Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2014, Article ID 923751, 14 pages
http://dx.doi.org/10.1155/2014/923751

Research Article

FLOC-SPANNER: An $O(1)$ Time, Locally Self-Stabilizing Algorithm for Geometric Spanner Construction in a Wireless Sensor Network

G. Ranganath and V. Kulathumani

Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26506-6109, USA

Correspondence should be addressed to V. Kulathumani; vinod.kulathumani@mail.wvu.edu

Received 26 June 2013; Accepted 25 November 2013; Published 18 February 2014

Academic Editor: Prabir Barooah

We present a distributed algorithm for creation of geometric spanners in a wireless sensor network. Given any connected network, we show that the algorithm terminates in $O(1)$ time, irrespective of network size. Our algorithm uses an underlying clustering algorithm as a foundation for creating spanners and only relies on the periodic heartbeat messages associated with cluster maintenance for the creation of the spanners. The algorithm is also shown to stabilize locally in the presence of node additions and deletions. The performance of our algorithm is verified using large scale simulations. The average path length ratio for routing along the spanner for large networks is shown to be less than 2.

1. Introduction

From a communications perspective, a wireless sensor actuator network can be modeled as a graph $G = (V, E)$, where V represents the set of nodes and E represents the set of edges. Let G be a connected graph and, given any pair of nodes u and v, let $\gamma_G(u, v)$ denote the length of the shortest path between u and v in graph G in terms of the number of links traversed. Let $G' \subseteq G$ be a connected subgraph of G. G' is a geometric spanner of G if there exists constant $q \geq 1$ such that for all pairs of vertices $u, v \in V$, $\gamma_{G'}(u, v) \leq q \cdot \gamma_G(u, v)$. The constant q is referred to as the *path stretch factor* for routing along the spanner. Geometric spanners are a popular form of topology control in wireless networks because they yield an efficient, reduced interference subgraph for both unicast and broadcast routing.

In this paper, we present *FLOC-SPANNER*, a distributed, locally self-stabilizing algorithm for creation of geometric spanners. Our algorithm is built on top of *FLOC* [1], a locally, self-stabilizing algorithm for creation of solid-disk clusters in wireless sensor networks. In summary, *FLOC* partitions a wireless network into clusters such that all nodes within a certain radius around each clusterhead, necessarily, belong to that cluster. This ensures that neighboring clusterheads are separated by a certain distance and also allows roughly uniform distribution of clusters and cluster-sizes across the network. While maintaining this *solid-disk* property, *FLOC* also ensures that node additions and deletions do not result in a global reformation of clusters by allowing a dilation factor m of at least 2 in the size of each cluster. Thus, while all nodes within a unit distance of a clusterhead belong to that cluster, nodes up to a distance of m hops can belong to that cluster. This property ensures that node additions and deletions can be handled locally without global restructuring. In this paper, we have considered $m = 2$. Thus, the solid-disk property ensures that all nodes within one hop of the clusterhead belong to that cluster and nodes up to 2 hops away *can* belong to the cluster.

While *FLOC* creates clusters in the network, there are no structures established for connecting these clusters and enabling communication between clusters. As a result, the clusters by themselves do not form a connected subgraph. Establishing a sparse but sufficient set of connections between the clusters and thus realizing a spanner graph is not trivial. This is because there are potentially a large number of candidate pairs between neighboring clusters that can form connecting links between neighboring clusters and the challenge is to avoid redundant connections while still ensuring that the

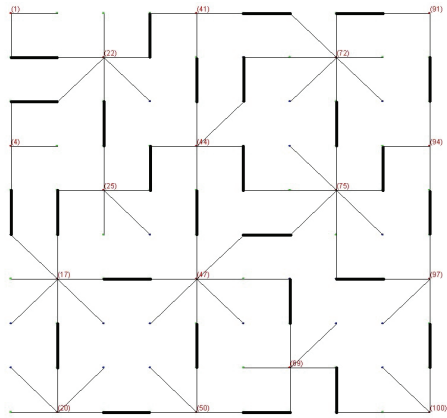


Figure 1: Illustration of a spanner graph created on a 10-by-10 Grid with FLOC-SPANNER. The bold edges represent the intercluster connections while the lighter edges represent the links from clusterhead to cluster members.

graph remains connected. The algorithm *FLOC-SPANNER* presented in this paper addresses this challenge and creates a geometric spanner by establishing sparse, yet sufficient connections between clusters created by *FLOC*. Figure 1 illustrates a spanner graph created using our algorithm on a 10 by 10 grid.

The key characteristics of our proposed algorithm are summarized below.

(i) During creation of connections between clusters, we maintain the property that if two nodes a and b are used to connect clusters c_a and c_b, then no other connections exist between these clusters within i-band range of a and b. This property ensures that the connectors are sparse and the total number of edges in the resulting spanner is $O(N)$, where N is the number of vertices as opposed to $O(Nd)$, where d is the average degree of nodes in the network.

(ii) We piggyback entirely on the periodic and low frequency maintenance messages of the underlying clustering algorithm to establish as well as maintain the spanner structure. As a result no further messaging overhead is needed for *FLOC-SPANNER*.

(iii) Note that *FLOC* creates clusters in $O(1)$ time and once any cluster is formed *FLOC-SPANNER* ensures that the cluster is connected to all neighboring clusters within $O(1)$ time (irrespective of the network size).

(iv) Unlike several other algorithms for spanner creation, *FLOC-SPANNER* does not require location information for its operation. This makes deployment easier.

(v) The upper bound on the path stretch factor in the resulting spanner is shown to be $O(5/2+4/d)$, where d is the length of the shortest path on the complete network graph. The upper bound thus converges to 2.5 as distance between nodes increases. Furthermore, the average path stretch factor is observed to be less than 2 on networks of 400 to 2000 nodes.

(vi) Utilization of *FLOC* for clustering also ensures that impact of link changes (additions or removals) is contained within a bounded distance from the source of the event.

(vii) The resulting spanner structure can be used for broadcasting system-wide state information in a one-to-all as well as all-to-all mode. Furthermore the upper bound on the path stretch factor ensures that, even in a broadcast mode, information can be transferred in time proportional to the shortest path between two nodes. This property of *distance-sensitivity* is significant for many querying, tracking, and control applications of wireless sensor actuator networks [2–4].

We analyze the performance of *FLOC-SPANNER* using large scale simulations in JProwler, a discrete event simulator for wireless sensor networks. Specifically, we measure the path stretch factor, number of spanner edges, and the time for spanner creation under different network sizes ranging from 400 nodes to 2000 nodes with different network densities. Our measurements show that the number of spanner edges grows linearly with the network size while the number of connector edges per cluster, path stretch factor, and the time to completion remain constant, independent of the network size. The average path stretch factor is observed to be <2.

2. Model

2.1. Network Model. We consider a wireless ad hoc network in which nodes lie on an undirected graph topology. Nodes are assumed to be identical in their processing, data transmission, and reception capacity. We assume a dual-band model for the radio range; that is, the nodes are considered to be in either inner-band or outer-band region. A node can communicate reliably with nodes that are in the inner-band range and unreliably with nodes that are in outer-band range. A similar model is assumed in [1]. This is a reasonable assumption to make considering that wireless radios have been shown to exhibit a dual-band characteristic in which received signal strengths are significantly high and isotropic within an inner band and exhibit high variance (due to time-varying interference and multipath effects) in an outer band [5]. Unit distance is defined as the i-band range of each node. Nodes can locally determine whether a neighbor is within the i-band by using received signal strength, time of flight, or ultrasound based techniques. The network graph consisting of only i-band links is assumed to be connected. Each node has access to local timers that are used for the tasks such as sending of heartbeats periodically and for timing out when waiting on a condition. The network is not required to be synchronized in time.

