FLOC-SPANNER: An $O(1)$ time, locally self-stabilizing algorithm for geometric spanner construction in a wireless sensor network

Goutham Ranganath
West Virginia University

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Recommended Citation
https://researchrepository.wvu.edu/etd/653

This Thesis is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Thesis has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.
FLOC-SPANNER: An $O(1)$ time, locally self-stabilizing algorithm for geometric spanner construction in a wireless sensor network

by

Goutham Ranganath

Thesis submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the Requirements
for the degree of

Master of Science
in
Electrical Engineering

Yaser P. Fallah, Ph.D.
James D. Mooney, Ph.D.
Vinod K. Kulathumani, Ph.D., Chair

Lane Department of Computer Science and Electrical Engineering

Morgantown West Virginia
2013

Keywords: Wireless Sensor Networks, Clustering, Geometric Spanners, Graph Algorithms

Copyright 2013 Goutham Ranganath
Abstract

FLOC-SPANNER: An $O(1)$ time, locally self-stabilizing algorithm for geometric spanner construction in a wireless sensor network

by

Goutham Ranganath

Master of Science in Electrical Engineering

West Virginia University

Vinod K. Kulathumani, Ph.D., Chair

From a communications perspective, a wireless sensor actuator and networks 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 a 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 thesis work a distributed algorithm for creation of geometric spanners in a wireless sensor network is presented. 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.
Geometric Spanners is a well-researched topic. The algorithm presented in this thesis differs from other spanner algorithms in the following ways

1. It is a distributed locally self-stabilizing algorithm.

2. It does not require location information for its operation.

3. Creates spanner network in constant time irrespective of network size and network density.
ACKNOWLEDGEMENTS

Firstly I would like to thank my committee chair and advisor, Dr. Vinod Kulathumani, for giving me an opportunity to work with him and his students. This thesis work has been made possible with his constant guidance and support. I also want to thank Dr. James Mooney and Dr. Yaser Fallah for being part of my committee. I have been fortunate to have had the opportunity to take courses with all of my committee members, and their teachings have been essential to my understanding of the subject. Next, I would also like to thank my colleagues with whom I’ve had the pleasure of working alongside.

Finally, a special thanks to my parents, family members and friends who always stood by my side at all times. I am always indebted to them.
# Contents

Acknowledgements ......................................................................................................................................................... iii

List of Figures ........................................................................................................................................................................ vi

Chapter 1 : Introduction ......................................................................................................................................................... 1

Chapter 2 : Background and Related Work ......................................................................................................................... 5

2.1 Background ........................................................................................................................................................................ 5

2.1.1 Spanner Properties ...................................................................................................................................................... 6

2.1.2 Applications of Spanners ............................................................................................................................................ 6

2.1.3 Spanners for Wireless Ad Hoc Networks ................................................................................................................... 7

2.1.4 Spanner Constructions ................................................................................................................................................ 8

2.2 Related work ........................................................................................................................................................................ 8

2.2.1 Gabriel Graph ........................................................................................................................................................... 9

2.2.2 Relative Neighborhood Graph .................................................................................................................................. 9

2.2.3 Yao-Graphs ............................................................................................................................................................... 10

2.2.4 Delaunay Triangulation ............................................................................................................................................ 11

2.2.5 Geometric Spanners for Wireless Ad Hoc Networks .................................................................................................. 12

Chapter 3 : Model .................................................................................................................................................................... 14

3.1 Network Model ............................................................................................................................................................. 14

3.2 Execution Semantics ..................................................................................................................................................... 15

3.3 Clustering ........................................................................................................................................................................ 15

3.4 Problem Statement ....................................................................................................................................................... 15

3.5 Fault Model ................................................................................................................................................................. 16

3.5 FLOC review and addition of spanner state .................................................................................................................. 17

Chapter 4 : FLOC_SPANNER Algorithm .............................................................................................................................. 21

4.1 Basic Idea ..................................................................................................................................................................... 21

4.2 Variables ..................................................................................................................................................................... 24

4.3 Program ..................................................................................................................................................................... 25

4.3.1 Monitor state ........................................................................................................................................................... 26

4.3.2 Conncand State ..................................................................................................................................................... 28
4.3.3 Initiator State .............................................................................................................. 29
4.3.3 Receptor State ........................................................................................................... 31
Chapter 5: Analysis of algorithm ....................................................................................... 37
  5.1 Fault-tolerance actions ................................................................................................. 41
Chapter 6: Performance Evaluation .................................................................................... 43
  6.1 Convergence time ......................................................................................................... 43
  6.2 Path stretch factor ......................................................................................................... 46
  6.3 Spanner edges ................................................................................................................ 48
Chapter 7: Conclusions ........................................................................................................ 51
REFERENCES ..................................................................................................................... 53
LIST OF FIGURES

