A Distributed Triangulation Algorithm for Wireless Sensor Networks on 2D and 3D Surface

Hongyu Zhou, Hongyi Wu, Su Xia, Miao Jin, and Ning Ding

***Abstract*—Triangulation serves as the basis for many geometry-based algorithms in wireless sensor networks. In this paper we propose a distributed algorithm that produces a triangulation for an arbitrary sensor network, with no constraints on communication model or granularity of the triangulation. We prove its correctness in 2D, and further extend it to sensor networks deployed on 3D open and closed surfaces. Our simulation results show that the proposed algorithms can tolerate distance measurement errors, and thus work well under practical sensor network settings and effectively promote the performance a range of applications that depend on triangulations.**

I. Introduction

A wireless sensor network can be represented by a graph, where a node corresponds to a sensor and an edge indicates the communication link between two sensors. A network graph under practical experiment settings or theoretical communication models[1] usually exhibits undesired randomness and intractability, calling for effective techniques that yield a well structured network subgraph to support target applications. For example, triangulation [2], [3] serves as the basis for many geometry-based routing [2], [4]–[6], localization [7], [8], coverage [9], segmentation [10], and data storage and processing [11] algorithms in wireless sensor networks.

In advanced geometry, a triangulation of a discrete set of points is a subdivision of the convex hull of the points into simplices such that any two simplices intersect in no more than one common face and the vertices of the subdividing simplices coincide with the points [12]. For a sensor network deployed on 2D or 3D surface, triangulation means removal of some edges in the network graph, yielding a subgraph that is a triangular subdivision. A network graph may have multiple triangulated subgraphs. Each of them is a maximal planar subdivision, where the addition of any edge results in a nonplanar graph [13]. A network graph is not triangulated in general. For example, Fig. 1(a) shows a 2D network graph, and its triangulation is given in Fig. 1(d). Similarly, Figs. 1(b) and 1(c) depict a 3D open surface network and a 3D closed surface network. Their triangulations are shown in Figs. 1(e) and 1(f), respectively. In addition, a triangulation can also be established based on a backbone structure as illustrated in Figs. 1(j)-1(l).

The salient properties of triangulated subgraphs support a wide range of 2D and 3D graphic tools that deliver theoretically sound and practically viable solutions. However, it is nontrivial to achieve distributed triangulation without location information. The best known solution is introduced in [2], [3], for 2D and 3D surface networks, respectively. They are based on the planarization algorithm proposed in [14]. While they have been employed to produce triangulation for several works [2], [4]–[6], [8], [11], they can only create a triangulated virtual backbone that is based on a set of landmarks and thus too coarse for many applications (see Figs. 8-10 for example). Moreover, as to be discussed in Sec. II, they do not ensure successful triangulation in polynomial time. They perform well under the Quasi-UDG model with $1 \geq \alpha \geq 1/\sqrt{2}$ and for sparsely selected landmarks only.

This research aims to develop efficient triangulation algorithms for wireless sensor networks. Our key contributions are summarized below:

- We propose a distributed algorithm that produces a triangulation for any arbitrary 2D sensor network, with no constraints on the communication model or the granularity of the triangulation (see Figs. 1(a), 1(d), and 1(g)-1(l) for examples).
- We prove the correctness of the algorithm in 2D networks.
- We further extend the algorithm to sensor networks deployed on 3D open and closed surfaces (as shown in Figs. 1(b)-1(c) and 1(e)-1(f)).
- Our simulation results show that the proposed algorithms can tolerate distance measurement errors up to 30% (as demonstrated in Figs. 1(m)-1(o)), and thus work well under practical sensor network settings and effectively promote the performance of a range of applications that depend on triangulations.

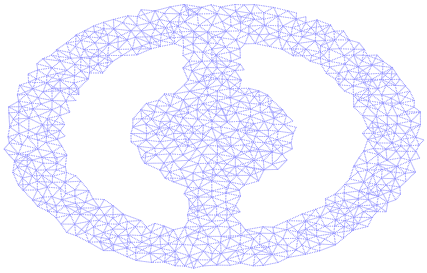
Sec. II of the paper discusses challenges and related work. Sec. III elaborates the proposed triangulation algorithm and proves its correctness. Sec. IV extends the algorithm to 3D surface networks. Sec. V presents the results of triangulation and its application. Finally, Sec. VI concludes the paper.

II. Challenges and Related Work

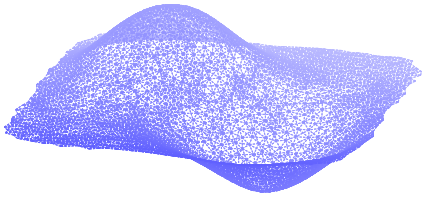
Based on the planarization algorithm proposed in [14], heuristic approaches have been introduced in [2], [3] for sensor networks deployed on 2D and 3D surface. The basic idea is to employ a distributed algorithm (e.g., [15]) to elect a

This work is supported in part by National Science Foundation under Award Number CNS-1018306. The authors are with the Center for Advanced Computer Studies, University of Louisiana at Lafayette, Lafayette, LA.

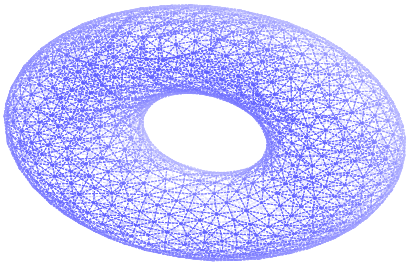
[1]Several communication models have been adopted for theoretic studies of sensor networks. For example, the *unit disk graph (UDG)* model assumes two nodes are connected if and only if their Euclidian distance is no greater than a unit. A more practical model, named *quasi unit disk graph (Quasi-UDG)*, sets a parameter $\alpha < 1$. Two nodes are connected if their distance is less than α, or disconnected if they are separated greater than one, or connected with a probability if their distance is between α and one. In another model based on log-normal shadowing channel [1], the received signal power is described by a Gaussian-distributed stochastic variable, and two nodes are connected if the received signal power is greater than a given threshold.



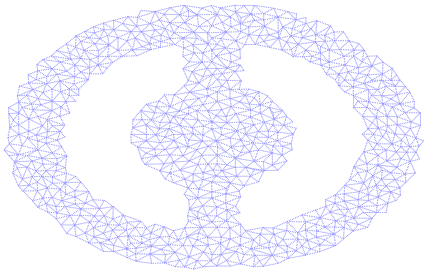
(a) A 2D network graph.



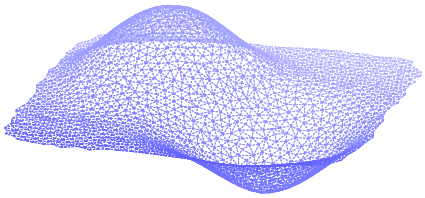
(b) A 3D open surface network graph.



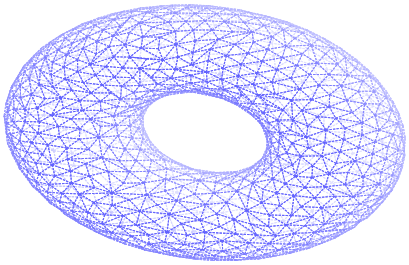
(c) A 3D closed surface network graph.