2.2. Execution Semantics. Nodes have unique ids. We use i, j, and k to denote the nodes and $j \cdot var$ to denote a program variable residing at j. We denote a message broadcast by j as msg_j. A program consists of a set of variables and actions at each node. Each action has the form

$$\langle \text{guard} \rangle \longrightarrow \langle \text{assignment statement} \rangle . \quad (1)$$

A guard is a Boolean expression over variables. An assignment statement updates one or more variables. A state is defined by a value for every variable in the program, chosen from the predefined domain of that variable. An action whose guard is true at some state is said to be enabled at that state and is executed.

2.3. Clustering. Each node runs an underlying clustering algorithm *FLOC*. By doing so, within $O(1)$ time (independent of the network size), each node j becomes a clusterhead or joins an existing cluster. The resulting clusters satisfy a solid-disk clustering property in which all nodes within unit distance of a clusterhead are required to be part of the same cluster and nodes up to a distance 2 may be within the same cluster. The actions in FLOC that result in these properties are briefly reviewed in Section 2.6.

2.4. Problem Statement. Before stating the objective, we first note the following definition for *neighboring clusters*.

Definition 1 (neighboring clusters). Two clusters X and Y are neighboring if there exist nodes i and j within i-band of each other such that $i \in X \wedge j \in Y$.

The objective in this paper is to utilize the underlying clustering and create a spanner in the network which ensures the following properties:

(i) all *neighboring clusters* are connected and thus the graph is connected;

(ii) if two nodes i and j connect clusters $C1$ and $C2$, then it is required that i and j are within i-band of each other;

(iii) if two nodes i and j connect clusters $C1$ and $C2$, then there are no other connections between clusters $C1$ and $C2$ using nodes that lie within the i-band of i or j.

These properties ensure that the network graph is connected using reliable links while also ensuring that unnecessary connections between neighboring clusters are avoided. We show in our analysis that this results in retaining only $O(N)$ edges in the spanner while also ensuring a bounded path length ratio over the complete graph.

2.5. Fault Model. Nodes may fail, stop, and crash and new nodes may join the network. The states of a node may be arbitrarily corrupted. In addition, messages may be lost. These faults can occur in any finite number, at any time, and in any order. A program is self-stabilizing if and only if, after faults stop occurring, the program eventually recovers to a state from where its specification is satisfied. A self-stabilizing program is fault-local self-stabilizing if the time and number of messages required for stabilization are bounded by functions of perturbation size rather than network size [1]. The perturbation size for a given state is the minimum number of nodes whose state must change to achieve a consistent state of the network.

2.6. FLOC Review and Addition of Spanner State. FLOC (fault-local clustering) [1] partitions a multihop wireless network into nonoverlapping and approximately equal-sized clusters such that all nodes within a certain radius around each clusterhead, necessarily, belong to that cluster. While maintaining this *solid-disk* property, *FLOC* also ensures that node additions and deletions do not result in a global reformation of clusters by allowing a dilation factor m of at least 2 in the size of each cluster. The basic FLOC actions are briefly reviewed here for completion. Each node can be in one of the following 5 states: idle, candidate, clusterhead, i-band, or o-band.

Action 1. When the node has been *idle* for some random time, it changes its state to *candidate* (i.e., becomes a potential candidate for becoming the clusterhead) and broadcasts a candidacy message.

Action 2. If the *candidacy* message is received by a node in the i-band region of the sender and the receiver is already an i-band member of some existing cluster, the receiver sends a *conflict* message.

Action 3. If the candidate node receives the conflict message for its candidacy request, it becomes an o-band member of the cluster of the node which sent the message.

Action 4. When the candidate node does not receive a conflict message within a certain predefined time the node becomes a clusterhead and announces the same.

Action 5. Any node in the *idle* state that receives a clusterhead message becomes an i-band or o-band member of the cluster based on determination of i-band/o-band status of the sender.

Action 6. A node in o-band region of some cluster receives leader message and determines that it is in i-band range of the node which sent the message and joins that cluster as i-band member.

FLOC thus exploits the *atomic* broadcast property of wireless networks to enable unique election of clusterheads. By atomically informing the neighbors of its intention to become a clusterhead through a broadcast message, a node is able to lock itself into the position of a clusterhead unless it is notified of conflicts explicitly. It has been shown in [1] that these actions result in creation of solid-disk clusters within $O(1)$ time. Furthermore, with addition of periodic heartbeat messages, the program is also shown to locally self-stabilize from topology changes and message losses.

The original actions of *FLOC* only focused on forming cluster membership; in this paper we are also interested in realizing a connected subgraph which is a spanner for the graph. Hence, after becoming a cluster member we add a variable $j \cdot cp$ at each node that consists of the node leading towards the clusterhead from itself (*the cluster-parent for node* j). If j is a direct neighbor of its clusterhead $j \cdot H$, then $j \cdot cp = j \cdot H$. If j is connected to $j \cdot H$ through a node k that is an i-band neighbor of $j \cdot H$ and j, then $j \cdot cp = k$. For a clusterhead $j \cdot cp = \perp$.

3. FLOC-SPANNER Protocol

In this section we describe the FLOC-SPANNER protocol and analyze its correctness. The actions are described for a given node j. The actions of this protocol at any node j are enabled as soon as the node becomes either an i-band or o-band member of some cluster using the FLOC protocol. Clusterhead nodes do not execute the actions of this protocol. Thus, a common precondition for all the guards in this program is that the node is either an i-band or o-band member. Note that all messages that are required for FLOC-SPANNER are piggybacked on periodic heartbeat messages sent out by every node to maintain cluster status. Thus there is no additional messaging overhead incurred.

3.1. Variables. Each node j maintains the following variables.

(i) $j \cdot H$: it is the cluster id for node j which is equal to the id of its clusterhead.

(ii) $j \cdot \gamma$: it specifies the current state of the node in spanner connection establishment. There are 4 possible states.

(a) Monitor: in this state, the node is not competing to establish a connection with any other node.

(b) Conncand: in this state, the node is ready to compete for connection establishment and is waiting for the next heartbeat message to send the request.

(c) Initiator: this is the *verification* state for a node that has sent a request for connection establishment and is waiting to learn about conflicts, that is, other existing connectors that are within its i-band range.

(d) Receptor: this is the *verification* state of a node which has received a request for connection establishment and is waiting to learn about conflicts, that is, other existing connectors that are within its i-band range.

(iii) $j \cdot \phi_x$: it is a Boolean variable which is set to be true if the node has a connection to the cluster x.

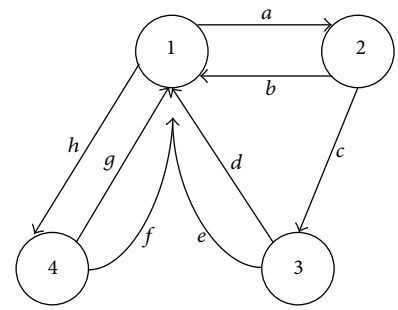
(iv) $j \cdot \nu_x$: it specifies the id of the node belonging to the cluster x to which connection has been established.

(v) $j \cdot Z$: it is a set of clusters for which node j is a connector, that is, the set of clusters x for which $j \cdot \phi_x$ is *true*.

(vi) $j \cdot \kappa_C$: it is the id of the cluster with which a potential connection is being initiated.

(vii) $j \cdot \kappa_n$: it is the id of the node with which a potential connection is being initiated.

(viii) $j \cdot L$: it is a list of clusters that node j is aware of and which are connected to j's cluster through a node within i-band range of $j \cdot j \cdot L_n(C)$ is used to denote the connector for cluster C in this list and $j \cdot L_\tau(C)$ denotes the time of last heard heartbeat from $j \cdot L_n(C)$.



