 100 serial joins.

than Simon *et al.*'s improved entity insertion algorithm.

Figure 4.10 compares communication costs of the ACE failure protocol and Simon *et al.*'s improved entity deletion algorithm. The number of messages

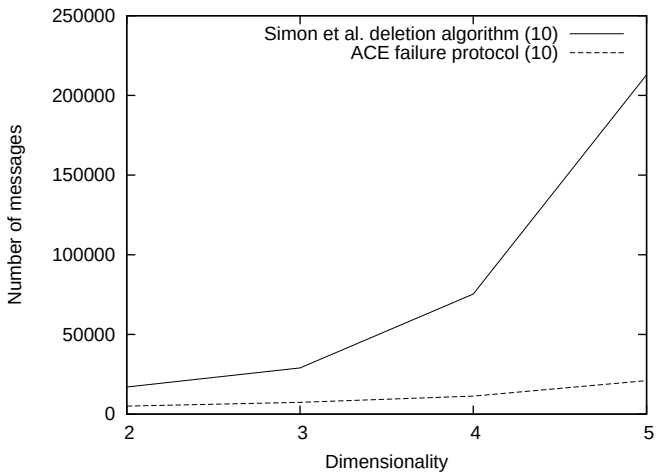

Figure 4.10: Costs of failure protocols for 100 serial failures.

used to recover from 100 serial failures from 300 initial nodes is measured. Both the ACE failure protocol and Simon *et al.*'s deletion algorithm use the same probing period of 10 seconds. The ACE failure protocol is much more efficient than Simon *et al.*'s improved entity deletion algorithm.

Figure 4.11 shows accuracy of ACE protocols without a maintenance protocol and Simon *et al.*'s improved algorithms under system churn. (Our old protocol suite is not shown because it does not have a failure protocol and is not usable without a maintenance protocol.) Each curve represents the average accuracy from 100 runs of simulation. Each vertical bar represents the range of accuracy values from 10th percentile to 90th percentile. From a correct distributed DT of 400 initial nodes in 3D, 100 concurrent joins and 100 concurrent failures occur from time 10 to 110 seconds, with an average inter-

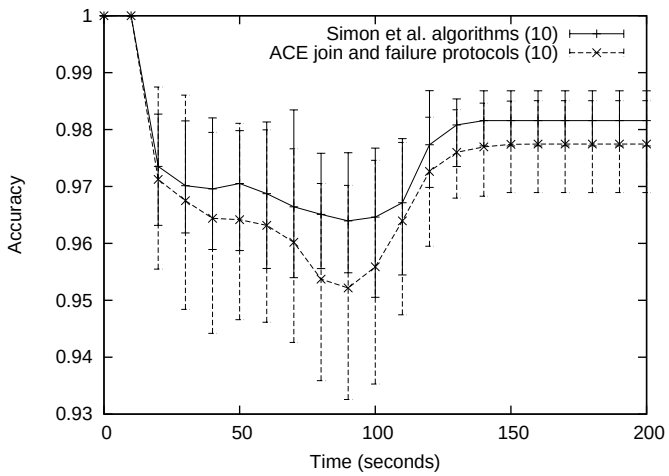

Figure 4.11: Accuracy without a maintenance protocol under system churn (join and fail).

arrival time of 1 second for both joins and failures.[4] In the ACE failure protocol as well as Simon *et al.*'s entity deletion algorithm, nodes are probed every 10 seconds. The accuracy of the distributed DT is measured every 10 seconds. Both the ACE join and failure protocols and Simon *et al.*'s entity insertion and deletion algorithms cannot fully recover after system churn, resulting in an incorrect distributed DT. The results in Figure 4.11 demonstrate that a maintenance protocol is really needed for a system under churn.

Figure 4.12 compares the accuracy of our old and ACE protocol suites including a maintenance protocol under system churn, where nodes join, leave, and fail concurrently. (Simon *et al.*'s algorithms are not shown because they

[4]By Little's Law, for a system size of 400 nodes, the average lifetime of a node is 400 seconds. For P2P file sharing systems, for example, this is considered a very high churn rate [35].

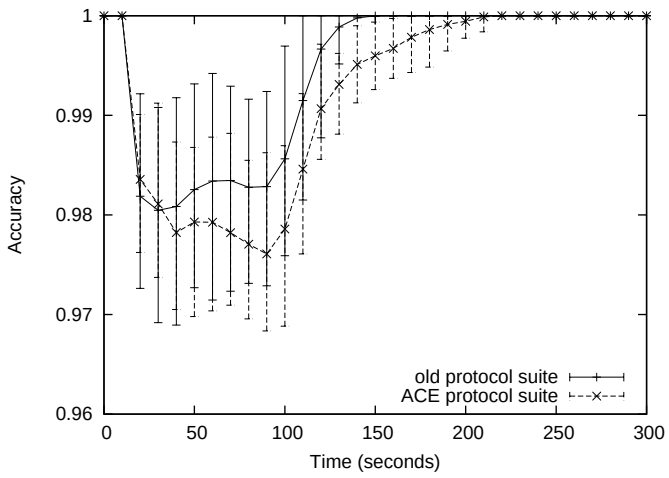

Figure 4.12: Accuracy of the old and ACE protocol suites under system churn (join, leave, and fail).