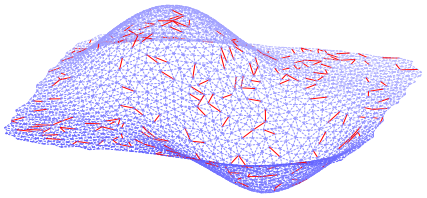
(d) Triangulation of Fig. 1(a).



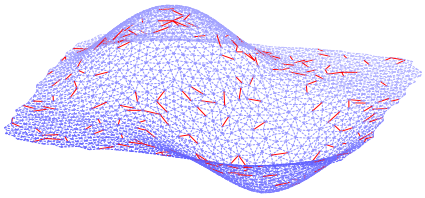
(e) Triangulation of Fig. 1(b).



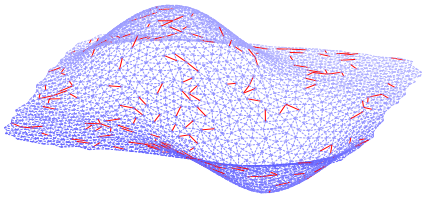
(f) Triangulation of Fig. 1(c).



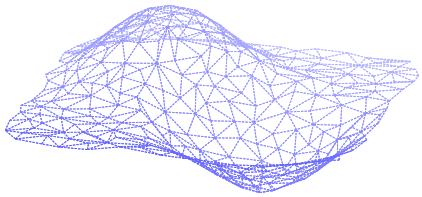
(g) Triangulation under Quasi-UDG ($\alpha = 0.4$).



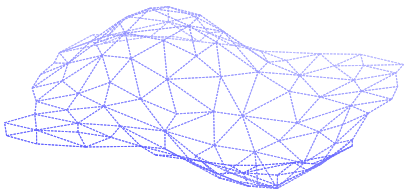
(h) Triangulation under Quasi-UDG ($\alpha = 0.6$).



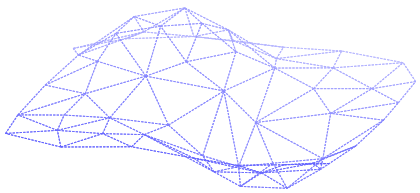
(i) Triangulation under Log-normal.



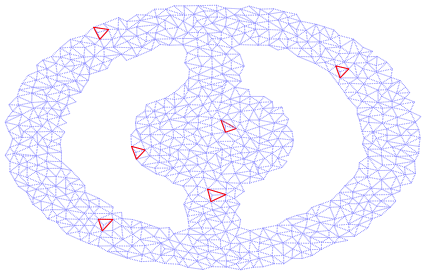
(j) Triangulation with $k = 1$.



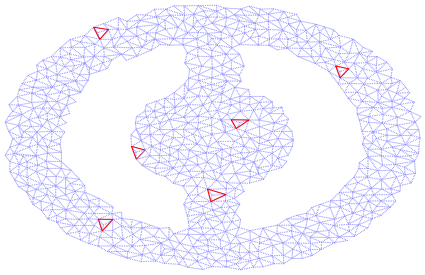
(k) Triangulation with $k = 2$.



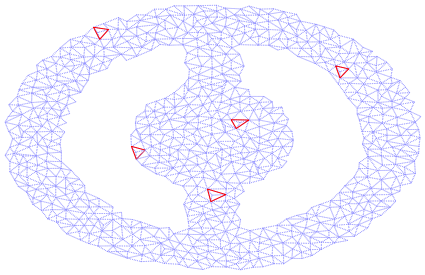
(l) Triangulation with $k = 3$.



(m) Triangulation under 10% distance errors.

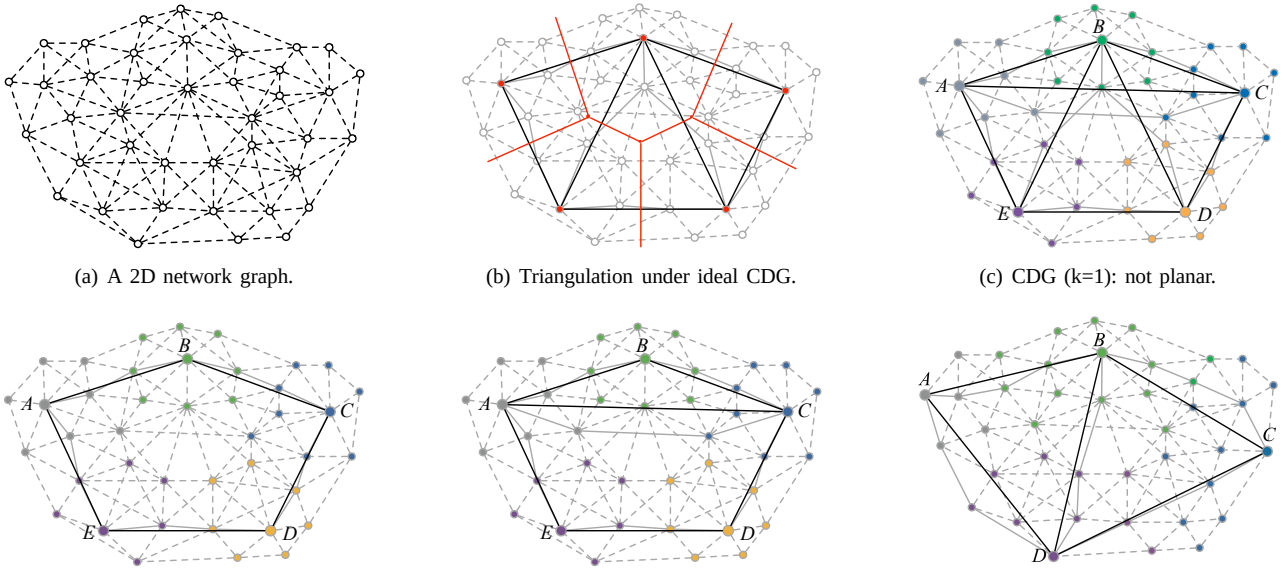


(n) Triangulation under 20% distance errors.



(o) Triangulation under 30% distance errors.

Fig. 1. Examples of triangulation by using the proposed algorithm. The default network setting is based the UDG communication model and the finest triangulation granularity. Figs. 1(g)-1(l) are based on the same set of sensor nodes as shown in Fig. 1(b), while Figs. 1(m)-1(o) are based on Fig. 1(a). The red lines in Figs. 1(g)-1(i) indicate different triangulation edges compared with Fig. 1(e) (under UDG model). The red lines in Figs. 1(m)-1(o) indicate incorrect weights due to distance errors. With such incorrect weights, however, triangulation is still successful in Figs. 1(m)-1(o).



(a) A 2D network graph. (b) Triangulation under ideal CDG. (c) CDG (k=1): not planar.

(d) CDM (k=1): planar but not triangulated. (e) Failed triangulation. (f) CDG (k=2): triangulated but coarse.