(a)

1	Monitor
2	Conncand
3	Initiator
4	Receptor

(b)

A	Detect new neighboring cluster
B	Learn about existing connection or connection request
C	Timeout in connection candidacy state
D	Learn about conflict and cancel connection
E	Timeout in initiator state: establish connection
F	Timeout in receptor state: establish connection
G	Learn about conflict and cancel connection
H	Receive connection initiation request

(c)

Figure 2: (a) State transition diagram, (b) list of states, and (c) list of transition events: note that events e and f are expected to happen concurrently at nodes in the initiator and receptor states corresponding to a connection.

3.2. Program. We describe the algorithm by grouping together the guard-action pairs in each of the 4 states. The algorithm is also formally stated in the form of guard-action pairs in each state. Each node maintains a periodic heartbeat timer to send out a heartbeat message and all information is piggybacked into these heartbeat messages. Heartbeat messages are classified into the following types: T_REGULAR, and T_INITIATE, T_CONFLICT. The message type is included as one of the fields in the heartbeat message. There is no extra overhead for the protocol to both create and maintain the spanner. Note that the message reception event in the following discussion is assumed to be enabled only when the sending node is within i-band range of the receiving node. The state transition diagram for the program is shown in Figure 2.

3.2.1. Monitor State. The actions executed in the monitor state are shown in Algorithm 1 using guard-action pairs. A

Node j **in MONITOR State**
Action M1: $timeout(HB\_timer) \rightarrow$
$bcast(HB\_msg_j(j, j \cdot H, j \cdot Z, T\_REGULAR)$

Action M2: $rcv(HB\_msg_k()) \rightarrow$
if $j \cdot H == k \cdot H$ **then**
  for all x in $k \cdot Z$ **do**
    if $x \in j \cdot L$ **then**
      update $j \cdot L_\tau(x)$ to current time
    else
      Add x to $j \cdot L$
    end if
  end for
end if

if $(type! = T\_INITIATOR) \wedge (j \cdot H! = k \cdot H) \wedge (k \cdot H \notin j \cdot L)$ **then**
  $\kappa_c = k \cdot H$; $\kappa_n = k$; $\gamma := conncand$;
end if

if $(type == T\_INITIATOR) \wedge (j \cdot H! = k \cdot H) \wedge (k \cdot H \notin j \cdot L) \wedge (k \cdot \kappa_n == j)$ **then**
  $\kappa_c = k \cdot H$; $\kappa_n = k$; $\gamma := receptor$;
  $reset(HB\_timer)$;
end if

Algorithm 1: FLOC-SPANNER actions at node j in *monitor* state.

node is in the *monitor* state whenever it is not competing to form a connection with a neighboring cluster. In the *monitor* state, a node periodically beacons its heartbeat message announcing its current state. The transition of a node to other states is guided based on the messages that are received. These transitions are summarized below.

(1) Heartbeat Timer Timeout. In the monitor state, when the heartbeat timer expires it sends a heartbeat message which contains its cluster id ($j \cdot H$) and a set of all clusters for which it is the connector (i.e., $j \cdot Z$).

(2) Receive Heartbeat Message. The actions taken by the node when it receives a heartbeat message depend on the state of the node from which it is received. The actions are grouped as follows.

(i) Receive heartbeat from node k which belongs to the same cluster as j (i.e., $j \cdot H = k \cdot H$): if node k is connector to the same cluster X, node j checks its list $j \cdot L$ to see if X belongs to $j \cdot L$. If so, the last heartbeat time for X, that is, $j \cdot L_\tau(X)$, is updated. Otherwise, cluster X is added to $j \cdot L$ along with information about the connecting node k and the last heartbeat time.

(ii) Receive heartbeat from node k belonging to different cluster and the message is of type T_REGULAR: node j checks if there is already another connector to k's cluster within j's i-band range; that is, it checks if $k \cdot H$ belongs to $j \cdot L$. If this condition is not satisfied node j does the following.

  (a) It becomes a candidate to establish connection with k's cluster via k and changes its state to *conncand*.

  (b) The node id k and its cluster $k \cdot H$ are recorded into κ_n and κ_c, respectively.

(iii) Receive heartbeat from node k belonging to different cluster and the message is of type T_INITIATE: when the node j receives the connection initiation message it does the following provided that there is no other connector to k's cluster within j's i-band range; that is, $k \cdot H$ does not belong to $j \cdot L$.

  (a) The node changes its state to *receptor*.

  (b) The node id k and its cluster $k \cdot H$ are recorded into κ_n and κ_c, respectively.

  (c) The node resets its heartbeat timer so that it gets one full heartbeat interval to learn if there are other confirmed connections within its i-band range.

3.2.2. Conncand State. The actions executed in the conncand state are shown in Algorithm 2 using guard-action pairs. A node moves to the *conncand* state when it learns about a new neighboring cluster and is not aware of existing connections to that cluster. In the *conncand* state, it waits until the next heartbeat timeout to initiate a connection with the neighboring cluster. Until that time it listens to heartbeat messages from neighbors to learn about already existing connections and connection requests from other nodes in its cluster. These actions are summarized below.

(1) Receive Heartbeat Messages. During the *conncand* state, node j checks all incoming heartbeat messages if there are existing connections with κ_c through a connector within i-band range of j. Even if a piggybacked connection request

Node j in CONNCAND State
Action C1: $timeout(HB\_timer) \rightarrow$
$\gamma := initiator$;
$bcast(HB\_msg_j(j, j \cdot H, j \cdot Z, T\_INITIATOR, j \cdot \kappa_n, j \cdot \kappa_c))$;

Action C2: $rcv(HB\_msg_k) \rightarrow$
if $(j \cdot \kappa_c == k \cdot H \wedge j \cdot H \in k \cdot Z) \vee (j \cdot H == k \cdot H \wedge j \cdot \kappa_c \in k \cdot Z)$ **then**
  $\gamma := monitor$; $\kappa_c = \perp$; $\kappa_n = \perp$;
end if

if $type == T\_INITIATOR \wedge j \cdot H == k \cdot H \wedge j \cdot \kappa_c == k \cdot \kappa_c$ **then**
  $\gamma := monitor$; $\kappa_c = \perp$; $\kappa_n = \perp$;
end if

Algorithm 2: FLOC-SPANNER actions at node j in *conncand* state.

Node j in INITIATOR State
Action I1: $timeout(HB\_timer) \rightarrow$
$\phi_{\kappa_c} := true$;
$\nu_{\kappa_c} = \kappa_n$;
Add κ_c to $j \cdot L$ and $j \cdot Z$;
$bcast(HB\_msg_j(j, j \cdot H, j \cdot Z, T\_REGULAR))$;
$\gamma := monitor$; $\kappa_c = \perp$; $\kappa_n = \perp$;

Action I2: $rcv(HB\_msg_k) \rightarrow$
if $(j \cdot \kappa_c == k \cdot H \wedge j \cdot H \in k \cdot Z) \vee (j \cdot H == k \cdot H \wedge j \cdot \kappa_c \in k \cdot Z)$ **then**
  $bcast(HB\_msg_j(j, j \cdot H, j \cdot Z, T\_CONFLICT, j \cdot \kappa_n))$;
  $\gamma := monitor$; $\kappa_c = \perp$; $\kappa_n = \perp$;
  $reset(HB\_timer)$;
end if

If $k == j \cdot \kappa_n$ **then**
  $\gamma := monitor$; $\kappa_c = \perp$; $\kappa_n = \perp$;
end if

Algorithm 3: FLOC-SPANNER actions at node j in *initiator* state.

is heard by j, node j cancels its candidacy and moves back to *monitor* state. This action ensures that only one node within an i-band range can compete in the connection establishment at one time to the same cluster.