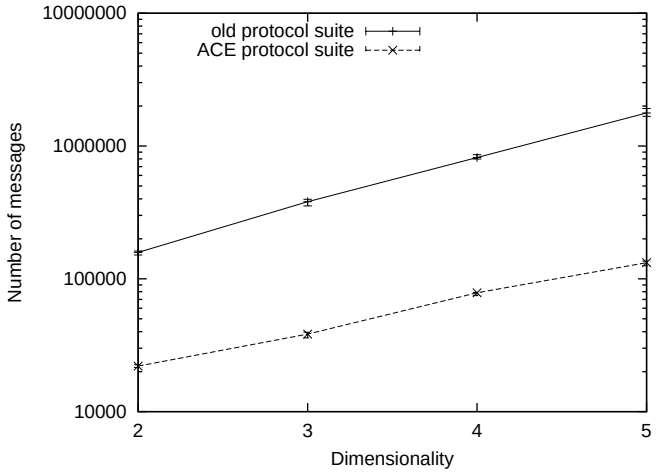

Figure 4.13: Costs of the old and ACE protocol suites under system churn (join, leave, and fail).

do not have a maintenance protocol to recover from incorrect system states during churn.) Each curve represents the average accuracy from 100 runs of simulation. Each vertical bar represents the range of accuracy values from 10th

percentile to 90th percentile. The scenario is similar to that of the previous experiments except that nodes either gracefully leave or fail instead of all failing.[5] From a correct distributed DT of 400 initial nodes in 3D, 100 joins, 50 leaves, and 50 failures occur from time 10 to 110. The average inter-arrival time is 1 second for joins, 2 seconds for leaves, and 2 seconds for failures. The old maintenance protocol is run every 10 seconds. The ACE maintenance protocol is run every 30 seconds. The ACE failure protocol uses a probing period of 10 seconds. After system churn stops at time 110 seconds, accuracy converges to 100% in every experiment for both protocol suites. The average accuracy of the ACE protocols is slightly lower than the average accuracy of our old protocols. The ACE protocols also take a longer time to converge to a correct distributed DT, in part due to the use of a longer period for the maintenance protocol (30 seconds instead of 10 seconds).

Figure 4.13 shows the communication costs of our old and ACE protocol suites in the same churn experiments. Each curve represents the average cost from 10 runs of simulation. Each vertical bar represents the range of all values from the 10 runs; the variance of these simulation results is small. The vertical scale for number of messages is logarithmic. Note that the ACE protocol suite provides an order of magnitude improvement in efficiency compared to the old protocol suite. Furthermore, the two curves diverge slightly indicating that efficiency improvement increases as the dimensionality increases (from 2 to 5).

[5]We have experimental results for the same scenario as the previous experiments showing accuracy and cost performance of our old and ACE protocol suites under system churn. The results are similar to those presented in Figure 4.12 and Figure 4.13.

Chapter 5

Impact of inaccurate coordinates

DT is defined in a Euclidean space. That is, each node has its coordinates in the Euclidean space. Thus in our system model, we have assumed that each node is located in a Euclidean space and knows its coordinates. In the model space, DT has desirable properties such that application protocols run correctly and efficiently on a DT. For example, greedy routing always succeeds and has a short routing path. GRPB and RadGRPM deliver a message to all target nodes using minimal number of messages. However, the model does not always fit reality in practice. And overall performance of applications may be affected by discrepancy between the model and the reality.

We consider the following three categories of relationship between the model and the reality.

The reality does not exist. In this category, the reality does not exist and the model exists by itself. For example, a distributed virtual environment such as multiplayer on-line games may be a Euclidean space by itself, and each entity has coordinates in the virtual space. Therefore exact coordinates can be used to construct a distributed DT.

The reality is a Euclidean space. Examples of this category are wireless ad-hoc networks and sensor networks, where coordinates are defined as the geographic location of nodes or sensors. A node estimates its geographic location and uses it as its coordinates. Error in the location estimation is the source of discrepancy between the model and the reality. We refer to the distance between the model coordinates and the actual coordinates as **coordinate error**. Accuracy of location estimation varies depending on technology. For example, a typical GPS device has several meters of error range.

The reality is not a Euclidean space. An example of this category is using virtual coordinates such as those obtained by GNP [31] and Vivaldi [4] as coordinates of nodes in Internet. Internet itself may not be a Euclidean space, and true coordinates of a node are not defined. Instead, quality of virtual coordinates can be measured by how accurately a distance in the virtual coordinate space predicts network delay between nodes.

