GeoPeer: A Location-Aware Peer-to-Peer System

Filipe Araújo
Luís Rodrigues

DI–FCUL TR–03–31

December 19, 2003

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

GeoPeer: A Location-Aware Peer-to-Peer System

Filipe ARAÚJO
University of Lisbon
filipius@di.fc.ul.pt

Luís RODRIGUES
University of Lisbon
ler@di.fc.ul.pt

***Abstract*— This paper presents a novel peer-to-peer system that is particularly well suited to support context-aware computing. The system, called GeoPeer, combines the advantages of general purpose peer-to-peer systems with the suitability of geographical routing for supporting location-constrained queries and information dissemination. To achieve this goal, GeoPeer incorporates sophisticated mechanisms to establish long-range contacts that permit to achieve a small network diameter. These mechanisms take explicitly into account the unbalanced distribution of nodes in the geographical space.**

I. INTRODUCTION

The importance of context-aware services has grown significantly in the last decade. Context-aware computing focus on enriching applications with contextual information, like position, user activity, nearby people and devices, time of day or weather conditions [1]. In this paper, we are particularly interested in supporting location-aware services, as location is one of the most important parameters that can be extracted from context information.

Examples of location-aware services include querying for specific resources available in a geographic area (for instance, looking for a restaurant or a hospital in a given neighborhood), reading and integrating information collected by sensor nodes in a given region (for security purposes or environmental monitoring), and disseminating notifications to all nodes in a given region (to multicast warnings about natural or human-induced hazards, such as floods, chemical leaks, etc). Several other example applications of context-aware computing are described in [1].

A significant body of work exists on services and infrastructures for supporting location-aware mobile applications [2] but also on location-aware sensor-network services [3]. Most of these architectures rely on the existence of stationary nodes connected to the wired network infrastructure. This paper is mainly concerned with the scalability of the network of stationary nodes that provide support to very large-scale location-aware services (possibly, in cooperation with mobile nodes and wireless sensors). To the best of our knowledge, the scalability, decentralization, and dynamic aspects of the stationary infrastructure that supports location-aware computing have been overlooked in the literature.

As we discuss in the related work section, existing peer-to-peer systems such as Pastry [4], Tapestry [5], Chord [6], D2B [7], Koorde [8] or Viceroy [9], do not own the characteristics required to support location-aware services. Systems such as CAN [10], TOPLUS [11], eCAN [12], and the Delaunay triangulation proposed by Liebeherr et al. [13], are closer in spirit to GeoPeer but, as we will discuss later, they also lack features which are essential to support location-aware services in an efficient manner.

To address the concerns above, we propose GeoPeer, a novel peer-to-peer architecture that works as an overlay network on top of IP. Nodes of GeoPeer arrange themselves to form a Delaunay triangulation augmented with long range contacts to achieve small network diameter. Motivated by the works of Kleinberg [14] and Barrière et al. [15] (that try to model small-world networks) and the work of Xu and Zhang [12] (that use long range contacts to extend CAN [10]), we propose and compare a number of light-weight mechanisms for managing these long range contacts. Moreover, some of these mechanisms try to overcome an unbalanced use of identification space, which is of crucial importance in a location-aware system like GeoPeer.

Unlike most other systems, the use of geographical location is inherent to GeoPeer. Therefore, GeoPeer owns a number of interesting properties: it is capable of providing location-awareness in fundamental operations performed by applications, such as reads, writes or queries. For instance, inheriting from the techniques used by Liebeherr et al. in [13], we can augment multicast messages or flooded queries with scope information to limit their range, e.g., when raising an alarm after some accident. Queries containing significant regional information can also benefit from our system: the collection of information may be performed by a local proxy on behalf of the client that may be located far away from the region of interest to the query. For instance, someone in Lisbon may be searching for restaurant information in New York: instead of making all the replies traverse the Atlantic, a proxy in New York could aggregate all the replies and send back a single message to the originator of the query. In this paper, we also discuss additional techniques that can be supported by our system, like *geographical pings*, which are capable of determining the IP address of a node located in some particular geographical area. These techniques aim to create IP tunnels that serve as additional long-range contacts of use to specific applications.

In summary, the paper has two main contributions:

- it proposes a new peer-to-peer system that is particularly well-suited to support applications requiring location-aware operations, e.g., queries or broadcasts;
- it proposes and evaluates three light-weight but effective schemes to manage long-range contacts that aim to achieve small network diameter.

The remainder of the paper is organized as follows: Sec-

tion II overviews previous work. The GeoPeer architecture is described in Section III and the details of its long range contacts mechanisms are described and evaluated, respectively in Sections IV and V. Section VI concludes the paper.

II. Related Work

There is a significantly wide body of research which is relevant to the GeoPeer architecture, including work on context-aware computing [16], work on wireless *ad hoc* networks, namely location-aware routing schemes [17] and Delaunay triangulations [18], [19], and work on peer-to-peer systems. Given the nature of our contributions, we limit ourselves in the following paragraphs to the discussion of previous work on peer-to-peer systems and on its suitability to support location-aware services.

In recent years many peer-to-peer systems providing distributed hash tables holding (key, value) pairs were proposed [4]–[12]. Typically, these systems are characterized by a number of features, like network diameter, node degree or node congestion. For constant node degree, the best that can be done is $O(\log n)$ network diameter, while for $O(\log n)$ node degree, the network diameter can be reduced to $O((\log n)/(\log \log n))$ [8].

Systems such as Pastry [4], Tapestry [5], Chord [6], D2B [7], Koorde [8] or Viceroy [9] are not eligible to efficiently support location-aware services: they all use unidimensional random addresses that cannot directly represent a concrete physical location. Furthermore, the use of geographical locations as node identifiers would break the uniform distribution of the name space on which these systems rely to offer properties such as limited network diameter.

On the other hand, systems like CAN [10], TOPLUS [11] or the Delaunay triangulation proposed by Liebeherr et al. [13] could, in principle, be adapted to meet our goals. However, such adaptations would require significant changes to the original algorithms. CAN would have to be converted to use real geographical positions instead of virtual ones, and that would, as with the previous systems, break identification and load balancing. TOPLUS would be even harder to adapt, because TOPLUS peers are organized into groups of IP addresses and therefore some additional mechanism to translate IP addresses into geographical locations would be required. Additionally, both CAN and Delaunay triangulations, have large network diameter, which is an important drawback. This inconvenience can be mitigated with the use of long range contacts. This approach has been followed in the design of eCAN [12], which extends the basic CAN architecture with a complex Long Range Contacts (LRCs) management scheme (in eCAN LRCs are called "expressways"). However, the basic CAN is not as efficient as Delaunay triangulations in the support of multicast, because the average node degree is smaller. Moreover, eCAN embodies a number of sophisticated mechanisms, not strictly required for location-aware services, that introduce a significant processing overhead when compared with our LRC schemes.