Figure 1.1: 10-by-10 Grid with FLOC-SPANNER .................................................................4
Figure 2.1: Graph $G(V, E)$ with $V$ vertices and $E$ edges ..................................................5
Figure 2.2: Spanner Graph ........................................................................................................5
Figure 2.3: Gabriel Graph ...........................................................................................................9
Figure 2.4: Relative Neighborhood Graph .............................................................................10
Figure 2.5: Yao Graph .............................................................................................................11
Figure 2.6: Delaunay triangulation ..........................................................................................12
Figure 3.1: Node $j$ is subsumed by one of its neighboring clusters ......................................18
Figure 3.2: $j$'s neighbors are $l_1$ and $k_1$ .........................................................................18
Figure 3.3: $j$ becomes the clusterhead .................................................................................18
Figure 3.4: Program actions for $j$ .......................................................................................20
Figure 4.1: Spanner Construction using FLOC_SPANNER Algorithm ....................................22
Figure 4.2: State transition diagram .....................................................................................25
Figure 4.3: List of states .........................................................................................................26
Figure 4.4: List of transition events .......................................................................................26
Figure 4.5: Algorithm – When node $j$ is in MONITOR State .................................................34
Figure 4.6: Algorithm – When node $j$ is in CONNCAND State ............................................35
Figure 4.7: Algorithm – When node $j$ is in INITIATOR State .................................................35
Figure 4.8: Algorithm – When node $j$ is in RECEPTOR State .............................................36
Figure 5.1: Algorithm – Fault Tolerance Actions ..................................................................42
Figure 6.1: Convergence ratio as a function of time for different network sizes ..................45
Figure 6.2: Average convergence time as a function of network ..........................................46
Figure 6.3: Path stretch factor as a function of time for different network sizes ..................47
Figure 6.4: Path stretch factor as a function of network size, averaged over different densities 48
Figure 6.5: Number of spanner edges as a function of time for different network sizes .......49
Figure 6.6: Number of spanner edges as a function of network size ...................................50
CHAPTER 1: INTRODUCTION

Wireless sensor actuator and networks 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 a 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 thesis work, FLOC_SPANNER, a distributed, locally self-stabilizing algorithm for creation of geometric spanners has been presented. FLOC_SPANNER 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 unit distance of a cluster-head 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. 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 establishing communication between clusters. As a result clusters 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 graph remains connected. The algorithm \textit{FLOC\_SPANNER} presented in this thesis addresses this challenge and creates a geometric spanner by establishing sparse, yet sufficient connections between clusters created by FLOC.

The key characteristics of our proposed algorithm are summarized below.

1. 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 \(N\) the number of vertices as opposed to is \(O(Nd)\), where \(d\) the average degree of nodes in the network is.

2. 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 \textit{FLOC\_SPANNER}. 
3. 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).

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

5. The upper bound on the path stretch factor in the resulting spanner is shown to be $O\left(\frac{5}{2} \cdot \frac{4}{d}\right)$, 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.

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

7. 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]. Fig.1.1 illustrates the spanner graph created using FLOC_SPANNER algorithm considering square grid arrangement of nodes.
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.
CHAPTER 2 : BACKGROUND AND RELATED WORK

2.1 Background

A geometric spanner or k-spanner of a graph G is a spanning graph in which the distance between every pair of vertices is at most k times their distance in G. k is referred to as the stretch factor of the spanner. The Fig.2.1 and Fig.2.2 show the actual graph G and the spanner graph of G.

Figure 2.1: Graph $G(V, E)$ with $V$ vertices and $E$ edges

Figure 2.2: Spanner Graph
2.1.1 Spanner Properties

To construct t-spanner with some good properties where the measures of quality to be considered are:

**Size**: The number of edges in the graph. Size is one of the most important property and all the algorithms implemented produce spanners with \(O(n)\) edges. This feature has made the construction of spanners one of the fundamental tools in the development of the fast and better algorithms for geometric problems.

**Degree**: The maximum number of edges incident to a vertex. This property is useful in the development of algorithms for the construction of ad hoc networks where small degree is essential in trying to develop fast-localized algorithms.

**Weight**: The weight of a Euclidean network \(G\) is the sum of the edge weights. The best that can be achieved is a constant times the weight of the minimum-spanning tree.

**Spanner Diameter**: Smallest integer \(d\) such that for any pair of vertices \(u\) and \(v\) in \(V\), there is a path of length at most \(k \cdot |uv|\) between \(u\) and \(v\) containing at most \(d\) edges. For wireless ad hoc networks it is often desirable to have small spanner diameter. If a graph has spanner diameter \(d\), then it is said to be a \(d\)-hop network.

2.1.2 Applications of Spanners

Some of the main applications where graph spanners arise are

- Network design - Communication, Transportation
- Distributed Algorithms - Synchronizers
- Graphics - Models
2.1.3 Spanners for Wireless Ad Hoc Networks

Wireless ad hoc networks, nodes are battery operated and have limited memory resources. These characteristics make it critical to compute and maintain, at each node, only subset of neighbors that a node communicates with. This subset of neighbors defines a topology and the problem of choosing an appropriate subset of neighbors is called topology control.