Fig. 2. Illustration of challenges in triangulation, where the circles indicate sensor nodes and the dashed lines are communication links. The solid black lines illustrate virtual edges, which are realized by corresponding paths shown as solid grey lines. In subfigure (b), the red lines depict Voronoi cells. In subfigures (c)-(f), the nodes with the same color belong to the same Voronoi cell, while the large circles (i.e., *A*-*E*) stand for landmarks.

subset of nodes as "landmarks" where any two landmarks are about $2k$-hops apart, where k is a given constant. A node is associated to its closest landmark, resulting in a landmark Voronoi complex (LVC). The respective dual of LVC, i.e., the combinatorial Delaunay graph (CDG), is obtained by connecting two landmarks by a virtual edge if a pair of their associated nodes are neighbors (i.e., their Voronoi cells share a side). In the ideal case where the nodal density is high and the Euclidian distances from a node to its nearby landmarks are known, CDG is a triangulated virtual backbone (see Fig. 2(b)). Under practical settings, however, CDG is not even planar, because multiple sensors associated with different landmarks may be adjacent to each other, leading to cross edges in CDG (as shown in Fig. 2(c) where *AC* intersects *BD* and *BE*).

To planarize CDG, each landmark sends a packet to a neighboring landmark through the shortest path. Two landmarks are said to be connected by a virtual edge if and only if the following two conditions are satisfied. First, all of the nodes visited by the packet and their 1-hop neighbors are associated to these two landmarks only. Second, assume the packet is sent from Landmark i to Landmark j. Then the packet must visit the nodes associated with Landmark i first, and then followed by the nodes associated with Landmark j, without interleaving. The resulting subgraph is a *Combinatorial Delaunay Map (CDM)*. It is proven that CDM is a planar graph under the Quasi-UDG communication model with $1 \geq \alpha \geq 1/\sqrt{2}$ [14].

However CDM is not always triangulated. Polygons with more than three edges may exist (see Polygon *ABCDE* in Fig 2(d)). To this end, heuristics are proposed in [2] and [3] to construct triangulated subgraphs by adding appropriate virtual edges in CDM. More specifically, if a landmark, e.g., Landmark i, has a non-connected neighboring landmark (e.g., Landmark j), it sends a *connection* packet to the latter. The packet will be dropped if it reaches an intermediate node that is already on the shortest path between two connected landmarks, in order to avoid cross virtual edges. If the *connection* packet arrives at Landmark j, a virtual edge can be safely added.

While the above scheme appears reasonable and has been employed to produce triangulated virtual backbone in several works, it does not ensure successful triangulation in polynomial time. For example, if a virtual edge is added between Landmarks *A* and *C* in Fig. 2(e), no other virtual edges can be further added, resulting in a failure in triangulation. None of the available heuristics [2], [3] can identify the appropriate sequence in adding virtual edges. Instead, they reduce the probability of failures by increasing k (i.e., by selecting a set of sparse landmarks). The larger the k, the bigger the Voronoi cells, and thus the lower the probability that multiple sensors associated with different landmarks are adjacent to each other. As a result, cross edges become rare in CDG (as demonstrated in Fig. 2(f)). In practice, k is usually set to $3 \sim 5$ or higher. Therefore, the triangulation is rather coarse, unsatisfactory for applications that require fine network graph features.

III. Triangulation for 2D Networks

In this section, we introduce the proposed triangulation algorithm for 2D networks.

A. Definitions

By following traditions, we represent a wireless sensor network by a graph $G(V,E)$, where V denotes the set of nodes and E the set of edges in the network. To facilitate our exposition, we first introduce several basic definitions.

Definition 1: The *node neighbor set (NNS)* of a node includes all of its one-hop neighboring nodes.

Let $N_v(i)$ denote the NNS of Node i. For example, $N_v(i) = \{h,k,l,m,j\}$ and $N_v(j) = \{k,l,m,n,i\}$ for Nodes i and j in Fig. 3(a), respectively.

Definition 2: The *edge neighbor set (ENS)* includes the common one-hop neighbors of the two end nodes of an edge.

The ENS of Edge e_{ij} is denoted by $N_e(e_{ij})$. For example, $N_e(e_{ij}) = N_v(i) \cap N_v(j) = \{k,l,m\}$, for Edge e_{ij} in Fig. 3(a).

Definition 3: The *refined edge neighbor set (RENS)* of an edge includes a subset of nodes in the ENS of the edge, such that each triangle formed by the edge and a node in its RENS does not contain any node in its ENS.

More specifically, let $R_e(e_{ij})$ denote the RENS of Edge e_{ij}. $R_e(e_{ij}) = \{v \mid v \in N_e(e_{ij})$ and $\triangle_{ijv}$ does not contain Node v', $\forall v' \in (N_e(e_{ij}) - v)\}$. For example, $R_e(e_{ij}) = \{l,m\}$ for Edge e_{ij} in Fig. 3(a). A node v' can judge if it is inside a triangle $\triangle_{ijv}$ based on locally estimated distances between the nodes. Such check excludes a triangle from containing small triangles, and intrinsically ensures no overlapped triangular faces in the final triangulation. The distances (e.g., approximately measured via received signal strength indicator (RSSI) or time difference of arrival (TDOA) [16]) are inaccurate in general. Such possible distance errors are considered in our simulations (see Figs. 1(m)-1(o) for example), and will be further discussed in Sec. V.

We assume that the edges on outer boundary and inner boundaries (i.e., boundaries of non-triangle polygon holes) are identified by an existing algorithm (e.g., [17]).

Definition 4: The *weight* of an edge is the cardinality of its RENS, if it is not on the boundary, or otherwise the cardinality of its RENS plus one.

The weight of Edge e_{ij} is denoted by $W(e_{ij})$. For example, $W(e_{ij}) = 2$ in Fig. 3(a). The weight of an edge indicates the number of triangles it forms with the nodes in its RENS.

Definition 5: The *associated edge neighbor set (AENS)* of an edge includes edges that are between one of the two end nodes of the edge and a node in the RENS of the edge.

Let $A(e_{ij})$ denote the AENS of Edge e_{ij}. Then $A(e_{ij}) = \{e_{ik} \mid i \in \{i,j\}$ *and* $k \in R_e(e_{ij})\}$. For example, $A(e_{ij}) = \{e_{il},\ e_{im},\ e_{jl},\ e_{jm}\}$ in Fig. 3(a). If Edge e_{ij} is removed, the weight of the edges in $A(e_{ij})$ will be reduced by one, while other edges in the graph remain unchanged (see Fig. 3(b)).

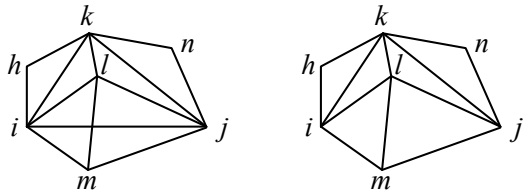
Definition 6: Two edges are called *equivalent edges* if they share the same AENS.

For example, Edges e_{ij} and e_{lm} in Fig. 3(a) are equivalent edges. Equivalent edges are two diagonals of a quadrilateral. The removal of any one of them leads to the same impact on the weight of the edges in their AENS.

Definition 7: For a given edge, four edges in its AENS may form a quadrilateral. If the edge is the only diagonal of the quadrilateral, it is marked as a *critical edge*.

For example, Edge e_{kj} in Fig. 3(a) is a critical edge. If a critical edge is removed, a hole will be formed in the graph, because there is no diagonal in the corresponding quadrilateral.

NNS, ENS, RENS, AENS, edge weight, and critical and equivalent edges can all be determined by local information.



(a) Original graph. (b) Triangulated subgraph.