Instead of performing a long series of adaptations to previous works to fully support in an efficient manner location-aware services, we have opted to design a new peer-to-peer architecture, called GeoPeer, that captures the positive aspects of each of the previous systems for the scalable support of context-aware computing.

III. Architecture of GeoPeer

A. Overview

In GeoPeer nodes self-organize into a planar Delaunay triangulation [20][1] augmented with carefully selected long range contacts (LRCs) to significantly reduce network diameter. A graph based on a Delaunay triangulation has the following desirable characteristics: i) expected $O(1)$ node degree, ii) good routing performance and iii) simple distributed construction. In GeoPeer, the identification of a node corresponds to its physical location. The combination of these features results in a peer-to-peer system with the following unique advantages:

- by creating a mesh of nodes identified by their physical location, support for applications that execute location-aware operations, such as queries or broadcasts, can be provided by very simple mechanisms;
- when compared to a bi-dimensional CAN-like network, the node degree in a Delaunay triangulation should be greater, but still $O(1)$ in expectation (around 6 instead of 4 in perfectly balanced cases) and, therefore, nearby routing should be improved;
- due to the LRCs that augment the Delaunay triangulation, GeoPeer has low network diameter. Moreover, our LRCs management schemes are elegant, but nevertheless, adaptive to unbalanced use of physical (and identification) space.

GeoPeer may also be applied to manage arbitrary objects by using a one-way hash function to compute keys from some relevant object attribute. Keys correspond to positions in space and, therefore, hashing some value representing an object yields the GeoPeer identification of that object. Since identification and position are equivalent, the hash function returns a pseudo-arbitrary position in space. Therefore, location information attributes may be used to carefully position resources in some application dependent way, e.g., by enforcing the use of a hash function that returns some node near the clients of a service.

B. Main Components

In the following paragraphs, the main components of the GeoPeer architecture are described in detail. These components are:

- An algorithm that creates and maintains the Delaunay triangulations. The algorithm includes the necessary mechanisms to preserve the triangulation when nodes join or leave the system.

[1]Alternatively, the Delaunay triangulation could be embedded in a sphere instead of a plane.

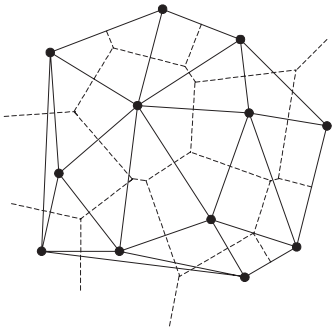

Fig. 1. Duality between Voronoi cells and Delaunay triangulation

- An algorithm that ensures that any possible key is held by exactly one existing node. Note that, as usual in peer-to-peer systems, occupation of the name space by nodes is sparse and each node must be made responsible for a portion of the name space. In GeoPeer, that division of space is made using geometrical arguments.
- An algorithm that performs routing of messages in the overlay network. Being based on a Delaunay triangulation, GeoPeer may use a myriad of well-known location-aware routing algorithms (e.g., see [18], [21]). As we will see ahead, for simplicity and efficiency reasons we opted for the greedy routing algorithm.
- A set of mechanisms that establishes LRCs. The discussion of these mechanisms is postponed to Section IV.

C. Notation and Definitions

In the text we will use the following conventions: nodes and points will be represented by capital letters , e.g., A; edges are represented by the two nodes that define them, for instance, AB; a triangle defined by nodes A, B and C is represented as $\triangle ABC$; the circumcircle of $\triangle ABC$ is represented as $\bigcirc ABC$; an angle ($< \pi$) between line segments AB and AC defined at A is interchangeably represented as $\angle BAC$ or $\angle CAB$ — the vertex where the angle is measured stays in the middle, while the position of remaining vertices is arbitrary;

A triangulation of a node set V is called a *Delaunay triangulation* if the circumcircle of each of its triangles does not contain any node of V [18], [19]. The Voronoi cell is a dual concept of the Delaunay triangulation, defined as follows: the Voronoi cell of node P is the set of points in space that are closer to P than to any other node. If the Voronoi cells of two nodes share a common border, then a Delaunay edge exists between those two nodes. This relation is illustrated in Figure 1 where borders of Voronoi cells have dashed lines, while Delaunay edges have solid lines.

D. Creation and Maintenance of Delaunay Triangulations

To create and maintain the Delaunay triangulation, GeoPeer uses a scheme that is similar to [13]. Note that many constructions proposed for wireless *ad hoc* networks, such as [18], [22], are not applicable in this context, since they assume static settings for triangulation.

Messages: To create and maintain the Delaunay triangulations, nodes periodically exchange messages with their geographic neighbors. The five message types exchanged by the algorithm are:

- The BEACON message, used by a node to inform its neighbors that it is still actively participating in the overlay network.
- The JOIN message, used to add new nodes to the network.
- The FAILURE message, used to disseminate information about the failure or departure of a node.
- The TRIANGULATE message, used by a node to propose the setup of a Delaunay triangle with its neighbors.
- The BREAKLINKS message, used to reconfigure the network in response to new joins and leaves.

The purpose and function of each of these message types will be detailed in the following. The algorithm does not require channels to be perfect: all messages that are critical to the convergence are retransmitted periodically, if no appropriate reply is received.

Steps: The algorithm is decentralized, as it does not rely on any centralized component. It consists of three logical steps:

1) The *neighbor discovery* step. Node N initiates this step to enter the network. To join, node N must use some out-of-band mean to discover one node already participating in the network, say P. P will then forward a special JOIN message on behalf of N destined to N. Since N does not yet belong to the network, the JOIN message will be received by some node X. This node X will forward the JOIN message to all the Delaunay neighbors of N that X knows about and will also reply with another JOIN message to N with the list of those Delaunay neighbors.
2) The *neighbor maintenance* step. In this step, nodes that belong to the same triangle periodically exchange BEACON messages to inform their neighbors that they are alive and actively participating in the network.
3) The Delaunay *triangulation* step. This is naturally, the most complex step of the algorithm.
   Based on the information collected in the previous steps, each node P computes a Delaunay triangulation using its own local knowledge. As a result, P may find out that there should exist a Delaunay triangle $\triangle PN_1N_2$, between P, N_1 and N_2. In this case, for convenience of exposition, we say that the predicate *Delaunay*$\triangle_P(N_1, N_2)$ is true at P.
   When *Delaunay*$\triangle_P(N_1, N_2)$ holds at P, P broadcasts a TRIANGULATE $\triangle PN_1N_2$ message to both N_1 and N_2. When P receives a TRIANGULATE $\triangle PN_1N_2$ from N_1, if *Delaunay*$\triangle_P(N_1, N_2)$ holds, then P replies to N_1 with another TRIANGULATE message, otherwise, P replies with a BREAKLINKS message including all nodes that it believes should triangulate with N_1.
   Therefore, if all neighbors agree on the triangulation, they will exchange a consistent set of TRIANGULATE messages and the corresponding Delaunay triangles are

set-up. Otherwise, they update their local information using the contents of the BREAKLINKS message and re-execute the local computation. Note that if there is some node inside $\bigcirc PN_1N_2$, the predicate *Delaunay*$\triangle_P(N_1, N_2)$ is immediately switched to false. A very simple way of checking this condition was presented by Sibson in [23].