(2) Heartbeat Timer Timeout. When the node heartbeat timer times out and it is in conncand state the node j does the following.

(a) It sends heartbeat message and piggybacks a connection request to the node κ_n for establishing connection with κ_c.

(b) It changes its state to *initiator*.

3.2.3. Initiator State. The actions executed in the initiator state are shown in Algorithm 3 using guard-action pairs. The *initiator* state represents the state where a node has initiated a connection request with a node in its neighboring cluster. It waits for an entire heartbeat interval to learn about conflicting connections between the same pair of clusters by listening to heartbeat messages. Only if no conflicts are detected, the node will confirm the connection and move to the *monitor* state. Otherwise, the connection request will be canceled. The actions of a node in in the *initiator* state are summarized below.

(1) Receive Heartbeat Messages. During the *initiator* state, node j checks all incoming heartbeat messages if there are existing connections with κ_c through a connector within i-band range of j. The actions of the node j depend upon the state of the node from which the heartbeat message is received. Accordingly, the actions are grouped as follows.

(i) Receive heartbeat from node k which belongs to the same cluster as j (i.e., $j \cdot H = k \cdot H$): if $j \cdot \kappa_c$ belongs to $k \cdot Z$, that is, if node k has a connector to the cluster with which node j is waiting to establish connection, then node j does the following.

(a) It sends heartbeat message indicating conflict to the receptor node stored in the κ_n.

(b) It resets its heartbeat timer.

(c) It changes its state to *monitor*.

Node j **in RECEPTOR State**
Action R1: $timeout(HB\_timer) \rightarrow$
$\phi_{\kappa_c} := true$;
$\nu_{\kappa_c} = \kappa_n$;
Add κ_c to $j \cdot L$ and $j \cdot Z$;
$bcast(HB\_msg_j(j, j \cdot H, j \cdot Z, T\_REGULAR))$;
$\gamma := monitor$; $\kappa_c = \perp$; $\kappa_n = \perp$;

Action R2: $rcv(HB\_msg_k) \rightarrow$
if $(j \cdot \kappa_c == k \cdot H \wedge j \cdot H \in k \cdot Z) \vee (j \cdot H == k \cdot H \wedge j \cdot \kappa_c \in k \cdot Z)$ **then**
$bcast(HB\_msg_j(j, j \cdot H, j \cdot Z, T\_CONFLICT, j \cdot \kappa_n))$;
$\gamma := monitor$; $\kappa_c = \perp$; $\kappa_n = \perp$;
$reset(HB\_timer)$;
end if

if $k == j \cdot \kappa_n \wedge type == T\_CONFLICT$ **then**
$\gamma := monitor$; $\kappa_c = \perp$; $\kappa_n = \perp$;
end if

Algorithm 4: FLOC-SPANNER actions at node j in *receptor* state.

(ii) Receive heartbeat from node k belonging to κ_c: if $j \cdot H$ belongs to $k \cdot Z$, that is, if node k belonging to the cluster with which node j is waiting to establish connection already has a connector to j's cluster, then node j does the following.

(a) It sends heartbeat message indicating conflict to the receptor node stored in the κ_n.

(b) It resets its heartbeat timer.

(c) It changes its state to *monitor*.

(iii) Receive heartbeat from node κ_n: this implies that the node with which the connection is being established has sent a premature heartbeat (without waiting for one heartbeat interval). This, in turn, implies the existence of a conflict for one of the following reasons and hence node j returns to the monitor state.

(a) Node κ_n has detected a conflicting connection within its i-band range between the same pair of clusters.

(b) Node κ_n has not accepted the connection request and hence not moved into receptor state (potentially because the initiator message was lost).

(c) Node κ_n has joined some other clusters because of reclustering by FLOC protocol.

The variables κ_n and κ_c are reset. The subsequent heartbeat message from j will thus not indicate connection with κ_n.

(2) Heartbeat Timer Timeout. When the heartbeat timer of a node which is in initiator state times out while node j is still in the *initiator* state, it means that the node j has not learned about any other connectors to κ_c within this interval and the node κ_n also has not sent any conflicting connections within its i-band range. So, the node j confirms the connection to κ_n in a heartbeat message and marks the boolean variable ϕ_x to *true*, where $x = \kappa_c$. The variables κ_n and κ_c are reset.

3.2.4. Receptor State. The actions executed in the receptor state are shown in Algorithm 4 using guard-action pairs. A node moves to the *receptor* state when it receives a connection initiation from a node in the neighboring cluster. It waits for an entire heartbeat interval to learn about conflicting connections between the same pair of clusters by listening to heartbeat messages. Only if no conflicts are detected, the node will confirm the connection and move to the *monitor* state. Otherwise, it will send a heartbeat message indicating conflict and reset its heartbeat interval.

(1) Receive Heartbeat Messages. During the *receptor* state, node j checks all incoming heartbeat messages if there are existing connections with κ_c through a connector within i-band range of j. The actions of the node j depend upon the state of the node from which the heartbeat message is received. Accordingly, the actions are grouped as follows.

(i) Receive heartbeat from node k which belongs to the same cluster as j (i.e., $j \cdot H = k \cdot H$): if $j \cdot \kappa_c$ belongs to $k \cdot Z$, that is, node k has a connector to the the cluster to which node j is waiting to establish connection, then node j does the following.

(a) It sends heartbeat message indicating conflict to the initiator node stored in the κ_n.

(b) It resets its heartbeat timer.

(c) It changes its state to *monitor*.

(ii) Receive heartbeat from node k belonging to κ_c: if $j \cdot H$ belongs to $k \cdot Z$, that is, if node k belonging to the cluster with which node j is waiting to establish connection already has a connector to j's cluster, then node j does the following.

(a) It sends heartbeat message indicating conflict to the initiator node stored in the κ_n.

(b) It resets its heartbeat timer.

(c) It changes its state to *monitor*.

(iii) Receive heartbeat from node κ_n: this implies that the node which initiated the connection has sent a premature heartbeat (without waiting for one heartbeat interval). This, in turn, implies the existence of a conflict for one of the following reasons and hence node j returns to the monitor state.

(a) Node κ_n has detected a conflicting connection within its i-band range between the same pair of clusters.

(b) Node κ_n has joined some other clusters because of reclustering by FLOC protocol. The variables κ_n and κ_c are reset. The subsequent heartbeat message from j will thus not indicate connection with κ_n.

(2) Heartbeat Timer Timeout. When the heartbeat timer of a node which is in receptor state times out while node j is still in the *receptor* state, it means that the node j has not learned about any other connectors to κ_c within this interval and the node κ_n also has not sent any conflicting connections within its i-band range. So, the node j confirms the connection to κ_n in a heartbeat message and marks the boolean variable ϕ_x to *true*, where $x = \kappa_c$. The variables κ_n and κ_c are reset. Note that, at the same time, the node κ_n would also mark node j as a connector to cluster $j \cdot H$, because otherwise a heartbeat message indicating a *conflict* would have been received.

3.3. Analysis. The FLOC-SPANNER protocol creates a geometric spanner by connecting all pairs of neighboring clusters created by *FLOC*. In this section, we analyze the correctness of the FLOC-SPANNER protocol and provide bounds on completion time and the path stretch factor. We first state the invariants for the program.

Lemma 2. *The following invariant holds for FLOC-SPANNER.*

(I1) *For any two nodes i and j belonging to the neighboring clusters x and y* $(i \cdot \phi_y = true \wedge i \cdot \nu_y = j) \equiv (j \cdot \phi_x = true \wedge j \cdot \nu_x = i)$.

