Efficient and Accurate Protocols for Distributed Delaunay Triangulation under Churn

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

TR-07-59 November 9, 2007
Revised, September 1, 2008

***Abstract*—We design a new suite of protocols for a set of nodes in d-dimension ($d > 1$) to construct and maintain a *distributed* Delaunay triangulation (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.**

I. Introduction

Delaunay triangulation [2] and Voronoi diagram [14] have a long history and many applications in different fields of science and engineering including networking applications, such as greedy routing, finding the closest node to a given point, broadcast, geocast, etc. [1], [5], [7], [8]. 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 [4].

Research sponsored by National Science Foundation grants CNS-0434515 and CNS-0830939.

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

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 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.

Our objective in this paper is a 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*. We present in this paper 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 paper, 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

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

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 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.

In previous papers [5], [7], we presented examples of networking applications to run on top of a distributed DT, namely: greedy routing, finding the closest existing node to a given point, clustering of network nodes, as well as broadcast and geocast without explicit maintenance of a broadcast/multicast tree. Three DT protocols were presented in [7]: 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 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

[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.

TABLE I
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

by Simon *et al.* [12]. 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 [15] 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 [3]. 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 I. 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.

The organization of this paper is as follows. In Section II, we introduce the concepts and definitions of a distributed DT, present our system model, and a correctness condition for a distributed DT. We present the ACE join protocol in Section III, the ACE failure and leave protocols in Section IV, and the ACE maintenance protocol in Section V. The accuracy metric is defined in Section VI and experimental results are presented in Section VII. We conclude in Section VIII.

II. DISTRIBUTED DELAUNAY TRIANGULATION

A. 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 $VC_S(u)$.*

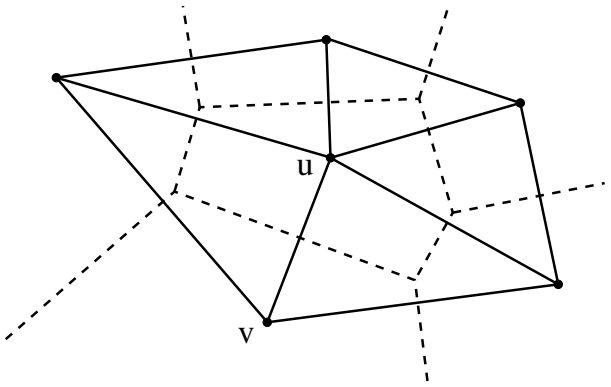

Fig. 1. A Voronoi diagram (dashed lines) and the corresponding DT (solid lines) in a 2-dimensional space.

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 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 is the dual of a Voronoi diagram.[4] Let us denote the DT of S as $DT(S)$.

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

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

That is, a distributed DT is correct when for every node u,

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

N_u is the same as the neighbors of u on $DT(S)$. Since u does not have global knowledge, it is not straightforward to achieve correctness.

B. 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 [8], [10], [12], [13]. The mechanism to obtain coordinates is beyond the scope of this study. Coordinates may be given by an application, a GPS device, or topology-aware virtual coordinates [9].[5] Also when we say a node *u knows* another node v, we assume that u knows v's coordinates as well.

Let S be a set of nodes to construct a distributed DT from. We will present protocols to enable each node $u \in S$ to get to know a set of its nearby nodes including u itself, denoted as C_u, to be referred to as u's *candidate set*. Then u determines the set of its neighbor nodes N_u 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.

C. 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 [6], [7], 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, which has not been published, is presented in the appendix.

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

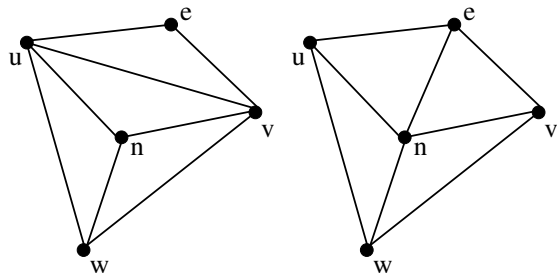

Fig. 2. An example of flipping in 2D.

III. Join protocols

A. 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 [15] and was proved to be correct [3].

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 assumption*, namely: no $d+1$ nodes are on the same hyperplane and no $d+2$ nodes are on the same hypersphere [4].

Figure 2 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 [8], 3D [13], and a d-dimensional space [12]. 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.