Dynamic Aspects of the Algorithm: As noted before, to cope with a dynamic topology, the algorithm must take into account the following aspects: *i)* the failure of nodes; *ii)* the emergence of new nodes and, as a consequence, the possibility of nodes having a different view of the network topology.

Node failures and departures (to simplify the presentation, we do not distinguish these two events, however departures are handled in a more gracious way to allow for redistribution of keys) are detected through the absence of BEACON messages from that node. When some neighbor of F detects that node F failed, it recomputes the Delaunay triangulation and sends a FAILURE message to all its Delaunay neighbors. All nodes that are neighbors of F should resend the FAILURE messages of F. This ensures that all Delaunay neighbors of F become aware of its failure. Since network is asynchronous, nodes must store information about the failure of F. Therefore, FAILURE and BREAKLINKS messages include a (possibly empty in the case of a BREAKLINKS message) list of nodes that are known to be failed. If after a TRIANGULATE message from P, N replies with a BREAKLINKS message, with indication of some node F, that P knows to be failed, P sends a FAILURE message and later retries the triangulation.

It is also possible for nodes to enter the graph at any instant. Assume that P becomes aware of the presence of some new node Q and, as a result of recalculating the Delaunay triangulation, some triangles, *Delaunay*$\triangle_P(N_1, N_2)$ commute from true to false. In such case P sends a BREAKLINKS message to the vertices of those triangles. If *Delaunay*$\triangle_P(Q, N_1)$ is true for some node N_1, P will again send TRIANGULATE messages as described before.

Optimizations: For clarity of exposition, we deferred discussion of the following issue: BREAKLINKS and FAILURE messages should not carry indefinitely information about all nodes that failed in the past, as this could become a considerable overhead. Therefore, N only resends information that F failed to some peer node A until A acknowledges. However, even if all the Delaunay neighbors of N are aware of F's failure, it is still possible for some other node P, that was previously not a Delaunay neighbor of A, to try the triangulation $\triangle APF$. This would require N to store information of F forever. This problem is overcome in practice by discarding information of F after the expiration of a timeout.

Four or More Co-circular Nodes: If four or more nodes are co-circular and there is not any other node inside that circle, Delaunay triangulation is no longer unique. This represents a challenge to the algorithm, because, even if provided with the same information, nodes may not agree with each other on the correct triangulation. This problem is solved as follows. If node C_1 is co-circular with $n \geq 3$ other nodes $C_2, \ldots, C_n$ it considers all these neighbors as its Delaunay neighbors. This ensures that all co-circular nodes, C_i, become aware of each other's failures. All we have to do is to ensure that nodes use the same algorithm to arbitrarily triangulate inside the circle. Whenever three co-circular nodes agree on some triangle $C_iC_jC_k$, they will exchange TRIANGULATE messages among themselves. If a node disagrees on some triangle $C_iC_jC_k$ it sends a BREAKLINKS message to update information of the peer, where it includes all the co-circular nodes that it knows. Theorem A.1, presented in the Appendix, shows that the triangulation between co-circular nodes cannot have overlapping triangles. Therefore, since information increases at nodes, triangulation of co-circular nodes terminates and is correct.

E. Space Division

For each point in space there is one and only one responsible GeoPeer node. Therefore, the region of responsibility of each node defines a division of the space into non-overlapping adjacent areas. This division is based on the Delaunay triangulation used to support routing and it ensures that a node can only be responsible for a point which is inside of the Delaunay triangle in which it participates.

Unfortunately, a simple division in Voronoi cells does not satisfy the previous constraint, because Voronoi cells may cross triangle borders and this would make nodes responsible for points outside their triangles as depicted in Figure 2a. Therefore, the following algorithm is used to divide the space in GeoPeer: nodes must determine the circumcircle and the perpendicular bisectors for each of its Delaunay triangles, as depicted in Figure 2b. In "well behaved" triangles, where the point O lies inside the triangles, division of the space is straightforward and is done according to the figure. Areas A_A, A_B and A_C cover the entire triangle and define the set of points that are, respectively, closer to A, B and C. If the point O lies outside the triangle, such a division is still possible. However, in this case, two of the areas will not share a common border. Trivial tie-breaking mechanisms, not involving any communication, are used to define to which node belong the points in the perpendicular bisectors and points dividing two different triangles.

A problem occurs near the borders of the plane, where no further triangulations are possible. In this case we use a proximity criterion to determine the areas of responsibility of the nodes. Figure 2c depicts this division.

F. Routing

Routing between GeoPeer nodes (not considering LRCs) resumes to routing in a Delaunay triangulation. A myriad of routing algorithms that are typically used in the context of wireless *ad hoc* networks can be used in a Delaunay triangulation, namely compass, randomized compass, greedy or Voronoi [17], [21]. Figure 3 depicts an example of routing from source node S to destination node D. Greedy algorithm will select the neighbor of S closest to D, which is

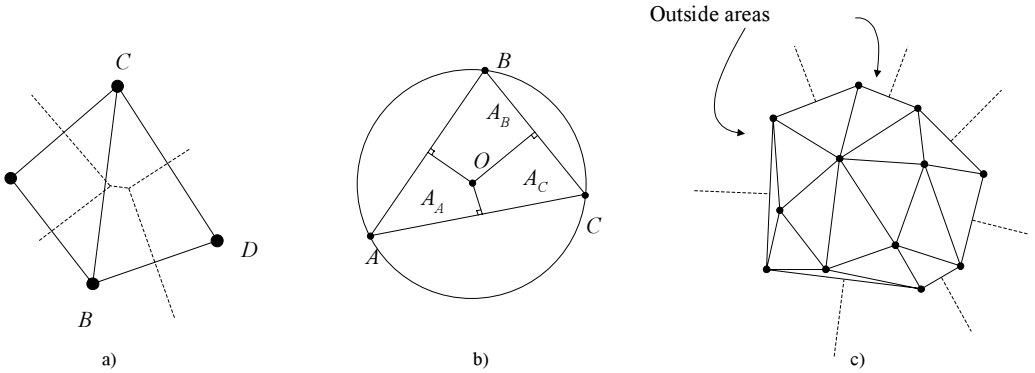

Fig. 2. a) Voronoi cells (dashed lines) cross triangle boundaries, b) Circumcircle, c) Outside areas