Fig. 3. Illustration of node neighbor set, edge neighbor set, refined edge neighbor set, edge weight, associated edge neighbor set, equivalent edges, and critical edges. For Nodes i and j in the original graph, $N_v(i) = \{h,k,l,m,j\}$ and $N_v(j) = \{k,l,m,n,i\}$. For Edge e_{ij} in the original graph, $N_e(e_{ij}) = \{k,l,m\}$, $R_e(e_{ij}) = \{l,m\}$, $W(e_{ij}) = 2$, and $A(e_{ij}) = \{e_{il},\ e_{im},\ e_{jl},\ e_{jm}\}$. Edges e_{ij} and e_{lm} in the original graph are equivalent edges. Edge e_{kj} is a critical edge.

Our objective is to develop a distributed algorithm that can identify a triangulated subgraph of a given network graph G. More specifically, we have:

Objective 1: Given a graph G, if there exists a subgraph T that is triangulated, our proposed algorithm can alway discover a triangulation of G.

Note that G may have multiple triangulated subgraphs. The discovery of any of them satisfies the above objective. In the rest of this section, we focus on this objective by assuming that a triangulation exists for a graph G and developing algorithm to identify a triangulated subgraph in G. If Objective 1 is achieved, then as a contrapositive, we also have:

Objective 2: If the proposed algorithm fails to discover a triangulated subgraph of G, then there does not exist a triangulation for G.

B. Theory

Our proposed triangulation algorithm is motivated by the property of edge weight in a triangulated subgraph, as revealed by Lemma 1.

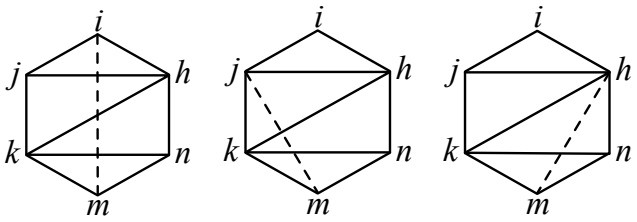
Lemma 1: A subgraph of G is triangulated if and only if every edge of the subgraph has a weight of two.

Proof: We first show the necessary condition. In a triangulated subgraph, a non-boundary edge is shared by two triangles and a boundary edge is involved in one triangle only. Therefore the weight of an edge must be two according to Definition 4. The proof for sufficient condition is straightforward too. If every edge of the subgraph has a weight of two, then any two simplices (i.e., triangles) intersect in no more than one common edge, thus satisfying the definition of triangulation given in Sec. I. ■

According to Lemma 1, if any edge in a graph has a weight not equal to 2, the graph is not triangulated. In other words, there are extra edges besides the edges in a triangulated subgraph and such extra edges must be removed to arrive at a triangulation.

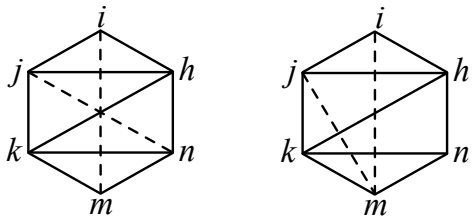
Definition 8: For a given graph G and a triangulated subgraph T of G, an edge in T is called a *triangulation edge*, while an edge in $G - T$ is called an *extra edge*.

Each extra edge can be viewed as an "added" edge to a triangulated subgraph. Let's first consider a single extra edge only and ignore other extra edges and the interaction



(a) e_0 (b) e_1 (c) e_2

Fig. 4. Three types of extra edge, e_0, e_1, and e_2, with $W(e_0) = 0$, $W(e_1) = 1$, and $W(e_2) = 2$.



(a) (b)

Fig. 5. Illustration of independent and dependent extra edges (indicated by dashed lines). (a) Independent extra edges. (b) Dependent extra edges.

among them for now. The extra edge exists in three ways as illustrated in Fig. 4, dubbed e_0, e_1, and e_2, respectively. An e_0 edge (see Edge e_{im} in Fig. 4(a) for example) does not form a triangle with any triangulation edges. Its weight is zero. Adding such an extra edge does not affect the weight of existing triangulation edges. An e_1 edge forms one triangle with two triangulation edges (as shown in Fig. 4(b)). It has a weight of one and increases the weight of two triangulation edges by one. Fig. 4(c) illustrates an e_2 edge, which has a weight of two and increases the weight of four triangulation edges by one.

Definition 9: Two extra edges are *independent* if they are not in each other's AENS.

If all extra edges are independent (see Fig. 5(a)), their weights are 0, 1, and 2 for e_0, e_1, and e_2 edges, respectively, as discussed above. However, if they are not independent, the extra edges themselves may form triangles, and thus increase their weight (as illustrated in Fig. 5(b)).

To produce a triangulated subgraph, we must remove all corresponding extra edges.

Theorem 1: An edge with a weight less than two must be removed in order to produce a triangulated subgraph.

Proof: To prove the theorem, we show that an edge with a weight less than two must be an extra edge of e_0 or e_1 type and thus must be removed.

Without extra edges, the weight of a triangulation edge is two. By taking extra edges into consideration, the weight of a triangulation edge either remains as two or increases to three or higher. Similarly, an e_2 extra edge must have a weigh of two or higher in G.

Therefore, an edge with a weight less than two must be an extra edge of e_0 or e_1 type. Such an extra edge must be removed in order to produce a triangulated subgraph. ■

Based on Theorem 1, it is safe to remove an edge whose weight is less than two. More specifically, if an edge finds its weight less than two, it is removed from G and each edge in the AENS of the removed edge must decrease its weigh by one, because it no longer forms a triangle with the removed edge. Note that after an edge is removed, it may lead to another edge's weight lower than two and thus being removed subsequently. After this step, we arrive at a subgraph of G, denoted by G', in which every edge has a weight no less than two. Most e_0 and e_1 edges, if not all, have been removed by now, except those in two special structures as to be discussed in Lemmas 2 and 3.

Next, we discuss how to remove e_2 edges in G'.

Theorem 2: An non-critical edge can be recognized as an e_2 edge and safely removed if all edges in its AENS have their weight greater than two.

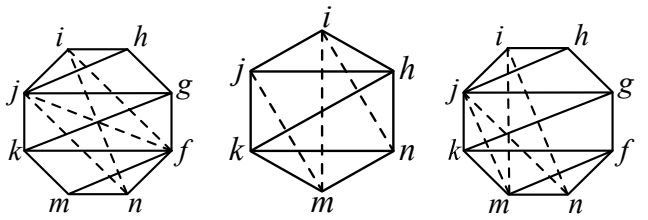
Proof: The theorem is twofold. First, all e_2 extra edges (or their corresponding equivalents) will be removed. Second, a triangulation edge (or its equivalent) will be safe (i.e., not be removed).