B. Candidate-set approach

In a previous paper [7], we proposed a join protocol based on the distributed system model using candidate sets and the correctness condition for a distributed DT introduced in

TABLE II
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.

Section 2. When a new node n joins a distributed DT, it is first led to the closest existing node z.[6] 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 [6].

C. 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 II 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 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.

D. 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 Figure 3. The protocol execution loop at a joining node, say n, and the response actions at an existing node, say v, are presented below.[7]

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:

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.

E. Correctness of the ACE join protocol

Lemma 1. *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*

[6]This can be done using the protocol for finding a closest existing node in [7].

[7]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 in the appendix.

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

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 $N'u$

Fig. 3. ACE join protocol at a node u.

sends a NEIGHBOR_SET_REQUEST to x. After n receives a NEIGHBOR_SET_REPLY from x, T is removed from $DT(C_n)$.

Lemma 2. *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 $DT(S')$.*

The following theorem states that the ACE join protocol is correct for a single join. Our proofs of Lemma 1, Lemma 2, and Theorem 2 are presented in the appendix.

Theorem 2. *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.*

IV. Leave and failure protocols

A. 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 [7].

The protocol pseudocode is presented in Figure 4. It is essentially the same as our old leave protocol [7] which is very efficient.

B. 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 Figure 5. The pseudocode for receiving a DELETE message and pseudocode for GRPB are given in Figure 4.

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

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

Fig. 4. ACE leave protocol at a node u

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.

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.

C. Correctness of the ACE leave and failure protocols

Lemma 3. *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 $DT(S)$.*

Theorem 3 and Theorem 4 below state that the ACE leave and failure protocols are correct for a single leave and a single failure, respectively. Our proofs of Lemma 3 and Theorem 4 are presented in the appendix. A proof of Theorem 3 is provided in [6]. We omit proof of Theorem 3 herein because it is very similar to that of Theorem 4.

Theorem 3. *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 4. *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.*

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

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)

Fig. 5. ACE failure protocol at a node u.

V. 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 10, 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 [7]. The ACE maintenance protocol pseudocode is given in Figure 6. The pseudocode for GRPB is given in Figure 4. The pseudocode for receiving NEIGHBOR_SET_REPLY and pseudocode for Update_Neighbors and Get_Neighbors_To_Check are the same as given in Figure 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 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.

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 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
if $w \in N_u$ **then**
  $N_w^u \leftarrow \{e \mid e, w,$ and u are in the same simplex on $DT(C_u)\}$
else
  $N_w^u \leftarrow \{e \mid e$ is a neighbor of w on $DT(C_u)\}$
end if
Send(w, NEIGHBOR_SET_REPLY(N_w^u))

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

Fig. 6. ACE maintenance protocol at u.

VI. Accuracy metric for a system under churn

We define an accuracy metric as in [7], 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 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.

VII. Experimental results

In all experiments presented in this section, 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 7 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.[8] 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 8 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 than Simon *et al.*'s improved entity insertion algorithm.

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

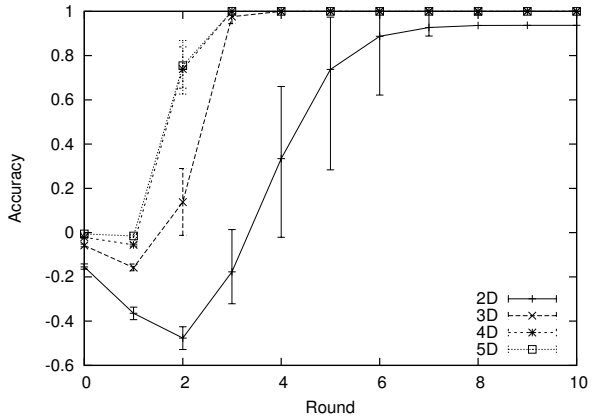

Fig. 7. Accuracy of the ACE maintenance protocol for a system with an initial unidirectional ring configuration.