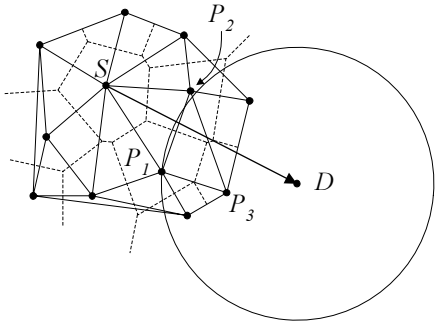

Fig. 3. Routing algorithms

P_1, compass algorithm will select P_2, because $\angle P_2SD$ is the smallest angle possible, defined by SD (in the figure, $\angle P_2SD \approxeq 23^o$, while $\angle P_1SD \approxeq 30^o$), randomized compass will select either P_1 or P_2 and Voronoi algorithm will select P_2 first and P_3 afterward. Note that these routing algorithms also work if start and end points do not correspond to nodes, but to arbitrary points in space, because these points must have some responsible node that will either send or receive a message on their behalf. In GeoPeer, we have opted to use the greedy algorithm to route messages. The following paragraph explains the rationale for this decision.

Bose et al. present in [21] a number of relevant results on routing in triangulations: They proved that greedy algorithm is not defeated by any Delaunay triangulation; compass algorithm is not defeated by any regular triangulation[2]; and randomized compass is not defeated by any triangulation; Voronoi routing algorithm is also not defeated by any Delaunay triangulation. Therefore, all these algorithms work in a Delaunay triangulation. Unfortunately, none of these algorithms is competitive. Informally, a routing scheme is competitive if for any pair of source and destination nodes the scheme always finds some path within a constant of the best possible path. In the same work [21] Bose et al. presented a competitive algorithm called *parallel Voronoi*. However, despite not being competitive in some particular cases, experimental results show that greedy and compass algorithms achieve very good results in general [18]. Additionally, parallel Voronoi algorithm consists of going back and forth along faces of triangles and does not seem very promising in practice. Therefore, we prefer the simpler not competitive, but efficient in practice, greedy algorithm.

[2]See [20] for a definition of regular triangulations. Delaunay triangulation is a particular case of a regular triangulation.

G. Applications of GeoPeer

GeoPeer may, as any other decentralized peer-to-peer system, be used to support any sort of application that benefits from a scalable implementation of a distributed hash table, such as, for instance, decentralized storage services [24]–[26]. However, some of the characteristics of GeoPeer, like location-awareness and uneven distribution of nodes, make it specially fit for the support of location-aware services. We now illustrate the benefits of the architecture by giving some examples of context-aware services that can be trivially implemented on top of GeoPeer.

- *Geographically-scoped multicast*. This service consists in disseminating a notification to all nodes located inside a given geographic region. This service can be used, for instance, to disseminate an alarm about some natural disaster such as a storm, flood or fire. The service can be easily implemented by routing a notification to the GeoPeer nodes responsible for the center of that area which will, in turn, initiate a scoped-broadcast of the notification, using the technique proposed in [13]. It should be noted that, with the exception of a bi-dimensional CAN (and variations like eCAN) no other peer-to-peer system would directly support this service. Furthermore, the use of Delaunay triangulations make GeoPeer more efficient than CAN or eCAN.
- *Geographically-scoped queries*. This service is the counterpart of the previous service. It is used to collect information from nodes located inside a given geographic region. This service can be used for environmental or security monitoring of geographical areas by connection the relevant sensors to the GeoPeer nodes. It can also be used to collect more mundane information, such as the location of cinemas or bars in the vicinity of a given location. The service works by having the node

responsible for the center of the region of interest acting as an ambassador of the client. This node can efficiently query all nodes in a given diameter, collect all the replies, and send the consolidated information back to the client in a single message. If needed, the proxy can also perform data fusion services (such as computing averages, selecting the lowest or highest values, etc).

- *Other location-aware services*. GeoPeer also opens new less obvious possibilities for applications that need to determine location of critical resources, like a rendezvous point in a core-based multicast tree [27] or in publish-subscribe applications [28], [29]. However a complete exploration and evaluation of such solutions is beyond the scope of this paper.

H. Challenges

If routing in GeoPeer was based exclusively on the use of a greedy algorithm, the resulting network operation would be inefficient due to the large network diameter of the underlying Delaunay triangulation; namely, communication in the GeoPeer network would have a very large latency. Therefore, we need to enhance GeoPeer with mechanisms capable of reducing the network diameter.

A simple solution to the previous problem consists in using a method that we have dubbed *geographical ping*. This method works as follows. When some node N wants to repeatedly send a message to some point of GeoPeer, say P, it sends a *ping* to P that should be replied by the node responsible for P, say M. Then N may set up and periodically refresh a tunnel to M directly using IP, thus bypassing a number of GeoPeer nodes. Of course, this solution only works if the number of messages exchanged is worthwhile the initial ping effort.

Another more versatile solution that we explore in this paper is the use of long range contacts (LRCs). Clearly, LRCs play a critical role in GeoPeer as they can significantly reduce network diameter, thus improving performance. In the next section we propose and compare a number of mechanisms that manage LRCs.

IV. Long Range Contacts (LRCs)

The goal of establishing LRCs is to reduce the network diameter, that is typically too high both in networks like CAN and in networks based on Delaunay triangulations. By carefully selecting LRCs, it is possible to effectively reduce network diameter, while, at the same time, maintaining a limited expected node degree (constant or at worst logarithmic with respect to the maximum number of nodes). Hopefully, it should be possible to establish and maintain LRCs in a simple and expedite way, to reduce the overhead caused by these mechanisms. Greedy routing using LRC is a simple extension of the basic Delaunay triangulation: all Delaunay neighbors and all LRC are eligible to be the next hop. The neighbor or LRC closest to destination is chosen.