In greedy routing, which is the basis of our application protocols, a destination is designated by its known coordinates, not by a different type of address, e.g. an IP address. Even if arbitrary coordinates are used to construct a distributed DT, greedy routing always succeeds to deliver a message to the destination node represented by the coordinates. However, the quality of the routing path is affected by coordinate inaccuracy, as will be investigated later.

RadGRPM is also correct in the sense that it delivers a message to all nodes whose known (inaccurate) coordinates are within a target region. If the requirement of an application is to deliver a message to all nodes whose actual coordinates are within a target region, the radius of the target region should be increased to account for the coordinate inaccuracy.

5.1 Impact of coordinate error on greedy routing performance

We first investigate the impact of coordinate error on the performance of greedy routing. We assume a situation where each node has some error in determining its coordinates. Thus a distributed DT is constructed using the known (inaccurate) coordinates, and greedy routing is performed using the inaccurate coordinates. Note that greedy routing still succeeds even if inaccurate coordinates are used. However, the path length may be affected due to coordinate inaccuracy. Figure 5.1 illustrates an example. a, b, c, and d represent accurate coordinates of four nodes, and a', b', c', and d' represent represent inaccurate coordinates of the nodes, respectively. Considering greedy routing from a to c. Using accurate coordinates, greedy routing would select path $\overline{abc}$ (thick line). However, if inaccurate coordinates are used, path $\overline{a'd'c'}$ (dotted line) would be selected rather than $\overline{a'b'c'}$. Then the actual path would be $\overline{adc}$ (thin line), which is longer than $\overline{abc}$.

As a measure of the impact of coordinate error on the performance of greedy routing, we use the ratio of length of greedy routing path using inaccu-

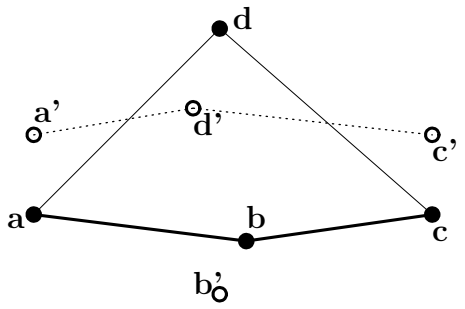

Figure 5.1: An example of longer routing path due to inaccurate coordinates. a, b, c, and d represent accurate coordinates, and a', b', c', and d' represent their respective inaccurate coordinates. If accurate coordinates are used, greedy routing would select path $\overline{abc}$ (thick line). However, greedy routing using inaccurate coordinates would result in path $\overline{adc}$ (thin line), which is longer than $\overline{abc}$.

rate coordinates (such as $\overline{adc}$ in Figure 5.1) to length of greedy routing path for the same source and destination nodes using accurate coordinates (such as $\overline{abc}$ in Figure 5.1). Figure 5.2 shows the ratio for varying degree of coordinate error. x axis represents average coordinate error, which is normalized by average inter-neighbor distance. Note that coordinate errors as large as average inter-neighbor distance would result in a completely different DT. Each curve represents the average result of 100 experiments for different dimensionalities. Each vertical bar indicates the range of results from 10th percentile to 90th percentile. For a reasonable range of coordinate error, the path length is not much affected. When average coordinate error is a quarter of average inter-neighbor distance, the path length is around 1.2 times of that using accurate coordinates. For example, consider a scenario where 1000 nodes are randomly placed in a 1km $\times$ 1km area. Then nodes are around 33m apart on the average along each axis, and a few meters of error range of a typical GPS device would

be around one-tenth of average inter-neighbor distance, in which case greedy routing will perform almost as good as when accurate coordinates are used.

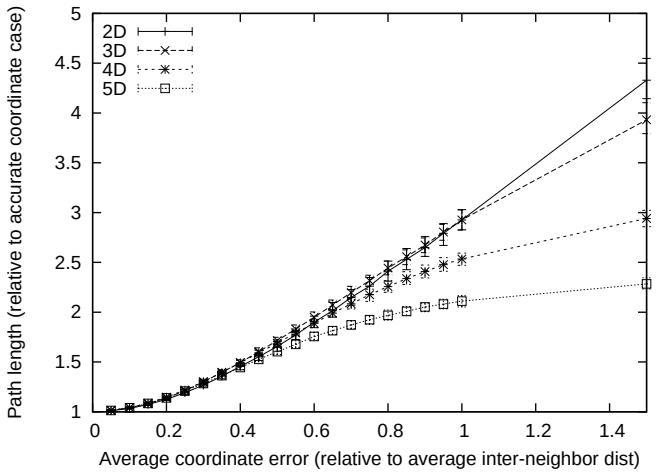

Figure 5.2: Ratio of greedy-routing path length on a distributed DT using inaccurate coordinates to that on a distributed DT using accurate coordinates.