We prove the first part of the theorem via deduction. Let $G'(k)$ denote a subgraph G' with k e_2 extra edges. First, if there is only one e_2 edge in G', it can be easily identified, since it has a weigh of two and each edge in its AENS has a weight of three (see Edge e_{mh} in Fig. 4(c) for example). Therefore, the edge can be removed, and the edges in its AENS reduce their weight by one, arriving at a triangulated subgraph where every edge has a weight of two. Note that there may exist multiple such edges (e.g., Edge e_{kn} and Edge e_{mh} in Fig. 4(c)) that can be treated as an e_2 extra edge and removed to yield a equally good triangulation. We do not differentiate them. Then we show that a subgraph with $k+1$ e_2 edges (i.e., $G'(k+1)$) can be reduced to a subgraph with k e_2 edges (i.e., $G'(k)$), where $k \geq 1$. $G'(k+1)$ can be considered as a result of adding an extra e_2 edge (denoted by e') in $G'(k)$. Since all edges in $G'(k)$ have their weight no less than two, adding e' increases the weight of the edges in $A(e')$ by one and thus become great than two. Therefore it can be identified and removed, reducing $G'(k+1)$ to $G'(k)$. Alternatively, any e_2 edge originally in $G'(k)$ has the same property as e' and thus may be removed to reduce $G'(k+1)$ to $G'(k)$ too. Therefore all e_2 edges (or their corresponding equivalents) can be removed.

Now we prove the second part of the theorem. A triangulation edge (e.g., Edge e_{kn} in Fig. 4(c)) is always associated with a quadrilateral formed by four edges in its AENS (i.e., Edges e_{km}, e_{mn}, e_{nh}, and e_{hk} in Fig. 4(c)). If the triangulation edge is the only diagonal of the quadrilateral (by assuming that Edge e_{mh} does not exist), it is a critical edge and thus will not be removed. Otherwise, if there are two diagonals, then they are equivalent. If any one of them is removed, the other edge becomes critical, and thus will be kept. As a result, either the triangulation edge or its equivalent will not be removed.

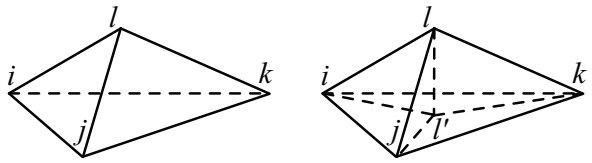
Therefore the theorem is proven. ■

The subgraph obtained by the above process is denoted by G''. If all extra edges in G' are e_2 type, then G'' is already a triangulated subgraph. Now we consider the scenarios with e_0



(a) Loop chain. (b) Non-loop chain. (c) Same head and tail.

Fig. 6. Illustration of non-loop and loop chains.



(a) Node l and $\triangle_{ijk}$ form a tetrahedron.

(b) Project Node l onto the plane of $\triangle_{ijk}$.

Fig. 7. Illustration of projection in 3D surface networks.

and e_1 edges remaining in G''.

Lemma 2: An e_0 extra edge can exist in G'', only if it depends on at least two other extra edges.

Proof: As discussed earlier, an e_0 edge has a weight of zero, if it doesn't depend on other extra edges. When it is dependent on another extra edge, or in other words, it is in the AENS of another extra edge, its weight is increased by one. Since the weight of every edge in G'' must be no less than 2, the e_0 extra edge must at least depend on two other extra edges. ■

Lemma 3: An e_1 extra edge can exist in G'', only if it at least depends on another extra edge.

Proof: Similar to the proof of Lemma 2. ■

To keep their weight no less than two, the e_0 and e_1 edges must depend on each other in G'' by forming a "chain". There are two types of chains, namely loop and non-loop chains (as shown Fig. 6). A loop chain consists of all e_0 edges, forming a loop as shown in Fig. 6(a). In a non-loop chain, the head or tail edge of the chain must be e_1, because it only has one dependent extra edge. An example of the non-loop chain in illustrated in Fig. 6(b): $e_{jm} - e_{mi} - e_{in}$, which is in a pattern of $e_1 - e_0 - e_1$. A non-loop chain has a minimum length of two edges. In addition, Fig. 6(c) shows a special case of the non-loop chain, where the head and tail are the same e_1 edge (i.e., Edge e_{jm}).

It is easy to identify a chain (either loop or non-loop). Except the e_1 edge in a non-loop chain that serves as both head and tail (e.g., Edge e_{jm} in Fig. 6(c)), an edge in a chain has a weight of two, and at least one edge in its AENS has its weigh equal to two and another edge in its AENS has its weigh greater than two. Therefore, we can start from one such edge, and expand the chain by adding another edge with the same property and in the AENS of (i.e., dependent to) an edge already in the chain, and so on and so forth, until no additional edges can be added.

If the identified chain has a length of two or longer, all edges in the chain are removed. Similar to our earlier discussions, once an edge is removed, its relevant edges update their weight accordingly. As a specific case, the e_1 edge that serves as both head and tail of a non-loop chain (as shown in Fig. 6(c)) will be left alone as a separate e_1 edge with a weight of one, after other edges in its chain are removed. Therefore, it removes itself, according the Theorem 1.

C. Algorithm

The theory discussed above provides a clear guideline for our algorithm development. The proposed algorithm is fully distributed. Each edge performs the following operations:

Step 1. Initialization. During initialization, each edge communicates with its neighbors to acquire local information to determine its NNS, ENS, RENS, AENS, edge weight, critical edge and equivalent edges.

Step 2. Iterative edge removal. After initialization, each edge checks if itself should be removed according to Theorems 1 and 2. This is an iterative process, because the removal of an edge may affect its neighboring edges (including all of their parameters discussed above). We assume that two dependent edges are not removed at exactly the same time via a local signaling scheme, to avoid inconsistency.

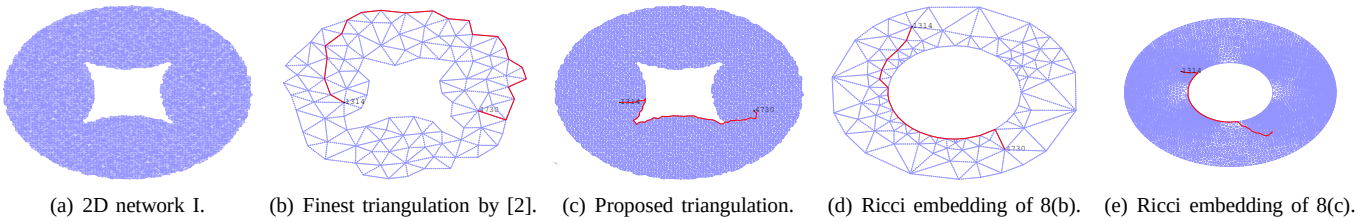
- An edge removes itself and informs its neighbors of the removal if it finds its weight less than two.
- An edge removes itself and informs its neighbors of the removal if it is an non-critical edge and all edges in its AENS have their weight greater than two.
- An edge updates its weight whenever there is a change in its AENS.

Step 3. Removal of e_0 and e_1 chains. The remaining extra edges not removed in Step 2 are e_0 and e_1 edges that form chain(s). A chain is identified according to its properties revealed by Lemmas 2 and 3 and relevant discussions in Sec. III-B. All edges in the chain with a length of two or longer are e_0 and e_1 edges and thus removed. At the same time, a remaining edge updates its weight and removes itself if its weight becomes less than two.