GeoPeer improves previous work [12] by supporting three new schemes, plus a scheme derived from eCAN, to establish and maintain LRCs that are suitable to be used in networks with heterogenous node density and in networks based on geographical identifiers. As it will be seen, our LRCs mechanisms obtain highly efficient results with considerable less computational overhead than previous work. The LRCs schemes supported by GeoPeer, which are described in the next paragraphs, are: a *hop level* mechanism, a *hit count balancing* mechanism, a *small-world* mechanism and a *eCAN-like* mechanism (for comparison).

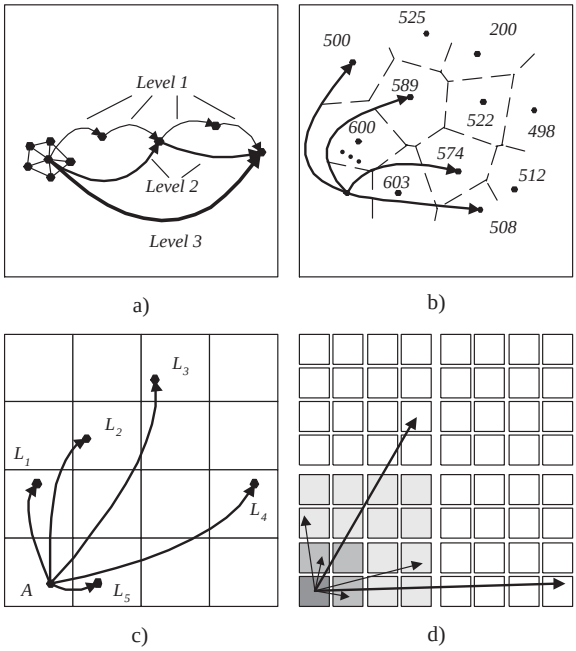

Fig. 4. LRC mechanisms: a) Hop level, b) Hit count, c) Small-world, d) eCAN-like

A. Hop Level Mechanism

In the *hop level* mechanism, nodes try to avoid doing more than a predefined number of hops, say b hops. For instance, if some node A_1 is forwarding a message from N and if A_1 is the b-th hop of the message, N should create a LRC to A_1. To do this, A_1 sends a message prompting N to create a LRC to itself. The process is repeated, this time starting at A_1: if after b hops, message reaches A_2, A_1 will create a LRC to A_2, and so on. Now suppose that the following series of b LRC is created: NA_1, A_1A_2, ..., $A_{b-1}A_b$. In this case, a new LRC from N to A_b would be created. This new LRC would be one level above of the others, i.e., while LRC NA_1, A_1A2, ..., $A_{b-1}A_b$ are of level 1, LRC NA_b is of level 2. The creation of LRC, possibly with increasing levels, is recursively repeated until message reaches its destination. Figure 4a illustrates the main idea. Hence, no more than $b-1$ hops are allowed in each of the levels, without raising the creation of a LRC in the next upper level. The creation of LRC is done according to the following property: the level of each LRC is determined by powers of base b, such that a LRC of level l jumps over b^l hops. Last possible level L is selected in a way such that b^{L+1} is the smallest power beyond

the greatest possible number of hops existing in the space (i.e., greater than the number of existing identifications). In practice no more than a small number of levels is used. For instance, in our simulations with 50000 nodes and with $b = 4$, no more than 5 levels were ever used.

To implement hop level LRC scheme, messages must carry two variables per level: the number of hops and the first node of that level. Initially, all values are clear. The jumps of level l and the creation of a LRC of level l trigger the following actions: (i) reset the message hop-count of all inferior levels $< l$, (ii) set the forwarding node as the originator of the message for all inferior levels and (iii) increment the hop-count of level l (this step may raise the creation of a new LRC). For instance, if $b = 4$ and node N is the 4-th hop of level 3 (order of 4^3 hops) and node S is the origin of the first level 3 hop, S creates a LRC of level 4 to N. Further, to send the message, N resets the number of hops for levels 0, 1, 2 and 3, marks itself as the origin of the paths of those levels and increments the number of hops for level 4.

Finally, we need to limit the number of LRC per node, because the previously described mechanism would lead to an unbounded increase of LRCs. Moreover, since the system is dynamic, some nodes that are LRCs may fail and new nodes may emerge. Therefore, nodes continuously refresh their LRC lists, by adding new contacts as they are identified and discarding least recently used contacts, whenever needed.

B. Hit Count Balancing Mechanism

In the *Hit Count Balancing* method, nodes start by arbitrarily selecting a predetermined number of LRCs in a way that evenly spreads those LRCs by the existing space. LRCs have hit counters, initially set to 0, that serve to count the number of times they were referenced. For instance, when routing algorithm uses one LRC, say to node L_1, the hit counter of LRC L_1 is incremented. Periodically, nodes will check which LRCs are used more often and which ones are rarely used. LRCs with many references are split, while LRC with few references are discarded. The goal is to keep the hit counters of all LRCs balanced, to achieve the best possible utilization of these LRCs. In fact, most often used LRCs should correspond to densely populated zones, or more precisely to zones with many contacts, while least often used LRCs should correspond to zones with fewer nodes or contacts. To ensure that hit counters do not diverge their value is periodically halved[3].

To determine which LRCs are to be eliminated and which ones are to be split we use the following algorithm. First, each node classifies its LRCs according to the values of the corresponding hit counters. This classification uses as a reference value the average of all LRCs' hit counters and considers four different levels: values above the twice the average are classified as "veryhigh" while values above the average but below this threshold are considered "high"; similarly, values below half the average are classified "verylow" while values below the average but above this threshold are deemed "low".

[3]Since $\sum_{i=0}^{\infty} 1/2^i = 2$, counters will not diverge.

The algorithm tries to split all the LRCs that are "veryhigh" as long as there are enough "low" LRCs, which gives $\min\{veryhigh, low\}$ top LRCs to be split and the same number of lowest hit counter LRCs to be eliminated. The symmetric procedure is applied to "verylow" LRCs: we eliminate them as long as there are enough "high" LRCs to be split, which gives $\min\{high, verylow\}$ LRCs. Hence, the total number of LRCs to be split from the top of the list and the total number of LRCs to be eliminated from the bottom of the list is given by the following formula: $\max\{\min\{veryhigh, low\}, \min\{high, verylow\}\}$.