(I2) *Given nodes i, j, and k such that i and j lie within i-band of each other and belong to the same cluster x, and node k belongs to a neighboring cluster y* $(i \cdot \phi_y = true) \equiv (j \cdot \phi_y = false)$.

Proof. (I1) states that for a connection to be successful both the nodes involved in the connection should connect to each other; that is, if node i of one cluster, connects to the node j of neighboring cluster, then node j should also connect to the node i. This ensures that one-sided connections will be avoided. In the FLOC-SPANNER protocol, as soon as a node discovers a neighboring cluster that is not in its list $j \cdot L$, it becomes a candidate for forming a connection. Multiple nodes may become a candidate upon learning about a new cluster through a heartbeat from a common node. The node, whose heartbeat timer first expires, *wins* the candidacy and moves to the initiator state. Upon becoming an initiator, the node notifies the intended connecting node which becomes a receptor almost atomically (only separated by a message transmission time). After this, both initiator and the corresponding receptor nodes wait for one heartbeat interval to check for conflicting connections between the same pair of clusters within their respective i-band region. If no conflicting messages are heard, both the nodes simultaneously set their states to indicate connection after one heartbeat interval. This ensures that (I1) is held.

(I2) states that if there is a connection between two clusters, then there exists no other connection between the same clusters within the inner-band range of the nodes that are involved in the connection. This property ensures that the number of pairwise connections used to establish the spanner graph is minimized. In the FLOC-SPANNER protocol, all nodes that learn about a new cluster through a heartbeat message move to the *conncand* state. However, However, only the node whose heartbeat timer first expires (say node i), initiates this connection. Other candidate nodes that hear a connection initiation or learn about an already existing connection between the clusters cancel their intention to form a connection. Furthermore, once an intention to form a connection has been announced, both the initiator and the corresponding receptor node (say node j) wait concurrently for one heartbeat interval to learn about connections within their respective i-band region. If such a conflicting connection exists, the nodes cancel their upcoming connection by sending a conflict message. Likewise, if any other node had initiated a connection between the same pair of clusters *after* the node i initiated the connection, the subsequent heartbeats sent out by i and j after confirmation of the connection will prevent the new initiation from succeeding. Thus, if a connection is established through a pair of nodes, then there exist no other connections between the same clusters through nodes within i-band region of either of these nodes. □

Note that the above invariants assume that all messages are successfully received. In the presence of message losses, the invariants may be violated. For example, when the heartbeat message with an initiation request is lost, a partial connection may be established. When a heartbeat message with a conflict notification is lost, there could be duplicate connections within i-band region of each other. Handling of these invariant violations and self-stabilization to the invariant states are incorporated through monitoring of invariant states and recovery by means of the fault-tolerance actions that are described in Section 3.4.

Theorem 3. *Irrespective of network size, the FLOC-SPANNER protocol establishes connections between all pairs of neighboring clusters in* $O(1)$ *time.*

Proof. First, consider a given pair of neighboring clusters x and y, that is, clusters which have at least one pair of nodes i and j, where $i \in x \wedge j \in y$ such that i and j are within i-band of each other. Once the clusters have been formed and memberships for i and j have been established, the nodes

in either clusters will discover each other within 1 heartbeat interval and move to *conncand* state for forming connections. The node in either of these clusters whose heartbeat timer first expires after moving to conncand state will initiate the connection request by moving to the *initiator* state and sending a heartbeat message. Within 1 heartbeat interval of this stage, a connection will be established between the two clusters.

Secondly, if all pairs of neighboring clusters were connected by a different pair of connector nodes, then all the connection establishment can take place concurrently, thus terminating the process in $O(1)$ time. Even if a given node was involved in connections with multiple neighboring clusters, we note that there can only be a bounded number of clusters whose members are within i-band range of the node. This property follows from the solid-disk property of the underlying clustering which guarantees that each cluster is at least of a unit radius around the clusterhead. Thus the connection establishment process will terminate in $O(1)$ time. □

Theorem 4. *Let $S = (V, E)$ denote a subgraph of the network in which V is the set of all nodes in the network and $E = E1 \cup E2$, where $E1$ is the set of links between j and $j \cdot cp$ for all nodes j in the network and $E2$ is the set of connector edges created by FLOC-SPANNER. Then S yields a spanner for the network graph; that is, all nodes are connected through edges in S.*

Proof. Recall from the description of *FLOC* in Section 2.6 that, for each node that is a member of a cluster, $j \cdot cp$ denotes the *cluster-parent*, that is, the id of the node that leads j towards the clusterhead. If j is a direct neighbor of its clusterhead $j \cdot H$, then $j \cdot cp = j \cdot H$. If j is connected to $j \cdot H$ through a node k that is an i-band neighbor of $j \cdot H$ and j, then $j \cdot cp = k$. For a clusterhead $j \cdot cp = \perp$. Since all nodes belong to the same cluster, the set $E1$ which comprises the links between j and $j \cdot cp$ provides a connection between every node and its respective clusterhead. Now the *FLOC-SPANNER* protocol ensures that all neighboring clusters are connected through the edges in the set $E2$. Thus the subgraph $S = (V, E)$ where $E = E1 \cup E2$ yields a spanner in which there exists a path between every pair of nodes in the network. □

Theorem 5. *The spanner graph $S = (V, E)$ created by the FLOC-SPANNER protocol is a geometric spanner with path stretch factor bounded by $5/2 + 4/d$.*

Proof. Let d_{ij} denote the *hop* distance between any two nodes i and j in the original network graph. Let $d_{ij} > 1$; that is, i and j are not direct neighbors (in which case they can directly exchange messages and communicate irrespective of the spanner graph). Let i_c and j_c denote the clusters of nodes i and j, respectively. Note that by the sold-disk property of the underlying clustering, the minimum diameter of each cluster is 2 units and the maximum diameter is $2m$ units (where $m \geq 2$ in order to ensure local self-stabilization of cluster formation). In this paper, we have considered $m = 2$, yielding a maximum diameter of 4 units. Given that the minimum diameter for each cluster is 2 units, the maximum number of cluster pairs that need to be traversed between i and j is bounded by $d/2$. The distance between neighboring clusterheads through edges in S is bounded by 5 (if each cluster is of diameter 4 units). The maximum distance from each node to its clusterhead is 2 units. Thus the total distance between i and j using only edges in S is bounded by $5d/2 + 4$. The path stretch factor is thus bounded by $5/2 + 4/d$. Thus, we observe that the path stretch factor has maximum of 4.5 for $d = 2$ and improves for nodes that are at a greater separation. □

3.4. Fault-Tolerance Actions. Topology changes and message losses can cause the protocol's invariants to be violated. For example, (i) when nodes are removed, existing connections between neighboring clusters may be broken; (ii) when a heartbeat message with a connection initiation request is lost, a partial (one-sided) connection may be established; (iii) when a conflict message is lost, multiple connections may be established within the same pair of clusters that are within i-band of each other. To handle these invariant violations and to guarantee self-stabilization to the invariant states, we introduce the following actions which involve monitoring of the system state to detect invariant violation. These fault-tolerance actions are shown in Algorithm 5 using guard-action pairs.

(1) A *connection liveness timer* is used to monitor the liveness of existing connections to neighboring clusters. The interval CT of this timer is set to a value p times the heartbeat interval where $p \geq 2$. Note that, in the FLOC-SPANNER protocol actions, whenever a node hears a heartbeat message from a connector node within its cluster (by checking against $j \cdot L$), it updates the timestamp for the last heartbeat. Whenever the connection liveness timer expires in any state, a node checks the last heartbeat time for all active connections in $j \cdot L$. If they are greater than CT, then the particular connection is removed from the list.