5.2 Impact of coordinate error on greedy routing in Internet

Figure 5.3 shows the performance of greedy routing in Internet when GNP [31] virtual coordinates are used. GNP virtual coordinates are calculated for 1000 nodes in MIT King dataset [10], using 10 landmark nodes that have delay measurement data to all the other nodes. Note that accurate coordinates are not defined in Internet, and neither is coordinate error. Thus the quality of virtual coordinates is measured by the average relative distance error between neighbors. **Relative distance error** for a pair of nodes a and b is defined as follows. Let d_{ab} denote the actual measured distance between a and b. Let

d'_{ab} denote the calculated distance between a and b using their coordinates. Then relative distance error between a and b is defined as $\frac{|d'_{ab}-d_{ab}|}{d_{ab}}$. The lower line in Figure 5.3 represents the average relative distance error between all pairs of neighbor nodes. As expected from the results in [31], the quality of virtual coordinates improves as the dimensionality increases. The upper line in Figure 5.3 represents the ratio of greedy-routing path length to direct distance, which shows that the performance of greedy routing on a distributed DT using the virtual coordinates also improves as the dimensionality increases. Note that, in Internet, once a destination node is reached using greedy routing, the source and destination nodes can directly communicate with each other. In such cases, a relatively high one-time cost of greedy routing to find the destination may be amortized over a long session.

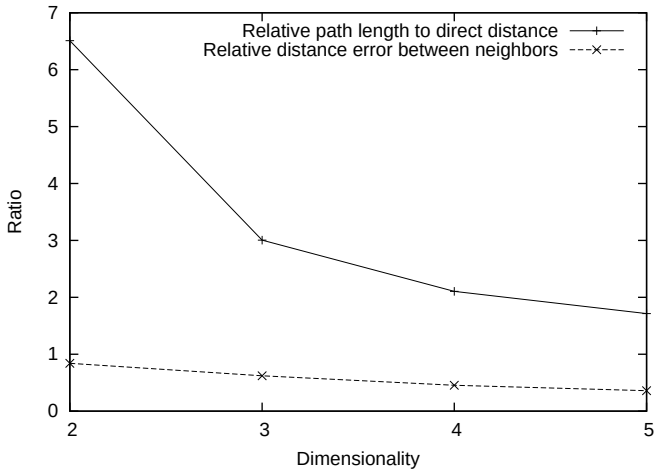

Figure 5.3: Ratio of greedy-routing path length to direct distance on a distributed DT using GNP virtual coordinates.

5.3 Impact of coordinate error on RadGRPM and general geocast

Both of our broadcast and geocast protocols, GRPB and RadGRPM, are based on greedy routing. Since they use the reverse-path of greedy routing, inaccurate coordinates have the same impact on the performance of GRPB and RadGRPM, namely on the path length from a source node to each destination node.

Inaccurate coordinates have another kind of impact on RadGRPM.[1] In geocast applications, a message is often required to be delivered to nodes whose *actual* coordinates are within a target region. For example, a warning message may be sent to all users who are actually within a region. In a sensor database, a query may need to be sent to all sensors that are actually within a region. In such cases, the radius of a target region should be adjusted to account for coordinate inaccuracy. Let r denote the original radius of a target region and E denote the upper bound of coordinate error. Theorem 8 states that if such an upper bound E of coordinate error exists, RadGRPM delivers a message to all nodes that actually are in the target region by using an increased radius of $r + E$. On the other hand, some nodes outside of the target region may unnecessarily receive a message. Theorem 8 also applies to general geocast.

Theorem 8. *Consider each node $u \in S$ with inaccurate coordinates c'_u. Let u's accurate coordinates be denoted by c_u. Suppose that the distance between*

[1] GRPB is not susceptible to this kind of impact.

the accurate coordinates and the inaccurate coordinates is less than or equal to an upper bound E. That is, for every node $u \in S$, $D(c'_u, c_u) \leq E$. The nodes in S form a distributed DT using their inaccurate coordinates. RadGRPM on the distributed DT to all nodes within a range $r + E$ from a point p delivers a message to all nodes whose accurate coordinates are within r from p.

Proof. Consider a node u.

(1) By triangular inequality, $D(p, c'_u) \leq D(p, c_u) + D(c'_u, c_u)$.

(2) By assumption, $D(c'_u, c_u) \leq E$.

(3) If $D(p, c_u) \leq r$, $D(p, c'_u) \leq r + E$. [From (1) and (2).]

(4) If $D(p, c'_u) \leq r+E$, RadGRPM on the distributed DT delivers a message to u. [Correctness of RadGRPM.]

(5) If $D(p, c_u) \leq r$, RadGRPM on the distributed DT delivers a message to u. [From (3) and (4).]