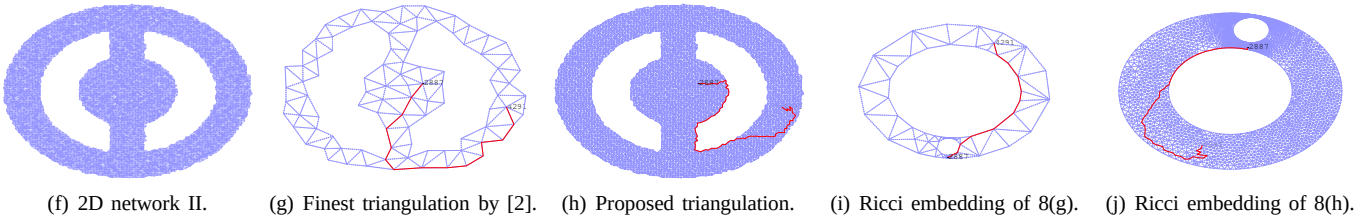
The time complexity of the algorithm depends on the iterations in Step 2. Since an iteration removes at least one edge and no edges are added during the process, the complexity is $O(n)$, where n is the number of nodes in the network.

IV. Triangulation for 3D Surface Networks

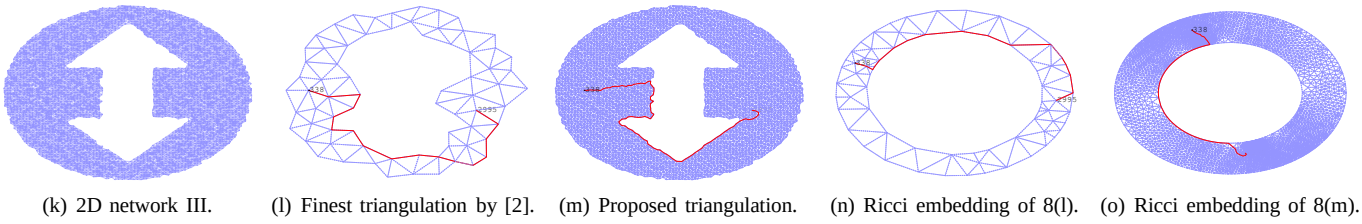
The triangulation algorithm discussed above can be extended to a wireless network deployed on a 3D surface. However, although a small chart of smooth 3D surface is intrinsically the same as the 2D plane in theory, the former introduces additional challenges in the calculation of edge weight under practical network settings. More specifically, to determine the RENS and accordingly the edge weight correctly, one needs to judge if a triangle contains any nodes. Under a 2D setting, it excludes a big triangle that contains nodes (see $\triangle_{ijk}$ illustrated in Fig 3(a)), intrinsically ensuring



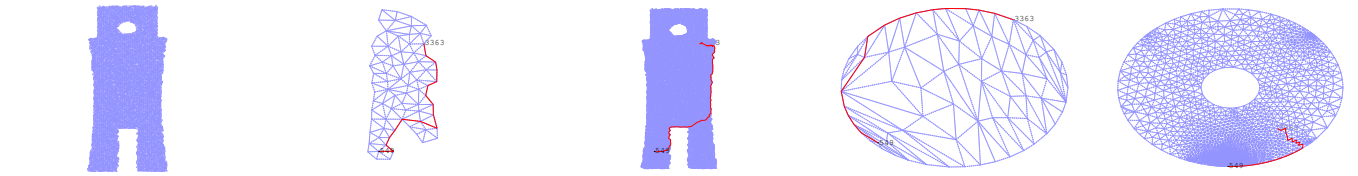
(a) 2D network I. (b) Finest triangulation by [2]. (c) Proposed triangulation. (d) Ricci embedding of 8(b). (e) Ricci embedding of 8(c).



(f) 2D network II. (g) Finest triangulation by [2]. (h) Proposed triangulation. (i) Ricci embedding of 8(g). (j) Ricci embedding of 8(h).

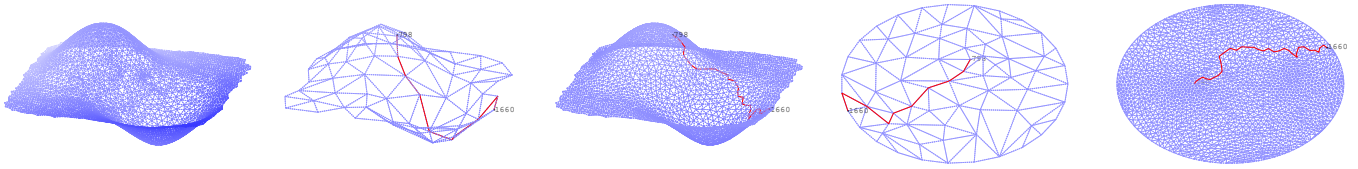


(k) 2D network III. (l) Finest triangulation by [2]. (m) Proposed triangulation. (n) Ricci embedding of 8(l). (o) Ricci embedding of 8(m).

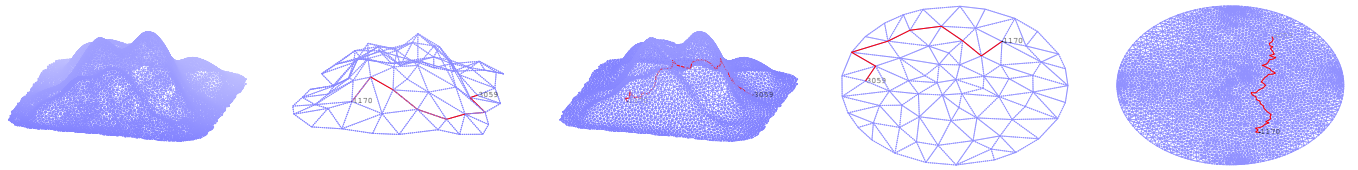


(p) 2D network IV. (q) Finest triangulation by [2]. (r) Proposed triangulation. (s) Ricci embedding of 8(q). (t) Ricci embedding of 8(r).

Fig. 8. Examples of 2D networks. The proposed algorithm produces finer triangulation than [2], and accordingly leads to shorter greedy routing path as indicated by the thick red lines.



(a) 3D open surface network I. (b) Finest triangulation by [3]. (c) Proposed triangulation. (d) Ricci embedding of 9(b). (e) Ricci embedding of 9(c).



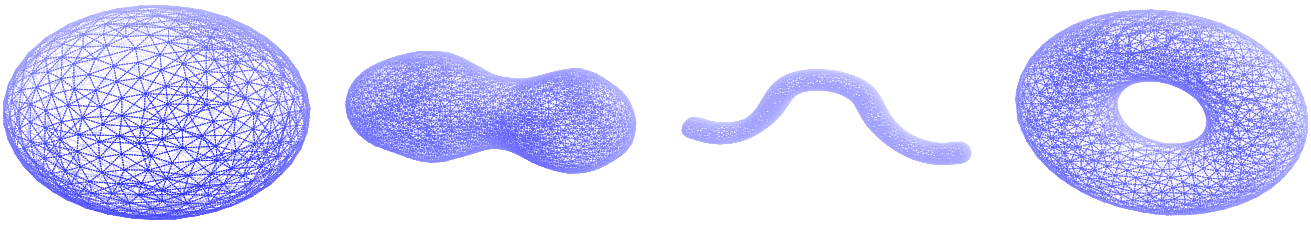
(f) 3D open surface network II. (g) Finest triangulation by [3]. (h) Proposed triangulation. (i) Ricci embedding of 9(g). (j) Ricci embedding of 9(h).

Fig. 9. Examples of 3D open surface networks. The proposed algorithm produces finer triangulation than [3], and accordingly leads to shorter greedy routing path as indicated by the thick red lines.