(2) Violation of invariants due to message losses can be detected by checking for inconsistencies in the connection information. Specifically, whenever a node that is a connector to a cluster X receives a heartbeat message from a node k that belongs to the same cluster as j (i.e., $j \cdot H = k \cdot H$) and is a connector to the same cluster X, a violation is detected. A heartbeat message with a type *T_VIOLATION* is sent out by j causing all nodes within i-band range to reset their state with respect to cluster X, including node k. This causes a new election of connectors to take place through the regular actions for FLOC-SPANNER. The heartbeat timer is reset at node j.

4. Performance Evaluation

We evaluate the performance of *FLOC-SPANNER* using JProwler, a Java based discrete event simulator for wireless sensor networks. We implement the underlying clustering protocol *FLOC* as well as the actions for the *FLOC-SPANNER* protocol as discussed in this paper. We use a static grid

Action S1: $timeout(CT\_timer)$ $\rightarrow$
for all x in $j \cdot L$ **do**
  if $now - j \cdot L_\tau(x) > p$ **then**
    remove x from $j \cdot L$ and $j \cdot Z$
    $\phi_x := false$
  end if
end for

Action S2: $rcv(HB\_msg_k)$ $\rightarrow$
if $j \cdot H == k \cdot H$ **then**
  for all x s.t $(j \cdot \phi_x == true \wedge k \cdot \phi_x == true)$ **do**
    remove x from $j \cdot L$ and $j \cdot Z$
    $\phi_x := false$
    $bcast(HB\_msg_j(j, j \cdot H, j \cdot Z, T\_VIOLATION, x))$;
    $reset\ (HB_{timer})$;
  end for
end if

Action S3: $rcv(HB\_msg_k)$ $\rightarrow$
if $j \cdot H == k \cdot H \wedge type == T\_VIOLATION$ **then**
  remove x from $j \cdot L$ and $j \cdot Z$
  $\phi_x := false$
end if

Algorithm 5: Fault-tolerance actions for FLOC-SPANNER.

topology and consider networks of different densities and different sizes. Specifically, we have simulated networks of sizes 400, 900, 1600, and 2025. For each network size, we have considered deployment densities with 5, 7, and 9 nodes per unit communication area. We denote these densities as $D1$, $D2$, and $D3$, respectively. Our goal is to evaluate the convergence characteristics of *FLOC-SPANNER* and to measure the number of spanner edges and path stretch factor as a function of network size and density. We have used a heartbeat interval of 5 seconds and a connection liveness interval of 15 seconds. We describe our observations in the following subsections.

4.1. Convergence Time. To compute the convergence time for the protocol, we measure the number of connected paths that exist in the network at intervals of 1 second. Given that the network is connected, the total number of paths in the system is $N * (N - 1)$. We define the *convergence ratio* at any time as the ratio of the number of paths that exist in the system between all pairs of nodes, to the total number of possible paths (i.e., $N * (N - 1)$). When all the clusters are created and connections have been established between the clusters using the *FLOC-SPANNER* protocol, we expect the number of valid paths in the network to be $N * (N - 1)$ and the convergence ratio to be 1.0. Once the convergence ratio reaches 1.0, we expect the number of paths to slightly fluctuate in that range because of transient message losses. In Figure 3(a), we show the convergence ratio attained in the network as a function of simulation time, for different network sizes. As seen in this figure, the convergence ratio first reaches 1.0 at approximately the same time irrespective of network size and this number is about 2 to 3 times the heartbeat interval from the time that the system is initialized. Note that clustering itself is expected to take between 0 and 5 seconds and the expected time for connections to be established is bounded by 2 times the heartbeat interval.

Figure 3(a) is shown for the density model $D1$ (i.e., 5 nodes per unit communication area). Similar convergence graphs are obtained under all three density models. The time at which the convergence ratio first reaches 1.0 is taken as the convergence time for the protocol. In Figure 3(b), we show the convergence time as a function of network size, at different network densities. We observe that the convergence time for the protocol stays steady, irrespective of network size and density highlighting the scalability of our protocol.

4.2. Path Stretch Factor. To compute the path stretch factor, we compute the length of the shortest path using Dijkstra's algorithm on the original network graph and then on the spanner graph resulting from our protocol. The ratio of these lengths is taken as the path stretch factor. The ratio is computed only for paths that exist on the spanner graph. In Figure 4(a), we observe the variation in path stretch factor over time for the density model $D1$ (i.e., 5 nodes per unit communication area), under different network sizes. Between 0 and about 5 seconds after initialization, clusters are still being formed in the underlying network and formation of cluster connections has not started. Hence the path stretch factor is zero. After this time, the number of pairwise node paths in the system begins to increase and the path stretch factor rises. When the total number of paths in the system reaches 100% (i.e., around 12–14 seconds from Figure 3(a)), we observe that the path stretch factor is around 2 to 3.3. In this phase, all the paths have been created for the first time but these paths have not really stabilized. We observe that around the 20-second mark, the path stretch factor stabilizes to values in the range of 1.4 to 1.8 and then stays in that range. We then repeat the computation of path stretch factor

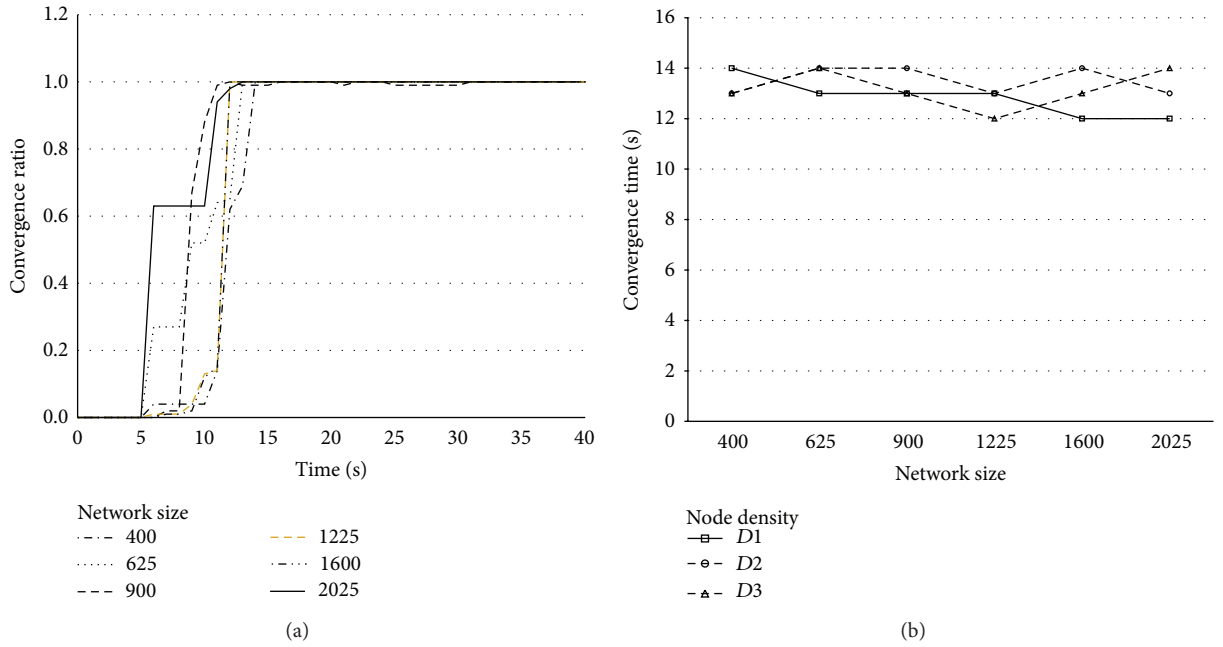


FIGURE 3: (a) Convergence ratio as a function of time for different network sizes. (b) Average convergence time as a function of network size for different network densities.