□

Figure 5.4 shows the impact of coordinate inaccuracy on RadGRPM. In an experiment, 1000 nodes are randomly placed in a Euclidean space whose range in each dimension is 10000. Then their coordinates are generated by injecting random coordinate error, the upper bound E of which ranges from 0 to 1000. 1000 sessions of RadGRPM are performed using the inaccurate

coordinates. The center of a target region is randomly selected. The radius of the original target region is fixed as 3000. As is proved in Theorem 8, in all of the experiments, RadGRPM delivered a message to all the target nodes by increasing the radius by the maximum coordinate error E. Average ratio of the number of non-target nodes that receive a message to the number of target nodes is shown. Note that in the original RadGRPM using accurate coordinates, no non-target node receives a message. As in previous figures, each curve represents the average result of 100 experiments for different dimensionalities. Each vertical bar indicates the range of results from 10th percentile to 90th percentile. The results can be explained by the volume of the original and increased target regions. For example, when E is 1000, in d dimension, the expected ratio is $\frac{4^d-3^d}{3^d}$.

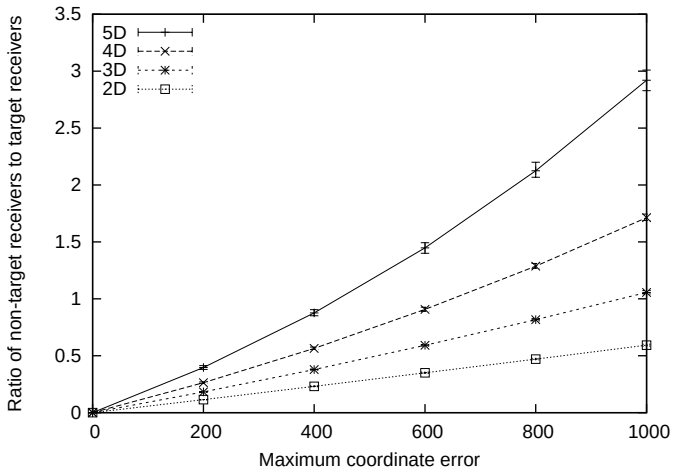

Figure 5.4: Ratio of number of non-target receivers to number of target receivers in general geocast when radius of a target region is 3000.

Figure 5.5 shows message delivery accuracy of RadGRPM using GNP

virtual coordinates. Two different target radii are used: two times and four times of the average inter-neighbor distance. Since upper bound in coordinate error is not defined, RadGRPM was performed without increasing radius. Thus some target nodes did not receive a message. The ratio of false positive delivery, namely the number of non-target nodes that received a message, and false negative delivery, namely the number of target nodes that did not receive a message, to the number of target nodes are shown. Larger radius resulted in better message delivery accuracy. Also, higher dimension has better message delivery accuracy due to improved quality of virtual coordinates.

Figure 5.6 shows message delivery accuracy of RadGRPM using GNP virtual coordinates when target radius is increased by the average inter-neighbor distance. At the cost of increased false positive ratio, false negative ratio was greatly decreased but not completely eliminated, although the number of target nodes that did not receive a message is very small.

We have also attempted further increasing target radius by two times and three times of the average inter-neighbor distance, which could not completely eliminate false negative, either.

5.4 Using geocast for unicast to actual coordinates

In Section 5.1, we have assumed that destination coordinates of greedy routing is given as known coordinates of a node, namely its coordinates in a model space. In that case, correctness of greedy routing is not affected by inaccurate coordinates. Even if arbitrary coordinates are used, greedy routing

always succeeds to deliver a message to the destination node represented by its coordinates.

In some applications, destination coordinates may be given as actual coordinates.[2] In such applications, coordinate error should be taken into consideration. That is, instead of unicast greedy routing, geocast may be used to a region that centers at the destination coordinates with a radius large enough to compensate for the coordinate error.

[2]If a node does not exist at the destination coordinates, greedy routing delivers the message to the node whose known coordinates is the closest to the destination coordinates.