Determining the location of a newly created LRC obtained after splitting is also an interesting problem that we solve in a simple but efficient way. We start by dividing the space in Voronoi cells of the existing LRC. Then we split LRC L_1 as follows: the original pointer L_1 remains in the same place; the newly created pointer L_n goes to one of the vertexes of the Voronoi cell of L_1, say a vertex defined by the borders of the Voronoi cells of L_1, L_x and L_y. This algorithm is illustrated in Figure 4b. Note that this vertex *maximizes the minimum distance* of L_n to L_1, L_x and L_y. From the several possible vertexes of the Voronoi cell of L_1 we select one that is simultaneously (i) distant from L_1 and (ii) has LRC L_x and L_y with high hit counters. This process is not too different from choosing a location for opening a new branch of the same company, e.g., a restaurant: the restaurant should be located as far as possible from the others in a densely populated zone. Finally, to conclude the splitting process, hit counter of L_1 is evenly divided by L_1 and L_n.

C. Small-World Mechanism

Small-worlds [14], [30] have some interesting properties, like constant node degree and squared logarithmic network diameter. In the *small-world* mechanism, nodes have some previously set limit for the number of LRCs, say q, that are selected as follows. First, the entire space is divided into n squares (assuming that the space is itself a square), whose centers are the points eligible to be the LRC for that square. Since $q < n$ we need to use a random process that outcomes centers of squares as LRC. Hence, we make q trials to determine the LRCs. To determine the probability of each one of the possible outcomes we adapt the utilization of the r-harmonic probability distribution of Equation 1, $p_r(U, V)$, used in the work of Kleinberg [14]. Here, U is the node selecting the LRC, while V is the center of a square. Note that, the probability of selecting center V to be a LRC of U decreases exponentially with the distance of U to V. Note also that the term $\sum_{W \neq U} d(U, W)^{-r}$ is used to normalize the probability distribution $p_r(U, V)$. Drawing from the conclusions of Kleinberg we will set r to 2. Barrière et al. [15] also use this distribution, but to build a small-world model by augmenting a ring.

$$p_r(U, V) = \frac{d(U, V)^{-r}}{\sum_{W \neq U} d(U, W)^{-r}} \quad (1)$$

Note that unlike previous work we do not assume a well-behaved network, like a mesh or a ring, where nodes have access to global information, e.g. the existing number of nodes, because it would not be reasonable in a peer-to-peer system. This kind of knowledge allowed other authors to use the harmonic distribution directly on peer nodes. Since we cannot do that, we circumvented the problem by dividing the space into squares. Of course, this raises an additional problem: the centers of the squares will probably not correspond to any existing node. Therefore, the node responsible for the center point serves as the *de facto* LRC. Note that this LRC management scheme does not adapt to unbalanced use of identification space. Figure 4c illustrates the small-world scheme.

D. eCAN-like Mechanism

To enable a comparison with eCAN, we use a mechanism that builds LRC in a way that closely resembles the expressways of eCAN. We must emphasize that this mechanism is an extremely simplified version of the complete eCAN solution, that only captures the fundamental impact of the expressway mechanism in routing, and does not attempt to reproduce other features of eCAN (such as the mechanisms that provide support for complex interaction schemes like publish/subscribe). In spite of these simplifications, we believe that our implementation of expressways mimics the eCAN LRC mechanism with enough accuracy to allow a fair comparison.

Hence, the idea is to make a first level division of the entire space in four big squares. Each node keeps LRC to two of these four squares: one to some node that is in the square above/below, the other to some node that is in the square at right/left. Then, the four big squares are further divided in other four smaller squares. This time, some of the squares in the middle may have a total number of four LRC (above, below, right and left). This process is repeated for as many levels as wanted. Figure 4d illustrates the eCAN-like LRC scheme. In our context we set the number of LRC to some predetermined level and stop creating new LRC as well as new subdivisions as soon as that number is reached. Since we select random points inside the squares we must proceed in a way similar to the small-world mechanism. Hence, the effective long range contact is the node responsible for the point that was randomly selected.

V. EVALUATION

In this section we experimentally evaluate GeoPeer through simulation. More precisely, we compare the average number of hops needed by the greedy routing algorithm under the different LRC schemes (pure Delaunay triangulation, hop level LRC, hit count LRC and small-world LRC) to route a message between two arbitrary nodes, under the following variables conditions: (i) 100, 500, 1000, 5000, 10000 and 50000 nodes; (ii) 10, 20 and 30 contacts; and (iii) balanced distribution of nodes, vs. unbalanced distribution. In the unbalanced distribution, nodes are more likely to be near the four corners of the square than anywhere else. We experimented more than 120 test scenarios. To let adaptive LRC schemes converge we never included in the figures more than the final 5% of the paths in each test scenario. Due to space constraints, it is impossible to present in the paper the results from all the test scenarios identified above. Therefore, we present only the most representative and omit those that show similar results.

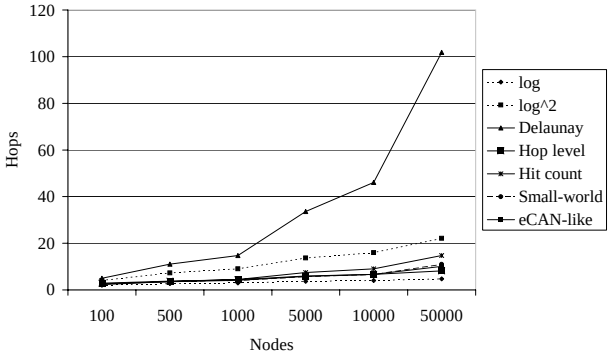

Fig. 5. Number of hops with 10 LRC - balanced scenario

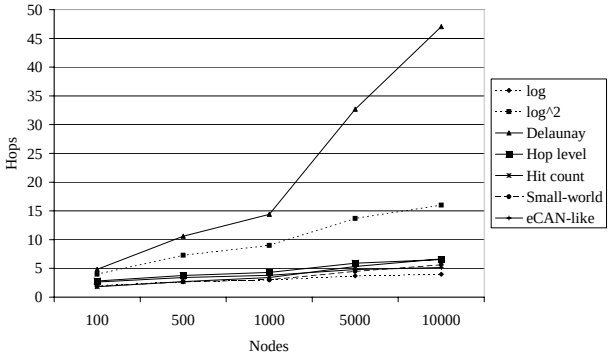

Fig. 6. Number of hops with 30 LRC - balanced scenario

Figures 5 and 6 present the number of hops for each of the routing schemes, including the $\log(n)$ and $\log^2(n)$ functions, for a balanced distribution of nodes. Figures 7 and 8 present the same metrics, but for a more realistic network, with an unbalanced distribution of nodes. Then, in Figure 9, we compare hit count, small-world schemes and eCAN-like against themselves in the balanced distribution, when the number of LRC is limited to 10 and 30. For the sake of readability, figures for 20 LRCs are omitted, because they fall between figures for 10 and 30 LRCs. We did not depict values for the hop level scheme in this figure because it is insensitive to this configuration parameter: this is due to the fact that for the network sizes we tested, most nodes in this scheme never collect more than a few LRCs (typically less than 10). Finally, in Figures 10 and 11 we, respectively, compare the performance of the hop level LRC scheme in the balanced vs. unbalanced scenario and then the same figures for hit count, small-world and eCAN-like schemes, with 10 LRCs.