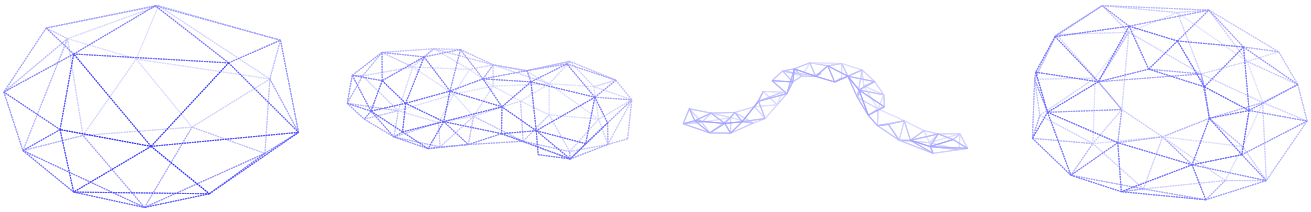
no overlapped triangular faces in the final triangulation. Similarly, in 3D networks, it excludes the underneath triangle such as $\triangle_{ijk}$ in Fig. 7(a) if $\triangle_{ijl}$, $\triangle_{ikl}$ and $\triangle_{jkl}$ are on the surface.

For Edge e_{ij}, Node k is in its RENS, if $\triangle_{ijk}$ contains no nodes in Edge e_{ij}'s edge neighbor set, i.e., $\triangle_{ijk}$ does not contain l, $\forall l \in (N_e(e_{ij}) - \{k\})$. While it is straightforward to check if this is true in a 2D network by a simple calculation based on locally estimated distances between nodes, similar method cannot be applied in 3D directly because Node l is not necessarily on the plane of $\triangle_{ijk}$ in a 3D network.

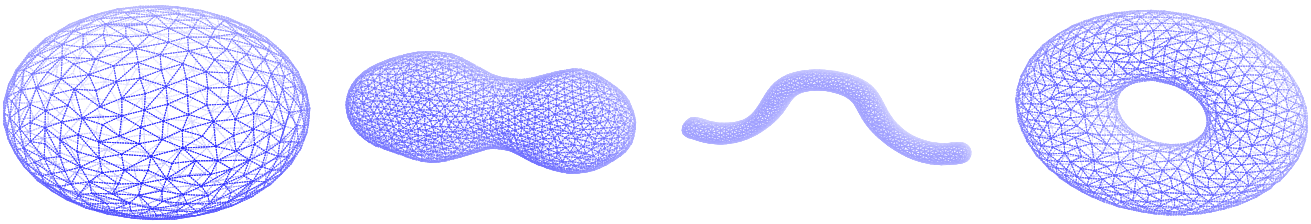
To this end, we propose a localized projection algorithm. Without loss of generality, let's consider Edge e_{ij}. For Node



(a) 3D closed surface network I. (b) 3D closed surface network III. (c) 3D closed surface network IV. (d) 3D closed surface network II.



(e) Finest triangulation by [3] (f) Finest triangulation by [3] (g) Finest triangulation by [3] (h) Finest triangulation by [3]



(i) Proposed triangulation. (j) Proposed triangulation. (k) Proposed triangulation. (l) Proposed triangulation.

Fig. 10. Examples of 3D closed surface networks. The proposed algorithm produces finer triangulation than [3].

$k \in N_e(e_{ij})$ and Node $l \in (N_e(e_{ij}) - \{k\})$, Nodes i, j, k and l form a tetrahedron as shown in Fig. 7(a). Based on locally estimated distances between them, a local coordinates system can be established for such four nodes by existing algorithms [18]–[22]. Similar to our discussions on 2D networks, the estimated distances are generally inaccurate, resulting in errors in local coordinates that will be discussed in Sec. V.

Let (x_i, y_i, z_i) denote the coordinates of Node i. It is now ready to calculate the projection of Node l on the plane defined by Nodes i, j and k, i.e., l', by solving the following equations:

$$\begin{cases} (x_{l'} - x_i)^2 + (y_{l'} - y_i)^2 + y_l^2 = L_{il}^2, \\ (x_{l'} - x_j)^2 + (y_{l'} - y_j)^2 + y_l^2 = L_{jl}^2, \\ (x_{l'} - x_k)^2 + (y_{l'} - y_k)^2 + y_l^2 = L_{kl}^2, \\ z_{l'} = 0. \end{cases}$$

Subsequently, it is straightforward to check if l' is inside $\triangle_{ijk}$. If it is true (see Fig. 7(b) for example), Node k is not included in the RENS of Edge e_{ij}. Note that when there are only two nodes in an edge's ENS, it is not necessary to apply projection. This prevents miscalculation of edge weight when the edge is located at the corner of a surface.

After each edge determines its RENS, it obtains its weight according to Definition 4. If the surface is closed (see Fig. 1(c) for example), there is no boundary edge. Otherwise (i.e., for an open surface network as illustrated in Fig. 1(b)), the boundary edges are marked similar to the 2D scenario. The rest of the triangulation algorithm in 3D surface network follows the same steps in the 2D scenario as discussed in Sec. III.

V. Applications and Simulation Results

To demonstrate the effectiveness of our proposed triangulation algorithm, we have applied it in various 2D, 3D open surface and closed surface networks, under different communication models and distance measurement errors. For example, Figs. 1(a)-1(f) show a 2D, a 3D open surface and a 3D closed surface network and their triangulations, respectively. Figs. 1(g)-1(i) illustrate triangulations under quasi-UDG and log-normal communication models, where the red lines indicate different edges in triangulation, in comparison with the result under the UDG model. Figs. 1(j)-1(l) demonstrate the triangulations with different granularities, where each edge in triangulations is correspondent to a path of $2k$ hops in the underlying sensor network. Figs. 1(m)-1(o) depict the triangulation results under different distance measurement errors, which results in incorrect weights of some edges as highlighted by red lines. Such incorrect weights, however, do not prevent successful triangulation as shown in Figs. 1(m)-1(o). More examples of 2D, 3D open surface and closed surface networks are given in Figs 8-10, respectively, showing that our proposed algorithm produces finer triangulation than [2] or [3] does.

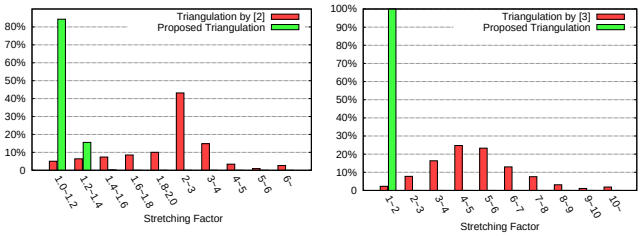
In addition to granularity, we further demonstrate the improvement of our triangulation by using an application, i.e., greedy routing based on Ricci flow [2], which maps a triangulated graph to an embedding with circular boundaries that ensure successful greedy routing. The algorithm can be applied on 2D and 3D open surface, but not 3D closed surface

TABLE I
COMPARISON OF STRETCH FACTOR IN GREEDY ROUTING.