Topology control problem is one of the problems in computational geometry and graph theory. Wireless ad hoc networks are modeled as graphs in which nodes represent wireless devices and connection between the nodes represent edges. The graph representing the wireless networks is usually called as network graph. If the network is homogeneous the transmission of nodes can be normalized to 1 and view the network graph as Unit Disk Graph. (The Unit Disk Graph is a Euclidean graph $G = (V, E)$ in which any two nodes are adjacent if and only if their Euclidean distance is almost 1. That is, for any two nodes $u, v \in V$, it holds that $uv \in E$ implies $|u, v| \leq 1$.)

Communication over network graph edges is inefficient due to the existence of too many edges that induce large amount of interference. The most common solution to this is to extract a subgraph $H = (V, E')$ of the network graph that is connected but is less prone to interference. However, as edges are eliminated, paths between the nodes get larger. This renders necessary an additional restriction on $H$ to contain short spanning paths. This property is known as spanning property.
2.1.4 Spanner Constructions

Any graph \( G \) has a 1-spanner, namely itself. The main criteria that are looked into while constructing a spanner are small size and good stretch. Clearly there is a trade-off between the two; for instance, the complete unweighted graph \( Kn \) contains only one 1-spanner, namely itself and this spanner thus has a quadratic number of edges. By having slightly larger stretch factor, the spanner size can be greatly reduced.

The main goals of constructing good spanners are:

- Find a skeleton of the graph
- Decrease the size of the graph while preserving distances
- Optimize stretch-size tradeoffs

Computing \( t \)-spanner of smallest size is combinatorial problem. However, computing \( t \)-spanner of smallest size is NP-hard. In fact, for \( t > 2 \), it is NP-hard.

2.2 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.
2.2.1 Gabriel Graph

In Gabriel Graph connection scheme two points are connected when the circle associated with the diameter that has the two points as endpoints does not have another point within its circumference. Formally the Gabriel Graph of a graph $G(V, E)$ comprises the set of edges $uv \in E$ such that there is no vertex or point $w$ that lies in the disk of a diameter $uv$.

![Figure 2.3: Gabriel Graph](image)

2.2.2 Relative Neighborhood Graph

The relative neighborhood graph $G = (V, E)$ denoted by $RNG(G)$, comprises the set of all edges $uv \in E$ such that the lune formed by these two nodes is empty of other nodes. The lune formed by the nodes $u$ and $v$ is represented as $D(u, |uv|) \cap D(v, |uv|)$ where $D(u, |uv|)$ represents the disk of center $u$ and radius $|uv|$. In other words there is no node $u \in E$ with $|uw| < |uv|$ and $|wv| < |uv|$.
Both the Gabriel graph and the Relative Neighborhood graph do not yield geometric spanners (constant bound on path stretch factor).

### 2.2.3 Yao-Graphs

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 \( \frac{\pi}{3} \) and retain the edge to the closest node in each sector, thus allowing local selection.

For a set of points in the plane representing wireless nodes, the Yao graph \( YG_k \), for \( k \geq 2 \), is defined as follows. At each node \( u \in V \), any \( k \) equal-separated rays originated at \( u \) define \( k \) cones. In each cone, pick a shortest edge \( uv \), if there is any and add to \( YG_k \) the directed edge \( \overrightarrow{uv} \). Ties are broken arbitrarily or by smallest ID. Yao graph has been proved to be a spanner for \( k > 6 \) with a stretch factor \( \frac{1}{1-2 \sin(\pi/k)} \). The major benefit of Yao graph is that it can be
computed fast, all nodes pick their incident edges in parallel. The out-degree of the Yao graph is less than or equal to k but in-degree could be as high as $O(N)$.

![Yao Graph]

**Figure 2.5: Yao Graph**

[12], [13] and [14] propose modifications to the Yao-graph that result in bounding the maximum degree. [15] Proposes an extension to Yao graphs that result in minimizing the transmission range at each node. However, in wireless networks, Yao-graphs and its variations may not result in reduction in the number of edges as each node independently choose 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).

### 2.2.4 Delaunay Triangulation

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.
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], [17] and [18] produce spanner that contain only $O(N)$ edges. However, all of these techniques utilize location information for creation of spanners. In our algorithm, 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.

2.2.5 Geometric Spanners for Wireless Ad Hoc Networks

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 inter-cluster 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 inter-cluster connection is atomically established using some criteria such as nodes with minimum node-id or maximum battery levels. Re-establishing 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, i.e., 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].
CHAPTER 3 : MODEL

3.1 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, i.e., 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 FLOC clustering. 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 multi-path 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. Nodes are capable of measuring the signal strength of a received message. This measurement may be used as an indication of distance from the sender. E.g. assuming a signal strength loss formula \( \frac{1}{1+d^2} \), where \( d \) denotes distance from the sender, the i-band neighbors receive the message with \([0.5, 1]\) of the transmission power, and the o-band neighbors receive the message with \([0.2, 0.5]\) power. 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.
3.2 Execution Semantics

Nodes have unique ids. Variables $i$, $j$, and $k$ are used to denote the nodes and $j.var$ to denote a program variable residing at $j$. Message broadcast by $j$ is denoted as $msg_j$. A program consists of a set of variables and actions at each node. Each action has the form:

$$<guard> \rightarrow <assignmentstatement>$$

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.