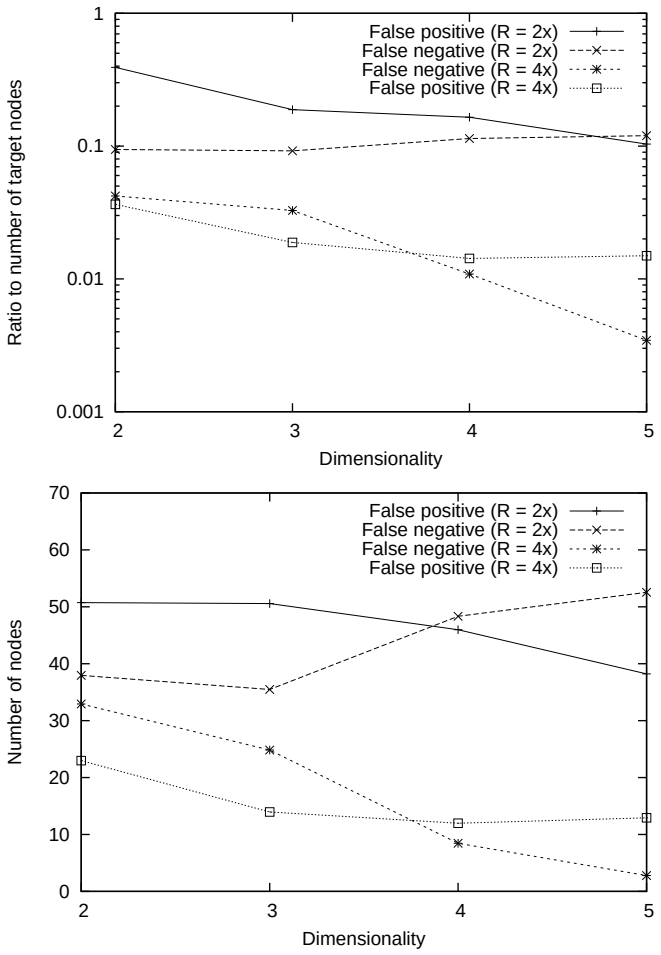

Figure 5.5: Ratio (top) and number (bottom) of non-target nodes that receive a message (false positive) and target nodes that do not receive a message (false negative) in RadGRPM using GNP virtual coordinates.

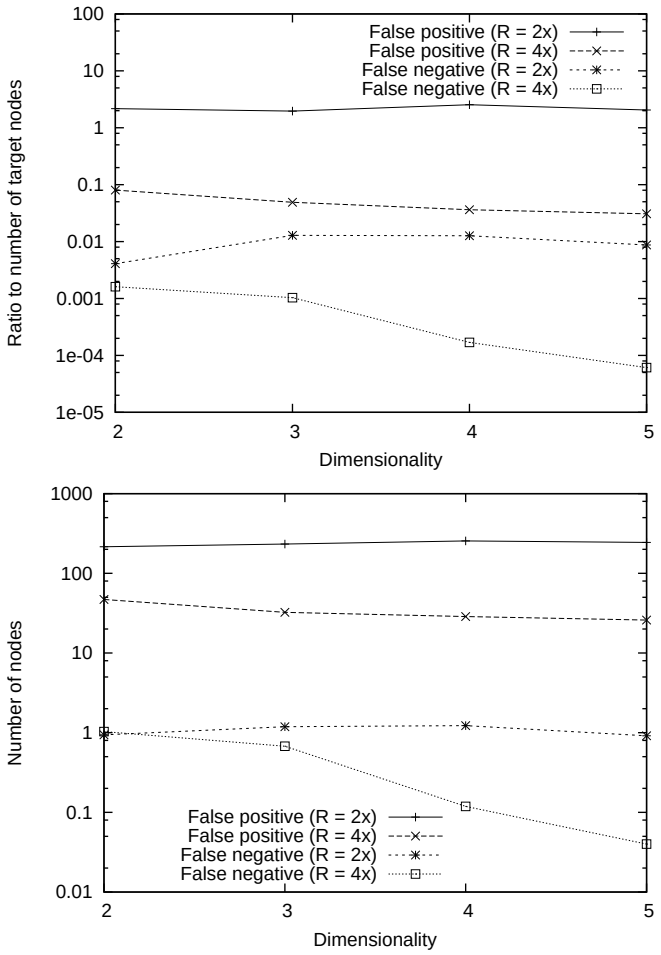

Figure 5.6: Ratio (top) and number (bottom) of non-target nodes that receive a message (false positive) and target nodes that do not receive a message (false negative) in RadGRPM using GNP virtual coordinates and increased radius by average inter-neighbor distance.

Chapter 6

Conclusions

We define a distributed system model for a set S of nodes, in which each node u maintains a set C_u of nodes it knows. Node u determines its neighbor set N_u by calculating $DT(C_u)$. We prove the following basic result (Theorem 1): The distributed DT of S is correct if and only if, for every $u \in S$, C_u includes all neighbor nodes of u on $DT(S)$. Note that C_u is local information while S is global knowledge.

We use the above correctness condition as a guide to design a suite of protocols, named ACE, for a dynamic set of nodes in d-dimension ($d > 1$) to construct and maintain a distributed DT. The join, leave, and failure protocols in the suite are proved to be correct for a single join, leave, and failure, respectively. We define an accuracy metric such that accuracy is 100% if and only if the distributed DT is correct. The ACE suite also includes a maintenance protocol designed to recover from incorrect system states due to concurrent event processing and to improve accuracy. Experimental results show that the ACE protocol suite is highly efficient, it maintains high accuracy for systems under churn, and each system converges to 100% accuracy after churning stopped.