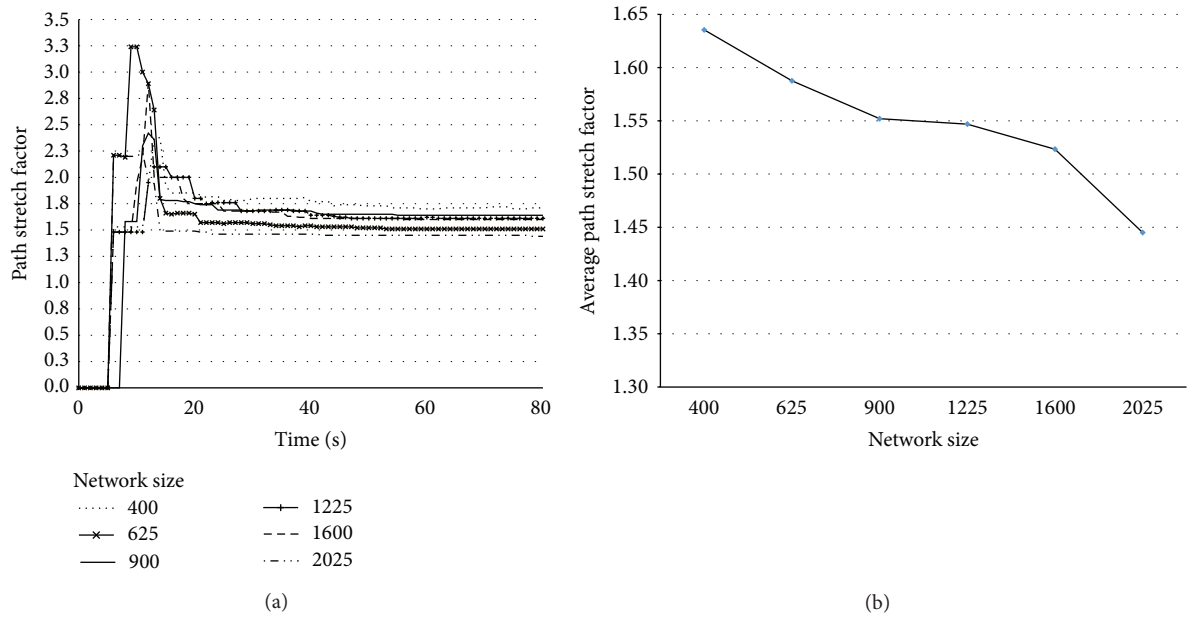


FIGURE 4: (a) Path stretch factor as a function of time for different network sizes. (b) Path stretch factor as a function of network size, averaged over different densities.

at all three density models. In Figure 4(b), we plot the path stretch factor for network of different sizes, where the values for the path stretch factor at a given size are averaged over the different density models. From Figure 4(b), we observe that larger networks experience slightly lower average path stretch factors, validating the scalability of our approach.

4.3. Spanner Edges. In Figure 5(a), we show the total number of spanner edges in the system as a function of simulation time for the density model $D1$. The number of spanner edges steadily rises as clusters and cluster connections are formed and remain steady after convergence. We repeat this computation for different network densities. In Figure 5(b), we show the number of spanner edges as a function of network size at different densities. First, we observe that the number of spanner edges grows only as $O(N)$, as opposed to $O(N * d)$ in the original graph, where d is the degree of each node. Next, we observe that the number of spanner edges

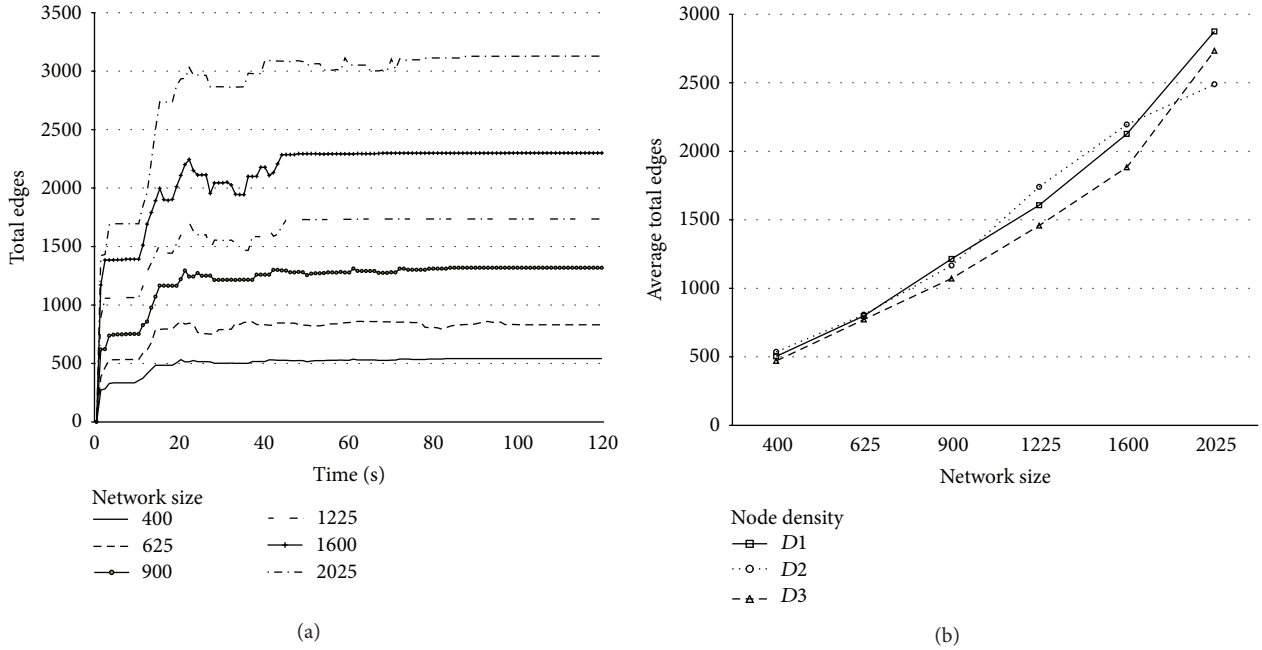


Figure 5: (a) Number of spanner edges as a function of time for different network sizes. (b) Number of spanner edges as a function of network size for different network densities.

remains steady irrespective of network density. Thus the density does not adversely affect the creation of cluster connections, thereby validating that the *FLOC-SPANNER* protocol remains scalable irrespective of network density.

5. Related Work

Designing algorithms for creation of spanners is a well-researched topic and some detailed surveys can be found in [6–8]. A brief summary is presented below.

From the perspective of wireless ad hoc networks, it is important to realize spanner structures in a distributed and local manner. In that context, relative neighborhood graphs [9] and Gabriel graphs [10] are examples of *proximity graphs* that can be realized in a distributed manner. However, both of these do not yield geometric spanners (constant bound on path stretch factor). Yao graphs [11] are an elegant generalization of proximity graphs that can be constructed locally and yield geometric spanners. The idea in Yao graphs is for each node to partition the space around it into sectors of angle $\pi/3$ and retain the edge to the closest node in each sector, thus allowing local selection. Refrences [12–14] propose modifications to the Yao graph that result in bounding the maximum degree. Reference [15] proposes an extension to Yao graphs that results in minimizing the transmission range at each node. However, in wireless networks, Yao graphs and their variations may not result in reduction in the number of edges as each node independently chooses a certain set of neighbors (the number of edges in the spanner may still be of the order $O(Nd)$, where d is the average degree of each node).