	2D networks	3D networks
Triangulation by [2], [3]	2.065	5.417
Proposed Triangulation	1.214	1.091

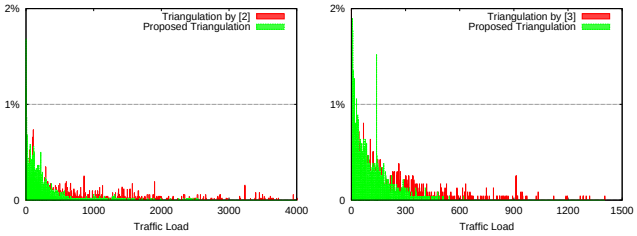
networks. Several examples are given in Figs. 8 and 9. For example, given a 2D network shown in Fig. 8(a), its triangulations under [2] and our proposed algorithm are depicted in Figs. 8(b) and 8(c), respectively, and their corresponding Ricci embeddings are shown in Figs. 8(d) and 8(e). Although greedy routing is successful under both triangulations because Ricci flow maps their boundaries to circles (see Figs. 8(d) and 8(e)), the routing path (indicated by the red line) is much longer under the triangulation by [2], because it must go through the long edges in the coarse triangulation.

We have quantitively evaluated the efficiency of greedy routing under different triangulations. Table I compares their stretch factor (i.e., the ratio of a greedy routing path to its corresponding shortest path), based on pair-wise paths of the nodes in the networks depicted in Figs. 8 and 9. As can be seen, our proposed triangulation yields significantly lower stretch factor in comparison with [2] or [3]. Fig. 11 illustrates the distribution of stretch factor. Routing paths always have low stretch factor under our triangulation, while the coarse triangulation by [2], [3] leads to a wide spread stretch factors up to over 10 (i.e., 10 times of the shortest path length). We have also evaluated load distribution, where the load of a node is signified by the number of routing paths it involves. As shown in Fig. 12, load is more uniformly distributed under the fine triangulation produced by our proposed algorithm, where less nodes suffer high load, because it includes more nodes in the triangulation and consequently the greedy routing paths, which together partake the traffic load.



(a) 2D networks. (b) 3D networks.

Fig. 11. Distribution of stretch factor in greedy routing.



(a) 2D networks. (b) 3D networks.

Fig. 12. Distribution of traffic load in greedy routing.

VI. CONCLUSION

In this paper we have proposed a distributed algorithm that triangulates an arbitrary sensor network, with no constraints on the communication model or the granularity of the triangulation. We have proven its correctness in 2D, and further extend it to 3D surface networks. Our simulation results have shown that the proposed algorithm can tolerate distance measurement errors, and thus work well under practical sensor network settings and effectively promote the performance a range of applications that depend on triangulations.

REFERENCES

[1] R. Hekmat and P. V. Mieghem, "Connectivity in Wireless Ad-Hoc Networks with a Log-Normal Radio Model," *Mobile Networks and Applications*, vol. 11, no. 3, pp. 351–360, 2006.

[2] R. Sarkar, X. Yin, J. Gao, F. Luo, and X. D. Gu, "Greedy Routing with Guaranteed Delivery Using Ricci Flows," in *Proc. of IPSN*, pp. 121–132, 2009.

[3] H. Zhou, S. Xia, M. Jin, and H. Wu, "Localized Algorithm for Precise Boundary Detection in 3D Wireless Networks," in *Proc. of ICDCS*, 2010.

[4] R. Flury, S. V. Pemmaraju, and R. Wattenhofer, "Greedy Routing with Bounded Stretch," in *Proc. of INFOCOM*, pp. 1737–1745, 2009.

[5] W. Zeng, R. Sarkar, F. Luo, X. D. Gu, and J. Gao, "Resilient Routing for Sensor Networks using Hyperbolic Embedding of Universal Covering Space," in *Proc. of INFOCOM*, 2010.

[6] S. Xia, M. Jin, and H. Wu, "GPS-Free Greedy Routing with Delivery Guarantee and Low Stretch Factor in 2D and 3D Surfaces," Tech. Rep. UL-CACS-2010-06-01, University of Louisiana at Lafayette, 2010. http://www.cacs.louisiana.edu/?q=research/techreports.

[7] S. Lederer, Y. Wang, and J. Gao, "Connectivity-based Localization of Large Scale Sensor Networks with Complex Shape," in *Proc. of INFOCOM*, pp. 789–797, 2008.

[8] S. Xia, H. Wu, and M. Jin, "Range-Free and Anchor-Free Localization Using Discrete Yamabe Flow," Tech. Rep. UL-CACS-2010-06-02, University of Louisiana at Lafayette, 2010. http://www.cacs.louisiana.edu/?q=research/techreports.

[9] M.-C. Zhao, J. Lei, M.-Y. Wu, Y. Liu, and W. Shu, "Surface Coverage in Wireless Sensor Networks," in *Proc. of INFOCOM*, pp. 109–117, 2009.

[10] H. Zhou, N. Ting, M. Jin, H. Wu, and S. Xia, "Segmentation in 3D Wireless Sensor Networks: Algorithms and Applications," Tech. Rep. UL-CACS-2010-06-03, University of Louisiana at Lafayette, 2010. http://www.cacs.louisiana.edu/?q=research/techreports.

[11] R. Sarkar, W. Zeng, J. Gao, and X. D. Gu, "Covering Space for In-Network Sensor Data Storage," in *Proc. of IPSN*, 2010.

[12] E. L. Lloyd, "On Triangulations of a Set of Points in the Plane," in *Proc. of the 18th Annual IEEE Symposium on Foundations of Computer Science*, pp. 228–240, 1977.

[13] M. Berg, O. Cheong, M. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Springer, 2008.

[14] S. Funke and N. Milosavljevi, "How Much Geometry Hides in Connectivity? - Part II," in *Proc. of SODA*, pp. 958–967, 2007.

[15] Q. Fang, J. Gao, L. J. Guibas, V. Silva, and L. Zhang, "GLIDER: Gradient Landmark-based Distributed Routing for Sensor Networks," in *Proc. of INFOCOM*, pp. 339–350, 2005.

[16] Z. Zhong and T. He, "MSP: Multi-Sequence Positioning of Wireless Sensor Nodes," in *Proc. of SenSys*, pp. 15–28, 2007.

[17] D. Dong, Y. Liu, and X. Liao, "Fine-Grained Boundary Recognition in Wireless Ad Hoc and Sensor Networks by Topological Methods," in *Proc. of MobiHOC*, pp. 135–144, 2009.

[18] H. Wu, C. Wang, and N.-F. Tzeng, "Novel Self-Configurable Positioning Technique for Multi-hop Wireless Networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 3, pp. 609–621, 2005.

[19] G. Giorgetti, S. Gupta, and G. Manes, "Wireless Localization Using Self-Organizing Maps," in *Proc. of IPSN*, pp. 293 – 302, 2007.

[20] L. Li and T. Kunz, "Localization Applying An Efficient Neural Network Mapping," in *Proc. of The 1st International Conference on Autonomic Computing and Communication Systems*, pp. 1–9, 2007.

[21] Y. Shang, W. Ruml, Y. Zhang, and M. P. J. Fromherz, "Localization from Mere Connectivity," in *Proc. of MobiHOC*, pp. 201–212, 2003.

[22] Y. Shang and W. Ruml, "Improved MDS-based Localization," in *Proc. of INFOCOM*, pp. 2640–2651, 2004.