Our experimental results show that ACE protocols are *an order of magnitude more efficient* than our old protocols in [23], which are the only other protocols that have been demonstrated to converge to 100% accuracy after churn. There is a tradeoff, however. During churn periods, the average accuracy of ACE protocols is slightly (a fraction of 1%) lower than the average accuracy of our old protocols. Also, ACE protocols provide slower convergence due in part to the use of a longer period for running the maintenance protocol.

We also design several application protocols including greedy routing, finding a closest existing node, clustering, broadcast, radius geocast, and general geocast. Correctness of the application protocols is discussed and proved, and their performance and characteristics are also investigated. Greedy routing always succeeds on a distributed DT, and the quality of the route is very good. Our broadcast and geocast protocols deliver a message to every target node using minimal number of messages.

In our system model, we have assumed that each node is located in a Euclidean space and knows its coordinates. In practice, a node may not accurately determine its coordinates, and inaccurate coordinates may affect overall performance of applications. We investigate the impact of inaccurate coordinates on performance of greedy routing, broadcast, and geocast protocols. We also discuss how to maintain correct operation of greedy routing and geocast protocols by accounting for coordinate errors.

Bibliography

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

[2] Eliya Buyukkaya and Maha Abdallah. Data management in Voronoi-based P2P gaming. In *Proc. IEEE International Workshop on Digital Entertainment, Networked Virtual Environments, and Creative Technology (CCNC 2008)*, January 2008.

[3] L. P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989.

[4] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proceedings SIGCOMM 2004*, 2004.

[5] Y. K. Dalal and R. M. Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, 21(12):1040–1048, 1978.

[6] B. Delaunay. Sur la sphère vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.

[7] D. P. Dobkin, S. J. Friedman, and K. J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete and Computational Geometry*, 5(1):399–407, 1990.

[8] H. Edelsbrunner. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15(3):223–241, 1996.

[9] R. G. Gallager, P. A. Humblet, and P. M. Spira. A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Transactions on Programming Languages and Systms (TOPLAS)*, 5(1):66–77, 1983.

[10] MIT Parallel & Distributed Operating Systems Group. p2psim King dataset, August 2004. http://pdos.csail.mit.edu/p2psim/kingdata/.

[11] P. M. Gruber and J. M. Wills. *Handbook of convex geometry.* North-Holland, 1993.

[12] S. Y. Hu, J. F. Chen, and T. H. Chen. VON: A scalable peer-to-peer network for virtual environments. *IEEE Network*, 20(4):22–31, Jul/Aug 2006.

[13] J. M. Keil and C. A. Gutwin. The Delauney triangulation closely approximates the complete Euclidean graph. In *Proceedings of the Workshop on Algorithms and Data Structures (LNCS 382)*, pages 47–56. Springer-Verlag London, UK, 1989.

[14] J. Keller and G. Simon. Solipsis: A massively multi-participant virtual world. In *Proc. Int. Conf. Parallel and Distributed Techniques and Applications (PDPTA 03)*, Las Vegas, NV, 2003.

[15] M. S. Kim, T. Kim, Y. J. Shin, S. S. Lam, and E. J. Powers. Scalable clustering of Internet paths by shared congestion. In *Proceedings of 25th*

IEEE International Conference on Computer Communications, pages 1–10, 2006.

[16] M. S. Kim, T. Kim, Y. J. Shin, S. S. Lam, and E. J. Powers. A wavelet-based approach to detect shared congestion. *IEEE/ACM Transactions on Networking*, 16(4):763–776, August 2008.

[17] M. S. Kim, Y. Li, and S. S. Lam. Eliminating bottlenecks in overlay multicast. In *Proceedings of IFIP Networking*, May 2005.

[18] Min S. Kim. *Eliminating bottlenecks in overlay multicast.* PhD thesis, Department of Computer Sciences, UT-Austin, August 2005.

[19] S. S. Lam and H. Liu. Failure recovery for structured P2P networks: protocol design and performance under churn. *Computer Networks*, 50(16), November 2006.

[20] Dong-Young Lee, Eui Kyung Chung, and Simon S. Lam. A radius geocast routing protocol. In *Proc. of IEEE International Conference on High Performance Computing and Communications*, Dalian, China, September 2008.

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

[22] Dong-Young Lee and Simon S. Lam. Efficient and Accurate Delaunay Triangulation Protocols under Churn. Technical Report TR-07-59, The

Univ. of Texas at Austin, Dept. of Computer Sciences, November 2007. Revised May 2008.

[23] Dong-Young Lee and Simon S. Lam. Protocol design for dynamic Delaunay triangulation. In *Proc. of IEEE International Conference on Distributed Computing Systems*, Toronto, Ontario, Canada, June 2007.

[24] Dong-Young Lee and Simon S. Lam. Efficient and accurate protocols for distributed Delaunay trangulation under churn. In *Proc. of IEEE International Conference on Network Protocols*, Orlando, Florida, October 2008.