Delaunay triangulation of a network graph, a set of edges such that for each edge there is a circle containing the edge end points but not containing any other points, also yields a geometric spanner. However, a Delaunay triangulation graph could potentially require inclusion of edges that are longer than the transmission range (not feasible in a wireless sensor network). Hence restricted Delaunay graphs (RDGs) and variations of RDG (such as localized Delaunay triangles) have been used in the context of wireless networks for localized spanner creation. RDGs utilize only local communication links and result in geometric spanners. By utilizing RDGs, techniques in [16–18] produce spanner that contain only $O(N)$ edges. However, all of these techniques utilize location information for creation of spanners. In this paper, we do not assume localization for creation of spanners, making the system easy to deploy and our algorithm terminates in $O(1)$ time. However, we do note that several of these approaches also focus on ensuring planarity of the spanner graphs, which is not a goal in our paper.

The idea of first creating clusters and then connecting them to create geometric spanners has been exploited in [19]. Without the requirement of planarity, such an approach does not require localization and the idea in *FLOC-SPANNER* is along the same lines. However, the key difference arises from the process of creating the intercluster connections in a self-stabilizing manner with very little message overhead. The technique in [19] relies on first building the two-hop neighborhood of each node using synchronous rounds of communication, after which the intercluster connection is *atomically* established using some criteria such as nodes with minimum node id or maximum battery levels. Reestablishing and maintaining this structure in the presence of node additions/deletions and clustering changes are not trivial in this model and have not been discussed. In contrast, our

solution is asynchronous and each node nominates itself as a candidate upon learning about any new cluster in its neighborhood. Yet, we ensure that there are no duplicate connections within the communication range of connector nodes by overhearing heartbeat messages and signaling a conflict before confirmation of connection. The proposed algorithm is thus able to dynamically react to topology changes, has lower memory requirement, converges in $O(1)$ time (including cluster formation), and is shown to self-stabilize from arbitrary faulty states. Furthermore, by closely integrating the algorithm with the underlying clustering protocol *FLOC*, we are also able to achieve locality in self-stabilization; that is, any topology change results in repairs only within a radius of 2 units around that change. A regular unit-disk clustering technique will not achieve this property; instead the allowed dilation factor ($m = 2$) in the size of each cluster enables this property [1].

The idea of exploiting the atomic broadcast characteristic of wireless networks to ensure that connections between clusters are separated by a minimum distance is similar to the idea used in *FLOC* to guarantee minimum cluster separation. However, it is to be noted that, in the process of clusterhead election, the conflict detection can be resolved entirely locally (i.e., within one hop of a candidate clusterhead). On the other hand, in the formation of cluster connectors, two nodes have to mutually agree that no conflicts exist within their respective locality and concurrently agree on the formation of the cluster connector. This imposes an additional challenge which is addressed in the *FLOC-SPANNER* protocol.

6. Conclusions

We presented *FLOC-SPANNER*, a distributed algorithm for creation of geometric spanners in a wireless sensor network. Our algorithm uses an underlying clustering algorithm as a foundation for creating spanners and only relies on the periodic heartbeat messages associated with cluster maintenance for the creation of the spanners. There is no extra overhead for spanner creation. Given any connected network, we showed analytically that the algorithm terminates in $O(1)$ time. We also showed that the path stretch factor is bounded by $O(5/2 + 4/d)$. Furthermore, *FLOC-SPANNER* also self-stabilizes in the presence of topology changes and message losses.

We verified the performance of our algorithm using large scale simulations in JProwler, a java based discrete event simulator. Our simulations show that the average path stretch factor for routing along the spanner for large networks is less than 2. During creation of connections between clusters, we maintain the property that if two nodes a and b are used to connect clusters c_a and c_b, then no other connections exist between these clusters within the i-band range of a and b. This property ensures that the connectors are sparse and the total number of edges in the resulting spanner is observed to be $O(N)$, where N is the number of vertices, as opposed to $O(Nd)$ on the original graph, where d is the average node degree. Our simulations also verified the fact that the path stretch factor and convergence time remain constant irrespective of the network size.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] M. Demirbas, A. Arora, V. Mittal, and V. Kulathumani, "A fault-local self-stabilizing clustering service for wireless ad hoc networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 9, pp. 912–922, 2006.

[2] H. Cao, E. Ertin, V. Kulathumani, M. Sridharan, and A. Arora, "Differential games in large-scale sensor-actuator networks," in *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN '06)*, pp. 77–84, April 2006.

[3] V. Kulathumani, A. Arora, and S. Ramagiri, "Pursuit control over wireless sensor networks using distance sensitivity properties," *IEEE Transactions on Automatic Control*, vol. 56, no. 10, pp. 2473–2478, 2011.

[4] V. Kulathumani and A. Arora, "Distance sensitive snapshots in wireless sensor networks," in *Principles of Distributed Systems (OPODIS)*, vol. 4878, pp. 143–158, 2007.

[5] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic, "Impact of radio irregularity on wireless sensor networks," *Proceedings of the 2nd International Conference on Mobile Systems, Applications and Services (MobiSys '04)*, pp. 125–138, 2004.

[6] G. Narasimhan and M. Smid, *Geometric Spanner Networks*, Cambridge University Press, New York, NY, USA, 2007.

[7] R. Rajaraman, "Topology control and routing in ad hoc networks: a survey," *ACM SIGACT News*, vol. 33, no. 2, pp. 60–73, 2002.

[8] M. Smid, "Closest-point problems in computational geometry," in *Handbook on Computational Geometry*, J. Sack, Ed., 1997.

[9] G. T. Toussaint, "The relative neighbourhood graph of a finite planar set," *Pattern Recognition*, vol. 12, no. 4, pp. 261–268, 1980.

[10] P. Bose, L. Devroye, W. S. Evans, and D. G. Kirkpatrick, "On the spanning ratio of gabriel graphs and beta-skeletons," in *Proceedings of the 5th Latin American Symposium on Theoretical Informatics (LATIN '02)*, pp. 479–493, 2002.

[11] A. C. Yao, "On constructing minimum spanning trees in k-dimensional spaces and related problems," *SIAM Journal on Computing*, vol. 11, no. 4, pp. 261–268, 1982.

[12] M. Damian, "A simple yao-yao-based spanner of bounded degree," *Computing Research Repository*, http://arxiv.org/abs/0802.4325.

[13] X.-Y. Li, G. Calinescu, and P.-J. Wan, "Distributed construction of a planar spanner and routing for ad hoc wireless networks," in *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '02)*, pp. 1268–1277, June 2002.

[14] R. Wattenhofer, L. Li, P. Bahl, and Y.-M. Wang, "Distributed topology control for power efficient operation in multihop wireless ad hoc networks," in *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, pp. 1388–1397, Anchorage, Alaska, USA, April 2001.

[15] L. Li, J. Y. Halpern, P. Bahl, Y.-M. Wang, and R. Wattenhofer, "A cone-based distributed topology-control algorithm for wireless multi-hop networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp. 147–159, 2005.

[16] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu, "Geometric spanner for routing in mobile networks," in *Proceedings*

of the 2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '01), pp. 45–55, October 2001.

[17] M. Damian and S. V. Pemmaraju, "Localized spanners for Ad Hoc wireless wetworks," *Ad Hoc & Sensor Wireless Networks*, vol. 9, no. 3-4, pp. 305–328, 2010.

[18] Y. U. Wang and X.-Y. Li, "Localized construction of bounded degree and planar spanner for wireless ad hoc networks," *Mobile Networks and Applications*, vol. 11, no. 2, pp. 161–175, 2006.

[19] K. Alzoubi, X.-Y. Li, Y. Wang, P.-J. Wan, and O. Frieder, "Geometric spanners for wireless ad hoc networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 4, pp. 408–421, 2003.