3.3 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 the later section.

3.4 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 $i \in X$ and $j \in Y$. 
The objective in this thesis work is to utilize the underlying clustering and create a spanner in the network, which ensures the following properties:

- All neighboring clusters are connected and thus the graph is connected.
- 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.
- 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. It is shown in analysis section that this results in retaining only $O(N)$ edges in the spanner while also ensuring a bounded path length ratio over the complete graph.

### 3.5 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 iff, 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 numbers 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.
3.5 FLOC review and addition of spanner state

FLOC (Fault-LOcal Clustering) [1] partitions a multihop wireless network into non-overlapping 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.

Solid-disc clustering reduces the intracluster signal connection. The clusterhead is shielded with the nodes that belong to its cluster, so that the clusterhead receives only the messages of the nodes that belong to its cluster. Solid-disc clustering also ensures the guaranteed upper bound on the number of clusters.

Illustration clustering using FLOC is shown with in the following Fig’s. The nodes are arranged linearly as shown in the Fig.3.1. Dilation factor is considered to be 2. The node $j$ is within two units from the clusterheads $L$ and $K$ and it can be subsumed to either of the clusters, here $j$ is subsumed to the cluster with clusterhead $L$ thus leading to the local clustering.

Fig.3.2 gives illustration of the how FLOC clusters are constructed locally when all the clusters are of radius 2 and node $j$ is to be assigned a cluster. In this case $j$ elects itself as a clusterhead since it is not within two units of the neighboring cluster-heads $L$ and $K$. Nodes $l1$ and $k1$ then join the cluster of $j$ because they are not within one unit of their respective clusterheads but are within one unit of $j$. Thus forms a legitimate cluster as shown in the Fig.3.3.

Some of the salient properties of the FLOC that makes it suitable for clustering of large-scale wireless ad hoc networks are

1. Locality - Each node is affected by the nodes within $m$ units.
2. Scalability – Clustering is achieved in the constant time independent of the network size.

3. Self-Stabilization – Faults are locally stabilized within $m$ untis.

![Figure 3.1](image1.png)

**Figure 3.1**: Node $j$ is subsumed by one of its neighboring clusters.

![Figure 3.2](image2.png)

**Figure 3.2**: $j$’s neighbors are $l_1$ and $k_1$.

![Figure 3.3](image3.png)

**Figure 3.3**: $j$ becomes the clusterhead.

The basic FLOC actions are briefly reviewed here for completion and Fig.3.4 shows the basic program actions of $j$. Each node can be in one of the following 5 states: idle, candidate, clusterhead, i-band or o-band. While the original actions only focused on forming cluster membership, in this paper we are also interested in realizing a connected sub-graph which is a spanner for the graph. Hence, after becoming a cluster member we add a variable $j.cp$ 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 \( j.H \), then \( j.cp = j.H \). If \( j \) is connected to \( j.H \) through a node \( k \) that is an i-band neighbor of \( j.H \) and \( j \), then \( j.cp = k \). For a clusterhead \( \cdot cp = \perp \).

**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. The id of the node that sent the conflict is stored as the cluster-parent in the variable \( cp \).

**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 cluster-head message, becomes an i-band or o-band member of the cluster based on determination of i-band/o-band status of the sender. The cluster-parent \( cp \) is set to the id of the clusterhead.

**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. The cluster-parent \( cp \) is reset to the id of the clusterhead.
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.
CHAPTER 4 : FLOC_SPANNER ALGORITHM

In this chapter 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.

4.1 Basic Idea

Spanner creation using FLOC_SPANNER is illustrated in the Fig.12. The node starts competing for spanner connection establishment once it becomes part of the cluster. Cluster-heads do not participate in the spanner connection establishment. Four different node states MONITOR, CONNCAND, INITIATOR and RECEPTOR are used for this algorithm. Fig.12 illustrates the spanner creation considering two neighboring clusters in 6 different stages. The activities in these stages are as follows

Stage 1 – All nodes in both the neighboring clusters are in MONITOR STATE

Stage 2 – Node i heartbeat timer times out and it broadcasts heartbeat message.
Figure 4.1: Spanner Construction for FLOC clustered nodes using FLOC_SPANNER Algorithm
Stage 3 – The heartbeat message sent by node $i$ is processed by the neighboring cluster nodes that are in the inner-band range from node $i$. Nodes $j$ and $k$ are in the inner-band range from node $i$, these nodes change their state to CONNCAND and begin competing for the spanner connector establishment since they do not have any connection already established with the $i$'s cluster. The nodes $j$ and $k$ record the information of node $i$'s id and cluster id as they wait for connection intitiation.