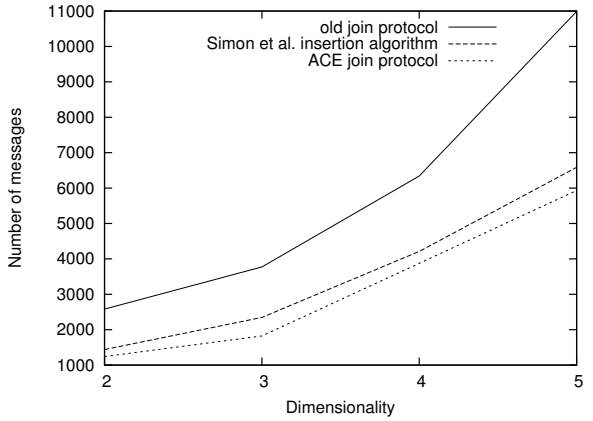

Fig. 8. Costs of join protocols for 100 serial joins.

Figure 9 compares communication costs of the ACE failure protocol and Simon *et al.*'s improved entity deletion algorithm. The number of messages 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 10 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-arrival time of 1 second for both joins and failures.[9] In the

[9]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 [11].

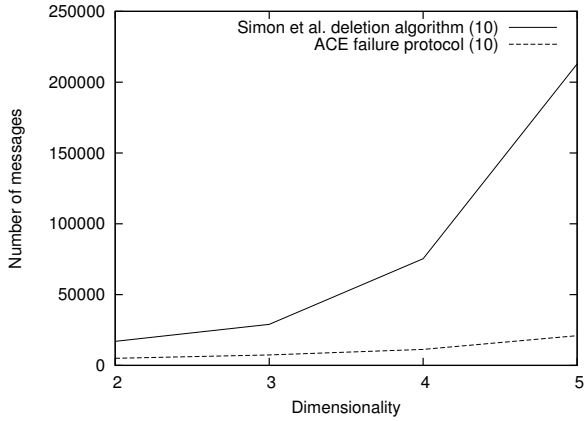

Fig. 9. Costs of failure protocols for 100 serial failures.

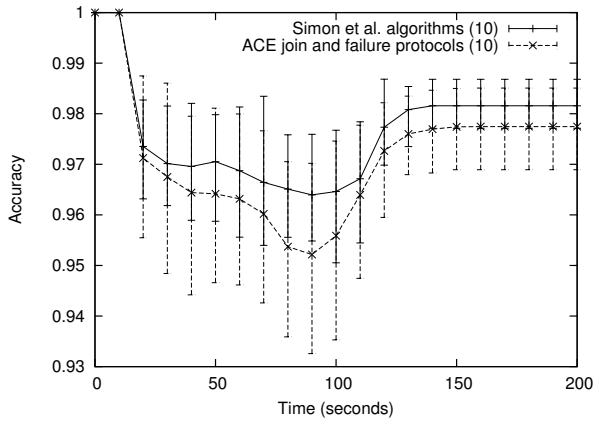

Fig. 10. Accuracy without a maintenance protocol under system churn (join and fail).

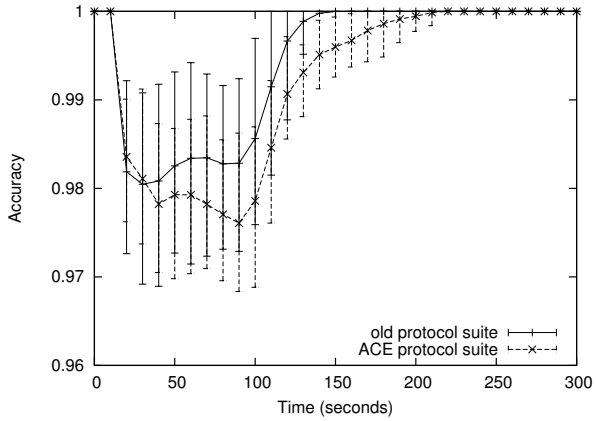

Fig. 11. Accuracy of the old and ACE protocol suites under system churn (join, leave, and fail).

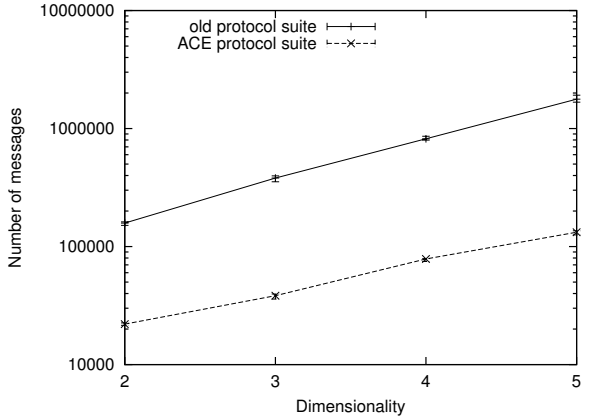

Fig. 12. Costs of the old and ACE protocol suites under system churn (join, leave, and fail).