From the graphics we can draw the following conclusions:

- in our experiments, the number of hops in all LRC management schemes were between $\log(n)$ and $\log^2(n)$.

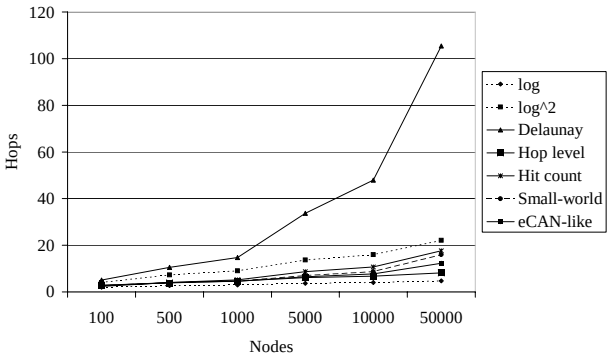

Fig. 7. Number of hops with 10 LRC - unbalanced scenario

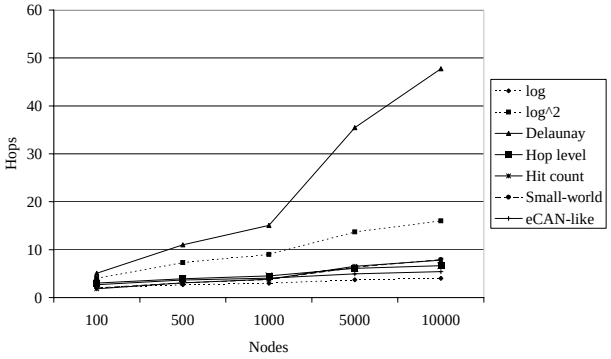

Fig. 8. Number of hops with 30 LRC - unbalanced scenario

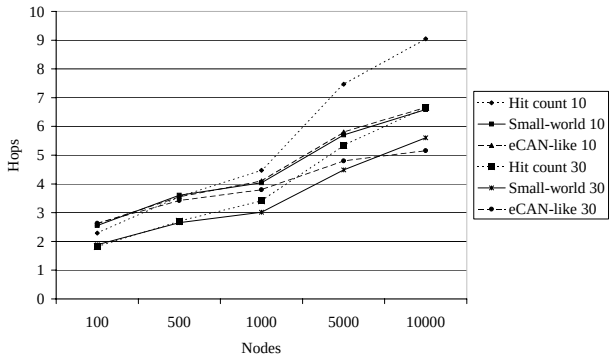

Fig. 9. Comparison of hit count, small-world and eCAN-like limited to 10 and 30 LRC

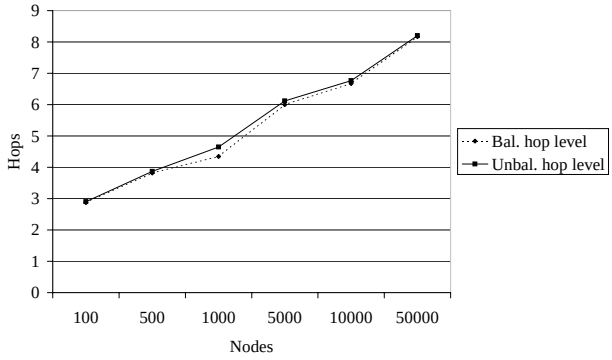

Fig. 10. Hop level LRC scheme in balanced vs. unbalanced distribution

Nevertheless, graphs show that, at least hit count and small-world are closer to $\log^2(n)$ when n grows, while the growth of hop level and eCAN-like is clearly more moderate;

- the number of LRCs per node did not emerge as an important factor, except for eCAN-like mechanism, that improves considerably with the increase in the number of LRC. Our interpretation of this fact is that performance of this mechanism is largely dependent of the size of the network. Either this size is accurately approximated, or, if the size of the network is underestimated, performance is likely to degrade;
- the distribution of the nodes in space matters. Clearly, it is essential to take into account the uneven distribution of nodes in space in peer-to-peer systems based on geographical location. From the experiments, the hop level scheme emerges as the less affected by an uneven distributions of nodes;
- of all the LRC management schemes we compared, hop level is the one that presents best results with a small number of LRC as the network size increases, while eCAN-like benefits more from a large number of LRC. Hence, hop level demonstrates to be quite immune to the selected number of LRC or to node distribution, thus precluding the need to any kind of *a priori* configuration. Additionally, hop level does not need any initial setup, because nodes establish LRCs on the fly. Therefore, while in static scenarios, where the node distribution in space in known beforehand, the eCAN-like scheme exhibits some advantages provided appropriate configuration is given to nodes, hop level is a preferable choice, if the environment is unknown.

As a final remark, these experimental results show the validity of our LRC solutions, in particular the use of the hop level mechanism. Unlike eCAN, that mixes in a single solution the support for LRCs with the support for complex interaction schemes such as publish/subscribe, the mechanisms proposed in this paper are light-weight, but, nevertheless, efficient. In particular, our novel hop level mechanism achieves excellent results even in scenarios with unbalanced distribution of nodes.

VI. Conclusions

This paper makes two major contributions: i) it defines a peer-to-peer architecture called GeoPeer, intended to support location-aware applications; and ii) it defines and compares three different algorithms to manage Long Range Contacts (LRCs). In spite of being studied in the context of GeoPeer, these LRC mechanisms are more general and can be applied to other systems as well.

GeoPeer is a peer-to-peer system that can be used to provide services to location-aware applications. These services include geographically-scoped multicasts or geographically-scoped queries, which can be used by applications willing

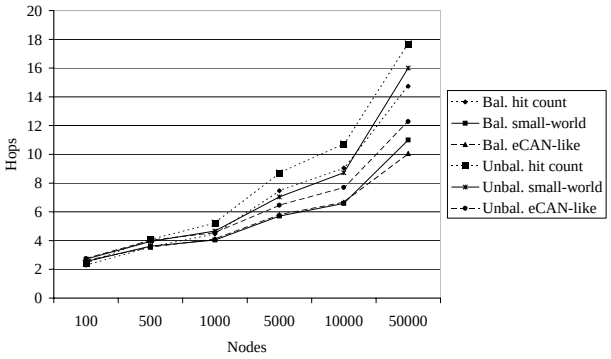

Fig. 11. Hit count, small-world and eCAN-like in balanced vs. unbalanced distribution

to spread some alarm within a limited region or applications searching for local facilities, like restaurants or cinemas, for instance. Additionally, location properties offered by GeoPeer can be explored by other more complex applications, like core-based multicast trees or publish-subscribe systems, to improve location of critical resources.