Stage 4 – Node $j$ heartbeat timer times out and it sends heartbeat message with candidacy request to the node $i$. Node $j$ changes its state to INITIATOR and waits for one heartbeat interval before establishing the connector to learn about the potential conflict for the connection establishment form its inner-band range nodes.

Stage 5 – Node $i$ receives the candidacy request from node $j$ and it changes its state to RECEPTOR by ensuring that there is no connector to the $j$'s cluster. Node $i$ resets its heartbeat timer so that it gets one full heartbeat interval to learn about the conflict for connection establishment from its inner-band region nodes. The node $k$ which is CONNCAND state receives the heartbeat message from $j$ and changes its state to MONITOR and stops competing for the connection establishment.

Stage 6 - If there are no conflicts for the connection establishment, node $i$ and $j$ establish the connector shown by the dotted lines in the Fig.4.1. Nodes $i$ and $j$ change their state to MONITOR. If there is any conflict then either of the node will send the conflict message and both the nodes stop waiting to establish the connection changing its state to MONITOR.
4.2 Variables

Each node $j$ maintains the following variables

- $j. H$ - The cluster id for node $j$ which is equal to the id of its clusterhead.

- $j. \gamma$ - It specifies the current state of the node in Spanner Connection establishment. There are 4 possible states.
  - monitor - In this state, the node is not competing to establish a connection with any other node.
  - conn cand - In this state, the node is ready to compete for connection establishment and is waiting for the next heartbeat message to send the request.
  - initiator - This is the verification state for a node that has sent a request for connection establishment and is waiting to learn about conflicts, i.e., other existing connectors that are within its i-band range.
  - receptor - This is the verification state of a node which has received a request for connection establishment and is waiting to learn about conflicts, i.e., other existing connectors that are within its i-band range.

- $j. \emptyset_x$ - A boolean variable which is set to true if the node has a connection to the cluster $x$.

- $j. v_x$ - Specifies the id of the node belonging to the cluster $x$ to which connection has been established.

- $j. Z$ - A set of clusters for which node $j$ is a connector, i.e. the set of clusters $x$ for which $j. \emptyset_x$ is true.

- $j. k_c$ - The id of the cluster with which a potential connection is being initiated.

- $j. k_n$ - The id of the node with which a potential connection is being initiated.
• \( j.L \) - 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 \( j \). \( j.Ln(C) \) is used to denote the connector for cluster \( C \) in this list and

• \( j.Lt(C) \) - denotes the time of last heard heartbeat from \( jLn(C) \).

**4.3 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, T_INITIATE, and 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 as well as 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.

![State transition diagram](image_url)

**Figure 4.2: State transition diagram**
4.3.1 Monitor state

A 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 are 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 \((j.H)\), and a set of all clusters for which it is the connector (i.e., \(j.Z\)).

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

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

- **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, i.e. it checks if \(k.H\) belongs to \(j.L\). If this condition is not satisfied node \(j\) does the following

  a) 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 \(k.H\) are recorded into \(κ_n\) and \(κ_c\) respectively.

  c) Waits for the next heartbeat timer to time out to send a request message to initiate this connection with \(k\) provided \(j\) does not learn about already
existing connections or other on-going connection with the $k$’s cluster within this time frame.

- **Receive heartbeat from node $k$ belonging to different cluster and the message is of type $T_{\text{INITIATE}}$:**

  When the node $j$ receives the connection initiation message it does the following provided there is no another connector to $k$’s cluster within $j$’s i-band range, i.e. $k.H$ does not belong to $j.L$

  a) Changes its state to **receptor**.

  b) The node id $k$ and its cluster $k.H$ are recorded into $\kappa_n$ and $\kappa_c$ respectively.

  c) 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.

  d) If node $j$ learns that there are no confirmed connections to $k$’s cluster within its i-band range, $j$ will confirm its connection with $k$.

### 4.3.2 Conncand State

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. 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 $\kappa_c$ through a connector within i-band range of $j$. Even if a piggybacked connection request 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 conn cand state the node j does the following

a) Sends heartbeat message and piggybacks a connection request to the node $\kappa_n$ for establishing connection with $\kappa_c$.

b) Changes its state to initiator.

c) Within the next heartbeat interval if node $j$ learns that there are no confirmed connections to the $k$’s cluster within its i-band range, $j$ will confirm its connection with $k$.

4.3.3 Initiator State

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 $\kappa_c$ 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.
• **Receive heartbeat from node k which belongs to the same cluster as j (i.e., j.H = k.H):**

  If j.κ belongs to k.Z i.e. node k has a connector to the cluster to which node j is waiting to establish connection then node j does the following

  a) Sends heartbeat message indicating conflict to the receptor node stored in the κn.

  b) Resets its heartbeat timer.

  c) Changes its state to **monitor**.

• **Receive heartbeat from node k belonging to κc:**

  If j.H belongs to k.Z i.e. node k belonging to the cluster to which node j is waiting to establish connection has a connector to the j’s cluster then node j does the following

  a) Sends heartbeat message indicating conflict to the receptor node stored in the κn.

  b) Resets its heartbeat timer.

  c) Changes its state to **monitor**.