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 10 demonstrate that a maintenance protocol is really needed for a system under churn.

Figure 11 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 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.[10] 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 12 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]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 11 and Figure 12.

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).

VIII. 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 $DT(S)$. Theorem 1 is proved in the appendix. 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 [7], 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.

Appendix
Proofs of theorems and lemmas

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

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

Lemma A.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 A.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 A.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 A.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 A.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 A.4, v is a neighbor of u on $DT(S)$.

To prove Lemma 1, we first prove Lemmas A.5 and A.6.

Lemma A.5. *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 A.2, u is a neighbor of n on $DT(S')$.

Lemma A.6. *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 A.5, 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 A.5, 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 A.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 1. *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 A.6, 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 A.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 A.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 2. *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 1, 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 1, 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 2. *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 2, 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.

Lemma 3. *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 A.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 A.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 A.2, v and w are neighbors on $DT(S)$ or u and w are neighbors on $DT(S)$.

Theorem 4. *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 A.4. If w is a neighbor of v on $DT(S)$, w is also a neighbor of v on $DT(S')$ by Lemma A.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 3, 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.

REFERENCES

[1] P. Bose and P. Morin, "Online routing in triangulations," *SIAM Journal on Computing*, vol. 33, no. 4, pp. 937–951, 2004.
[2] B. Delaunay, "Sur la sphère vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, pp. 793–800, 1934.
[3] H. Edelsbrunner, "Incremental topological flipping works for regular triangulations," *Algorithmica*, vol. 15, no. 3, pp. 223–241, 1996.
[4] P. M. Gruber and J. M. Wills, *Handbook of convex geometry*. North-Holland, 1993.
[5] D.-Y. Lee, E. K. Chung, and S. S. Lam, "A radius geocast routing protocol," in *Proc. of IEEE International Conference on High Performance Computing and Communications*, Dalian, China, September 2008.
[6] D.-Y. Lee and S. S. Lam, "Protocol design for dynamic Delaunay triangulation," The Univ. of Texas at Austin, Dept. of Computer Sciences, Tech. Rep. TR-06-48, October 2006.
[7] ——, "Protocol design for dynamic Delaunay triangulation," in *Proc. of IEEE International Conference on Distributed Computing Systems*, Toronto, Ontario, Canada, June 2007.
[8] J. N. Liebeherr, M. Nahas, and W. Si, "Application-layer multicasting with Delaunay triangulation overlays," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1472–1488, 2002.
[9] T. S. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proceedings of IEEE Infocom*, New York City, New York, USA, June 2002.
[10] 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*, Kyoto, Japan, January 2005, pp. 75–82.
[11] 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.
[12] G. Simon, M. Steiner, and E. Biersack, "Distributed dynamic Delaunay triangulation in d-dimensional spaces," Institut Eurecom, Tech. Rep., August 2005.
[13] 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.
[14] G. Voronoï, "Nouvelles applications des paramètres continus à la théorie des formes quadratiques," *J. Reine Angew. Math*, vol. 134, pp. 198–287, 1908.
[15] D. F. Watson, "Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes," *The Computer Journal*, vol. 24, no. 2, pp. 167–172, 1981.

Corrigenda

Dong-Young Lee and Simon S. Lam, "Efficient and Accurate Protocols for Distributed Delaunay Triangulation under Churn," in *Proceedings of IEEE ICNP*, October 2008.

1. The pseudocode of node u when it has received a NEIGHBOR_SET REQUEST from w is the following instead of the pseudocode shown in Figure 3 for the join protocol and in Figure 6 for the maintenance protocol.

   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 in $DT(C_u)$
   end if
   ${N_w}^u \leftarrow \{e \mid e$ is a neighbor of w in DT(C_u)$\}$
   Send(w, NEIGHBOR_SET_REPLY(${N_w}^u$))

2. The proof of Theorem 2 requires the following unstated assumption: The joining node n is inside the convex hull of the set S of existing nodes. If this assumption is not satisfied, then the joining node should send neighbor-set requests to all new neighbors it discovers instead of just one neighbor in each simplex including n in DT(C_n). That is, the join protocol in technical report TR-06-48 (Department of Computer Science, The University of Texas at Austin, October 2006) should be used.