[25] Y. Li, Y. Zhang, L. L. Qiu, and S. S. Lam. SmartTunnel: A multipath approach to achieving reliability in the Internet. In *Proceedings of IEEE Infocom*, May 2007.

[26] J. Liebeherr and M. Nahas. Application-layer multicast with Delaunay triangulations. volume 3, pages 1651–1655, San Antonio, Texas, November 2001.

[27] J. N. Liebeherr, M. Nahas, and W. Si. Application-layer multicasting with Delaunay triangulation overlays. *IEEE Journal on Selected Areas in Communications*, 20(8):1472–1488, 2002.

[28] Y. Mao, F. Wang, L. Qiu, S. S. Lam, and J. M. Smith. S4: Small state and small stretch routing protocol for large wireless sensor networks. In

Proc. of the 4th USENIX Symposium on Networked System Design and Implementation (NSDI), Cambridge, Massachusetts, April 2007.

[29] T. Melodia, D. Pompili, and I. F. Akyildiz. A communication architecture for mobile wireless sensor and actor networks. In *Proc. of the third Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, Reston, Virginia, USA, September 2006.

[30] J. C. Navas and T. Imielinski. Geocast - geographic addressing and routing. In *Proceedings of the ACM/IEEE Int. Conf. on Mobile Computing and Networking (MobiCom '97)*, Budapest, Hungary, September 1997.

[31] T. S. Eugene Ng and Hui Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proceedings of IEEE Infocom*, New York City, New York, USA, June 2002.

[32] M. Ohnishi, R. Nishide, and S. Ueshima. Incremental construction of Delaunay overlaid network for virtual collaborative space. In *Proc. of the third International Conference on Creating, Connecting and Collaborating through Computing*, pages 75–82, Kyoto, Japan, January 2005.

[33] Robert Pless. Lecture 17: Voronoi Diagrams and Delauney Triangulations. In *http://www.cs.wustl.edu/ pless/506.html*, 2003.

[34] Laura Ricci and Andrea Salvadori. Nomad: Virtual environments on P2P Voronoi overlays. In *First International Workshop on Peer to Peer Networks (PPN 2007, LNCS Vol. 4806)*, November 2007.

[35] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of Multimedia Computing and Networking*, San Jose, California, USA, January 2002.

[36] G. Simon, M. Steiner, and E. Biersack. Distributed dynamic Delaunay triangulation in d-dimensional spaces. Technical report, Institut Eurecom, August 2005.

[37] M. Steiner and E. Biersack. A fully distributed peer to peer structure based on 3D Delaunay Triangulation. In *Proc. of Septièmes rencontres francopohones sur les aspects Algorithmiques des Télécommunications (AlgoTel)*, Presqu'île de Giens, France, May 2005.

[38] Shinji Tsuboi, Tomoteru Oku, Masaaki Ohnishi, and Shinichi Ueshima. Generating skip Delaunay network for P2P Geocasting. In *Proc. of the Sixth International Conference on Creating, Connecting and Collaborating through Computing*, Poitiers, France, January 2008.

[39] Matteo Varvello, Ernst Biersack, and Christophe Diot. Dynamic clustering in Delaunay-based P2P networked virtual environments. In *Proc. of NetGames*, September 2007.

[40] G. Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *J. Reine Angew. Math*, 134:198–287, 1908.

[41] F. Wang, L. Qiu, and S. S. Lam. Probabilistic region-based localization for wireless networks. *ACM SIGMOBILE Mobile Computing and*

Communications Review (MC2R), 11(1):3–14, 2007.

[42] D. F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981.

[43] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: a lightweight network location service without virtual coordinates. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 85–96. ACM Press New York, NY, USA, 2005.

[44] T. Yoo, H. Lee, J. Lee, S. Choi, and J. Song. Distributed kinetic Delaunay triangulation. Technical Report CS/TR-2005-240, KAIST, Korea, 2005.

[45] X. B. Zhang, S. S. Lam, and H. Liu. Efficient group rekeying using application-layer multicast. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, pages 303–313. IEEE Computer Society Washington, DC, USA, 2005.

[46] X. Y. Zhang, Q. Zhang, Z. Zhang, G. Song, and W. Zhu. A construction of locality-aware overlay network: mOverlay and its performance. *IEEE Journal on Selected Areas in Communications*, 22(1):18–28, 2004.

[47] Xincheng Zhang. *Protocol design for scalable and reliable group rekeying.* PhD thesis, Department of Computer Sciences, UT-Austin, June 2005.

Vita

Dong-Young Lee was born in Seoul, Korea on 16 May 1973, the son of Prof. Hwi Gyo Lee and Prof. Jeom Sook Ryu. He received the Bachelor of Science degree in Computer Science from Seoul National University in 1998, and received the Master of Science in Computer Science from the same university in 2000. He started Ph.D. program in Computer Sciences in the University of Texas at Austin in August, 2000.

Permanent address: 16238 RR 620 F#124,
Austin, Texas 78717

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.