• **Receive heartbeat from node κn:**

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

  a) Node κn has detected a conflicting connection within its i-band range between the same pair of clusters.
a) Node $\kappa_n$ has not accepted the connection request and hence not moved into receptor state (potentially because the initiator message was lost).

b) Node $\kappa_n$ has joined some other cluster because of re-clustering by FLOC protocol.

The variables $\kappa_n$ and $\kappa_c$ are reset. The subsequent heartbeat message from $j$ will thus not indicate connection with $\kappa_n$.

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 $\kappa_c$ within this interval and the node $\kappa_n$ also has not sent any conflicting connections within its i-band range. So, the node $j$ confirms the connection to $\kappa_n$ in a heartbeat message, marks the boolean variable $\varnothing_x$ to true where $x = \kappa_c$. The variables $\kappa_n$ and $\kappa_c$ are reset.

### 4.3.3 Receptor State

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 $\kappa_c$ 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
• Receive heartbeat from node k which belongs to the same cluster as j (i.e., \( j.H = k.H \)):

If \( j.\kappa_c \) belongs to \( k.Z \) i.e. node k has a connector to the cluster to which node j is waiting to establish connection then node j does the following

   d) Sends heartbeat message indicating conflict to the receptor node stored in the \( \kappa_n \).

   e) Resets its heartbeat timer.

   f) Changes its state to monitor.

• Receive heartbeat from node k belonging to \( \kappa_c \):

If \( j.H \) belongs to \( k.Z \) i.e. node k belonging to the cluster to which node j is waiting to establish connection has a connector to the j’s cluster then node j does the following

   d) Sends heartbeat message indicating conflict to the receptor node stored in the \( \kappa_n \).

   e) Resets its heartbeat timer.

   f) Changes its state to monitor.

• Receive heartbeat from node \( \kappa_n \):

This implies that the node with which the connection is being established has sent a premature heartbeat (without waiting for one heartbeat interval). This implies the existence of a conflict for the following reasons and hence node j returns to the monitor state.
b) Node $\kappa_n$ has detected a conflicting connection within its i-band range between the same pair of clusters.

c) Node $\kappa_n$ has joined some other cluster because of re-clustering by FLOC protocol.

The variables $\kappa_n$ and $\kappa_c$ are reset. The subsequent heartbeat message from $j$ will thus not indicate connection with $\kappa_n$.

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 $\kappa_c$ within this interval and the node $\kappa_n$ also has not sent any conflicting connections within its i-band range. So, the node $j$ confirms the connection to $\kappa_n$ in a heartbeat message, marks the boolean variable $\emptyset_x$ to true where $x = \kappa_c$. The variables $\kappa_n$ and $\kappa_c$ are reset. Note that at the same time, the node $\kappa_n$ would also mark node $j$ as a connector to cluster $j.H$, because otherwise a heartbeat message indicating a conflict would have been received.
Algorithm 1 FLOC-SPANNER actions at node $j$ in monitor state

Node $j$ in MONITOR State

Action M1: $\text{timeout}(HB\_timer) \rightarrow \text{broadcast}(HB\_msg_j(j, j.H, j.Z, T\_REGULAR))$

Action M2: $\text{recv}(HB\_msg_k()) \rightarrow$
if $j.H == k.H$ then
  for all $x$ in $k.Z$ do
    if $x \in j.L$ then
      update $j.L_r(x)$ to current time
    else
      Add $x$ to $j.L$
  end if
end for
end if

if $(\text{type}! = T\_INITIATOR) \land (j.H! = k.H) \land (k.H \notin j.L)$ then
  $\kappa_c = k.H; \quad \kappa_n = k; \quad \gamma := \text{conneand}$;
end if

if $(\text{type} == T\_INITIATOR) \land (j.H! = k.H) \land (k.H \notin j.L) \land (k.\kappa_n == j)$ then
  $\kappa_c = k.H; \quad \kappa_n = k; \quad \gamma := \text{receptor}$;
  $\text{reset}(HB\_timer)$;
end if

Figure 4.5: Algorithm – When node $j$ is in MONITOR State
Algorithm 2 FLOC-SPANNER actions at node $j$ in conncand state

Node $j$ in conncand State

Action C1: \texttt{timeout(HB\_timer)} →
\[ \gamma := \text{initiator}; \]
\[ \text{bcast}(HB\_msg_{j}(j, j.H, j.Z, T\_INITIATOR, j.k_n, j.k_c)); \]

Action C2: \texttt{recv(HB\_msg_{k})} →
\[ \text{if } (j.k_c == k.H \land j.H \in k.Z) \lor (j.H == k.H \land j.k_c \in k.Z) \text{ then} \]
\[ \gamma := \text{monitor}; \; \kappa_c = \bot; \; \kappa_n = \bot; \]
\[ \text{end if} \]
\[ \text{if } type == T\_INITIATOR \land j.H == k.H \land j.k_c == k.k_c \text{ then} \]
\[ \gamma := \text{monitor}; \; \kappa_c = \bot; \; \kappa_n = \bot; \]
\[ \text{end if} \]