The fundamental core that supports location-aware routing in GeoPeer is a Delaunay triangulation of nodes. However, if exclusively based on a Delaunay triangulation, performance of GeoPeer would be seriously compromised by a very large network diameter. Therefore, we proposed and validated experimentally three light-weight but effective schemes to augment the Delaunay triangulation with LRCs. As can be drawn by these experimental results, the best LRC schemes achieve results between $\log(n)$ and $\log^2(n)$ even in the presence of unbalanced node distribution, a fundamental characteristic of location-aware peer-to-peer systems.

Acknowledgements

The authors are thankful to Patrick Eugster and Rachid Guerraoui for their comments on earlier versions of this paper.

REFERENCES

[1] G. Chen and D. Kotz, "A survey of context-aware mobile computing research," Dept. of Computer Science, Dartmouth College, Tech. Rep. TR2000-381, November 2000.

[2] H. Maass, "Location-aware mobile applications based on directory services," *Mobile Networks and Applications*, vol. 3, no. 2, pp. 157–173, 1998.

[3] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava, "Coverage problems in wireless ad-hoc sensor networks," in *INFOCOM*, 2001, pp. 1380–1387.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *Lecture Notes in Computer Science*, vol. 2218, pp. 329–350, 2001.

[5] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," UC Berkeley, Tech. Rep. UCB/CSD-01-1141, Apr. 2001.

[6] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *ACM SIGCOMM*, San Diego, August 2001.

[7] P. Fraigniaud and P. Gauron, "The content-addressable network D2B," LRI, Univ. Paris-Sud, France, Tech. Rep. 1349, Jan 2003.

[8] F. Kaashoek and D. R. Karger, "Koorde: A simple degree-optimal distributed hash table," 2003.

[9] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," in *Twenty-First ACM Symposium on Principles of Distributed Computing (PODC 2002)*, Monterey, California, July 2002.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *Conference on applications, technologies, architectures, and protocols for computer communications*. ACM Press, 2001, pp. 161–172.

[11] L. Garcés-Erice, K. Ross, E. Biersack, P. Felber, and G. Urvoy-Keller, "Topology-centric look-up service," in *COST264/ACM Fifth International Workshop on Networked Group Communications (NGC)*, Munich, Germany, 2003.

[12] Z. Xu and Z. Zhang, "Building low-maintenance expressways for p2p systems," HP, Tech. Rep. HPL-2002-41, 2002.

[13] J. Liebeherr, M. Nahas, and W. Si, "Application-layer multicasting with Delaunay triangulation overlays," University of Virginia, Department of Computer Science, Tech. Rep. CS-2001-26, 5 2001.

[14] J. Kleinberg, "The Small-World Phenomenon: An Algorithmic Perspective," in *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[15] L. Barrière, P. Fraigniaud, E. Kranakis, and D. Krizanc, "Efficient routing in networks with long range contacts (extended abstract)," in *15th Internation Conference on Distributed Computing*, ser. Lecture Notes in Computer Science, J. Welch, Ed., no. LNCS 2180. Lisbon, Portugal: Springer, October 2001.

[16] J. Hightower and G. Borriella, "Location systems for ubiquitous computing," *IEEE Computer*, vol. 34, no. 8, pp. 57–66, 2001.

[17] I. Stojmenovic, "Position-based routing in ad hoc networks," *IEEE Communications Magazine*, July 2002.

[18] X.-Y. Li, G. Calinescu, and P.-J. Wan, "Distributed construction of a planar spanner and routing for ad hoc wireless networks," in *The 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2002.

[19] L. Lan and H. Wen-Jing, "Localized delaunay triangulation for topological construction and routing on manets," in *2nd ACM Workshop on Principles of Mobile Computing (POMC'02)*, 2002.

[20] G. M. Ziegler, *Lectures on Polytopes*, ser. Graduate Texts in Mathematics. New York: Springer-Verlag, 1994, no. 154.

[21] P. Bose and P. Morin, "Online routing in triangulations," in *10th Annual Internation Symposium on Algorithms and Computation (ISAAC)*, 1999.

[22] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu, "Geometric spanners for routing in mobile networks," in *2nd ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 01)*, 2001.

[23] R. Sibson, "Locally equiangular triangulations," *The Computer Journal*, vol. 21, no. 3, pp. 243–245, 1977.

[24] J. Douceur and R. Wattenhofer, "Optimizing file availability in a secure serverless distributed file system," in *Proceedings of 20th IEEE SRDS*, 2001, pp. 4–13.

[25] P. Druschel and A. Rowstron, "Past: A large-scale, persistent peer-to-peer storage utility," in *HotOS VIII*, Schoss Elmau, Germany, May 2001.

[26] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-free global data storage," *IEEE Internet Computing*, vol. 5, no. 5, pp. 40–49, 2001.

[27] A. Ballardie, "Core based trees (cbt version 2) multicast routing," Request for Comments 2189, September 1997.

[28] P. Pietzuch and J. Bacon, "Hermes: A distributed event-based middleware architecture," in *22nd IEEE International Conference on Distributed Computing Systems Workshops (DEBS '02)*, 2002.

[29] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications (JSAC)*, 2002.

[30] D. Watts and S. Strogatz, "Collective dynamics of small-world networks," *Nature*, no. 393, pp. 440–442, June 1998.

Appendix

A. PROOFS

Theorem A.1 *Triangulation between co-circular nodes cannot have overlapping triangles.*

Proof: We assume that the remaining triangulation (outside the circle) is correct. This implies that nodes that

are consecutive in the circle line are Delaunay neighbors. If there are two overlapping triangles there are at least two crossing edges, say C_iC_j and C_mC_n. Now consider all the consecutive nodes between C_i and C_m in the circle line, $C_i = C_p, C_{p+1}, \ldots, C_{p+q} = C_m$. Consider without loss of generality that C_m is at the right of edge C_iC_j. Since C_{p+1} triangulates with C_i, triangles of C_{p+1} must also be at right of C_iC_j. Additionally, C_{p+2} triangulates with C_{p+1} and, therefore, triangles of C_{p+2} must also be at the right of the triangle defined by C_i and C_{p+1} and by majority of reason at the right of C_iC_j. Hence, by induction, C_mC_n is at the right of C_iC_j, which is impossible, and therefore, no overlapping is possible. ■