Figure 4.6: Algorithm – When node $j$ is in conncand State

Algorithm 3 FLOC-SPANNER actions at node $j$ in initiator state

Node $j$ in initiator State

Action I1: \texttt{timeout(HB\_timer)} →
\[ \phi_{k_c} := \text{true}; \]
\[ \nu_{k_c} = \kappa_n; \]
\[ \text{Add } \kappa_c \text{ to } j.L \text{ and } j.Z; \]
\[ \text{bcast}(HB\_msg_{j}(j, j.H, j.Z, T\_REGULAR)); \]
\[ \gamma := \text{monitor}; \; \kappa_c = \bot; \; \kappa_n = \bot; \]

Action I2: \texttt{recv(HB\_msg_{k})} →
\[ \text{if } (j.k_c == k.H \land j.H \in k.Z) \lor (j.H == k.H \land j.k_c \in k.Z) \text{ then} \]
\[ \text{bcast}(HB\_msg_{j}(j, j.H, j.Z, T\_CONFLICT, j.k_n)); \]
\[ \gamma := \text{monitor}; \; \kappa_c = \bot; \; \kappa_n = \bot; \]
\[ \text{reset}(HB\_timer)); \]
\[ \text{end if} \]
\[ \text{if } k == j.k_n \text{ then} \]
\[ \gamma := \text{monitor}; \; \kappa_c = \bot; \; \kappa_n = \bot; \]
\[ \text{end if} \]

Figure 4.7: Algorithm – When node $j$ is in initiator State
Algorithm 4 FLOC-SPANNER actions at node $j$ in receptor state

Node $j$ in RECEPTOR State

Action R1: timeout($HB_{timer}$) →

$\phi_{\kappa_c} := \text{true}$;
$\nu_{\kappa_c} = \kappa_n$;
Add $\kappa_c$ to $j.L$ and $j.Z$;
bcast($HB_{msg_{j}}(j, j.H, j.Z, T\_REGULAR)$);
$\gamma := \text{monitor}$; $\kappa_c = \perp$; $\kappa_n = \perp$;

Action R2: recv($HB_{msg_k}$) →

if $(j.\kappa_c == k.H \land j.H \in k.Z) \lor (j.H == k.H \land j.\kappa_c \in k.Z)$ then
bcast($HB_{msg_{j}}(j, j.H, j.Z, T\_CONFLICT, j.\kappa_n)$);
$\gamma := \text{monitor}$; $\kappa_c = \perp$; $\kappa_n = \perp$;
reset($HB_{timer}$);
end if

if $k == j.\kappa_n \land \text{type} == T\_CONFLICT$ then
$\gamma := \text{monitor}$; $\kappa_c = \perp$; $\kappa_n = \perp$;
end if

Figure 4.8: Algorithm – When node $j$ is in RECEPTOR State
CHAPTER 5 : ANALYSIS OF ALGORITHM

The FLOC-SPANNER protocol creates a geometric spanner by connecting all pairs of neighboring clusters created by FLOC. In this chapter, 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 1.** The following invariant holds for FLOC-SPANNER:

**I1.** For any two nodes i and j belonging to the neighboring clusters x and y

\[(i.\varphi_y = true \land i.\varphi_y = j) \equiv (j.\varphi_x = true \land j.\varphi_x = i)\]

**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

\[(i.\varphi_y = true) \equiv (j.\varphi_y = false)\]

**Proof:** I1 states that for a connection to be successful both the nodes involved in the connection should connect to each other i.e. 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.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 conn cand state. However, only the node that first times out for its heartbeat interval (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.

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 the later sections.

**Theorem 2.** 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 $y$, i.e., clusters which have at least one pair of nodes $i$ and $j$, where $i \in x$ and $j \in y$ 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 connconsad state for forming connections. The node in either of these clusters whose heartbeat timer first expires after moving to connconsad 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 3.** 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 $j.cp$ for all nodes
Proof: As described in section 3.5 for each node that is a member of a cluster, \( j.\text{cp} \) denotes the cluster-parent, i.e., the id of the node that leads \( j \) towards the cluster-head. If \( j \) is a direct neighbor of its clusterhead \( j.H \), then \( j.\text{cp} = j.H \). If \( j \) is connected to \( j.H \) through a node \( k \) that is an i-band neighbor of \( j.H \) and \( j \), then \( j.\text{cp} = k \). For a clusterhead \( j.\text{cp} = \perp \). Since all nodes belong to some cluster, the set \( E1 \), which comprises the links between \( j \) and \( j.\text{cp} \), 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 4.** The spanner graph \( S = (V, E) \) created by the FLOC-SPANNER protocol is a geometric spanner with path stretch factor bounded by \( \frac{5}{2} + \frac{4}{d} \).

Proof: Let \( d_{ij} \) denote the hop distance between any two nodes \( i \) and \( j \) in the original network graph. Let \( d_{ij} > 1 \), i.e., \( i \) 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 solid-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 \( \frac{d}{2} \). 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 $\frac{5d}{2} + 4$. The path stretch factor is thus bounded by $\frac{5}{2} + \frac{4}{d}$.

Thus, we observe 22d that the path stretch factor has maximum of 4.5 for $d = 2$ and improves
for nodes that are at a greater separation.

5.1 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.

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 $j.L$), 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 $j.L$. If they are greater than $CT$,
then the particular connection is removed from the list.

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 same cluster as \( j \) (i.e., \( j.H = k.H \)), 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 rest 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 \( j \).

---

**Algorithm 5** Fault-tolerance actions for FLOC-SPANNER

**Action S1:** \( \text{timeout}(CT\_timer) \rightarrow \)

\[
\text{for all } x \text{ in } j.L \text{ do}
\]
\[
\text{if } now - j.L_r(x) > p \text{ then}
\]
\[
\phi_x := \text{false}
\]
\[
\text{end if}
\]
\[
\text{end for}
\]

**Action S2:** \( \text{rcv}(HB\_msg_k) \rightarrow \)

\[
\text{if } j.H = k.H \text{ then}
\]
\[
\text{for all } x \text{ s.t } (j.\phi_x = \text{true} \land k.\phi_x = \text{true}) \text{ do}
\]
\[
\phi_x := \text{false}
\]
\[
\text{bcast}(HB\_msg_j(j,j.H,j.Z,T\_VIOLATION,x));
\]
\[
\text{reset}(HB\_timer));
\]
\[
\text{end for}
\]
\[
\text{end if}
\]

**Action S3:** \( \text{rcv}(HB\_msg_k) \rightarrow \)

\[
\text{if } j.H = k.H \land \text{type} = \text{T\_VIOLATION} \text{ then}
\]
\[
\phi_x := \text{false}
\]
\[
\text{end if}
\]

---

**Figure 5.1:** Algorithm – Fault Tolerance Actions
CHAPTER 6: PERFORMANCE EVALUATION

The performance of FLOC-SPANNER is evaluated 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 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.

6.1 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 \( N \times (N - 1) \). 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., \( N \times (N - 1) \)). 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 \( N \times (N - 1) \) 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 Fig. 6.1, 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 to 5 seconds and the expected time for connections to be established is bounded by 2 times the heartbeat interval.

Fig. 6.1 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 Fig. 6.2, 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.
Figure 6.1: Convergence ratio as a function of time for different network sizes
Figure 6.2: Average convergence time as a function of network size for different network densities

6.2 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 Fig.6.3, 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 to about 5 seconds after initialization, clusters are still being formed in the underlying network and formation of cluster connections have not started. Hence the path stretch factor is zero. After this time, the number of pair-wise node paths in the system
being 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 Fig.6.1), 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 at all three density models. In Fig.6.3, 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 Fig.6.4, we observe that larger networks experience slightly lower average path stretch factors, validating the scalability of our approach.

Figure 6.3: Path stretch factor as a function of time for different network sizes
6.3 Spanner edges

In Fig.6.5, 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 Fig.6.6, 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 $O(N \times d)$ in the original graph, where $d$ is the degree of each node. Next, we observe that the number of spanner edges remains steady irrespective of network size.

Figure 6.4: Path stretch factor as a function of network size, averaged over different densities
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.

![Figure 6.5: Number of spanner edges as a function of time for different network sizes](image)
Figure 6.6: Number of spanner edges as a function of network size for different network densities
CHAPTER 7: CONCLUSIONS

In this thesis FLOC-SPANNER, a distributed algorithm for creation of geometric spanners in a wireless sensor network was presented. FLOC_SPANNER 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, the algorithm terminates in $O(1)$ time and it was proved analytically. The path stretch factor of FLOC_SPANNER algorithm is bounded by $O\left(\frac{5}{2} + \frac{d}{2}\right)$. Furthermore, FLOC-SPANNER also self-stabilizes in the presence of 2d topology changes and message losses.

The performance of the algorithm was verified using large scale simulations in JProwler, a java based discrete event simulator. 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. Simulations also verified the fact that the path stretch factor and convergence time remain constant irrespective of the network size.

The main properties that make FLOC-SPANNER algorithm efficient and can be used in large scale are

- Less memory requirement or merely no extra memory at each node to maintain the
connector information.

- No additional overhead of exchanging messages for spanner connection, use the heartbeat message effectively to establish spanner connections.

- Scalability, the algorithm is not dependent on the size of the network. Each connector formation takes same amount of time.

- Locality, the performance of the algorithm is affected by only the nodes which are with inner-band region to each other.

- Self-Stabilization can be handled by analyzing heartbeat messages and no additional overhead is required.

The main use of creating spanner graph on a clustered network with connectors between the clusters is that it greatly reduces the messages broadcasted in sending the information from one node to another. Without spanner graph every node which receives the message broadcasts the message and this continues by the other received nodes until it is received by the destined node, thus spanner network greatly decreases the communication overhead and collisions in the network.
REFERENCES


