Mesh-based Dynamic Location Service in WSANs by a Team of Robots

by

Yuanye Zhou

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc. degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

A team of robots (also called actors) in Wireless Sensor and Actor Networks (WSANs) may not be able to optimally cover an area. Hence, after an event is reported by a sensor, all robots may not be quickly reachable for possible action in time. Furthermore, sensors may not be able to immediately deliver a report to a nearby robot. In this research work, we focus on a dynamic location service to deal with a sequence of multiple events in WSANs. We propose a novel dynamic mesh-based protocol (DMesh) to boost real-time performance of the location service. We investigate new strategies for efficient event coverage, so that the robot team may appropriately partition, visit sensors periodically, and minimize the reporting delay while preserving the ability to respond, possibly with increasing number of robots as an event (such as a fire) progresses and more robots arrive near the scene.

Acknowledgements

I would like to acknowledge my sincerest gratitude to Prof. Amiya Nayak and Prof. Ivan Stojmenovic in School of Electrical Engineering and Computer Science, University of Ottawa. During the period of my master's program, they provide enthusiastic and professional guidance to my thesis. Without their help, it would barely have been possible for me to complete my degree.

Dedication

This thesis is dedicated to my parents, for their tremendous love to me.

It is also dedicated to my girlfriend, for her companionship during these years.

To all my friends, I would like to thanks for your friendship and for making my life a wonderful experience.

Table of Contents

List of Tables

List of Figures

Nomenclature

Abbreviations

A	Network Size	m^2
d_c	Communication Delay	ms
D_s	Distance of Sensor Gap	m
k_t	Time Ratio of Construction	
N	Number of Robot	
n	Number of Sensor	
R_c	Communication Range	m
T_{period_update}	Mesh Update Cycle	
T_{wait}	Wait Time	
v_r	Speed of Robot	m/s
AODV	Ad-hoc On-Demand Distance Vector	
DSDV	Destination-Sequenced Distance Vector	
DSR	Dynamic Source Routing	
e.g.	For Example	
GFG	Greedy-Face-Greedy	
GPS	Global Positioning System	
KNN	K nearest neighbour	
ROI	Region of Interest	
SCs	Service Customers	
SPs	Service Providers	
UDG	Unit Disk Graph	
WSANs	Wireless Sensor and Actor Networks	
WSNs	Wireless Sensor Networks	

Chapter 1

Introduction

1.1 Introduction

1.1.1 What are WSANs?

Recent decades have witnessed a dramatic boost in the field of Wireless Sensor Networks (WSNs) due to the advancement in the area of integrated circuits which makes it possible to manufacture tiny, low-cost sensors. WSNs gather wireless sensors to perform collaborative measuring and transmitting processes in an assigned area, called the Region of Interest (ROI). Typically, thousands of sensors, called nodes, are distributed in that area to monitor physical variables such as temperature, humidity, pressure and radiation. There are numerous applications like process management, environment monitoring, and smart grids. For example, in the general application of a smart city, WSNs are used to monitor environmental pollution, realize traffic congestion control, localize cell phones, and manage waste. Once an event is detected in a certain area, sensors will report it to sinks which are treated as gateways connected to an outer network like the Internet. In this way, data can be routed to informational aggregation and central control systems which gather and store them for responses and decisions. In [2], the authors provide a comprehensive survey on WSNs. A more recent survey on WSNs can be found in [46].

Wireless Sensor and Actor Networks (WSANs) have evolved from WSNs by integrating a group of multi-function actors (usually a few orders of magnitude less than the number of sensors) which are capable of moving upon task requests. Similar to traditional sensor networks, WSANs are not limited to simply monitoring the environment and gathering data to sinks. Furthermore, the emergence of WSANs makes the network system much more flexible and resourceful. These actors, usually static or mobile robots, have tremendously broadened the applications of WSNs because they have the capability to process data and even react to emergencies in the physical world. For example, a fire in the forest could trigger a group of mobile robots to automatically extinguish the disaster. In another situation, where a danger happens in a hazardous environment (e.g., a chemical or nuclear explosion), a few robots could form a pioneering rescue team to provide necessary life supplies and valuable information at the event spot. On the other hand, robots equipped with battery chargers could refill the power cells of sensors so that the lifetime of the whole system could be extended. In this research project, we consider all the actors in WSANs to be vehicular robots carrying specific equipment for different tasks. For convenience, we treat actor, robot and service provider as interchangeable; sensor and service consumer are the same.

In WSANs, a few principal issues have been considered by many researchers. Location service is an essential component which is concerned with how robots can efficiently respond to events detected in the assigned environment. According to its way of implementing update and search process, the location service can be categorized into different classifications. A survey of this problem can be found in [28]. In [21], Li et al. provide a mesh-based service discovery scheme which has been proven to have a very high probability of finding the closest service provider. In [22], Stojmenovic et al. consider a location service for a mobile sink with either controllable mobility or uncontrollable mobility using localized information to predict the breakage of links based on the speed and direction of the mobile sink.

Coordination of nodes in WSANs is another field that evokes great interest. The research challenges and a survey of coordination can be found in [1]. Mobile robots coordinate with each other for different tasks. The consensus issue in mobile ad-hoc networks has been studied as multi-agent systems for many years [30, 35, 19]. For node placement, several strategies and techniques have been proposed through the years and they can be found in [47]. With the relative maturity of protocols for stationary topology, much research focuses on methods to relocate sensors by moving robots [31, 10, 9]. More advanced, dynamic strategies for mobile sensors in the optimal coverage problem are also proposed [23, 13]. Even though robots in WSANs are more sophisticated than sensors, their electrical power is still bounded. Therefore, coordination for efficient energy consumption is yet another objective. In [24, 12], energy-efficient protocols in service discovery have been proposed in order to prolong the lifetimes of robots with limited-power. In WSANs, the robot coverage problem represents the quality of surveillance and response to the event. Designing coordination strategies for robots will decrease response delay and enhance the efficiency of the network.

1.1.2 Characteristics of WSANs

Before formally stating our problem, we will look at a handful of characteristics which distinguish WSANs.

- Network Constitution: WSANs consist of massive sensors and far fewer robots. Usually, sensors are not designed to be equipped with mobile devices in order to keep them less expensive and to increase their lifetimes; however, their locations and topologies could probably be changed if necessary [11, 41]. Robots in WSANs play the role of managers. If they can measure environmental parameters whether by carrying regular sensors or integrating compressed sensors, they may interact with sensors to monitor the ROI. If they do not have the capacity of sensing, they can be

viewed as acting above the sensor networks which manage the requests from sensors in their dominant areas. We ignore the sensing capacity of our robots in this thesis. All robots move only on the request in order to lower their energy cost. Sensors and robots in WSANs can be treated as being on different layers, as illustrated in Figure 1.1. In the figure, five robots charge an array of sensors to monitor a specific area collaboratively. Each robot may have its own pre-determined zone of responsibility, as shown in the figure; alternatively, it may supervise all the sensors and coordinate with other robots to dispose of possible events in ROI together.

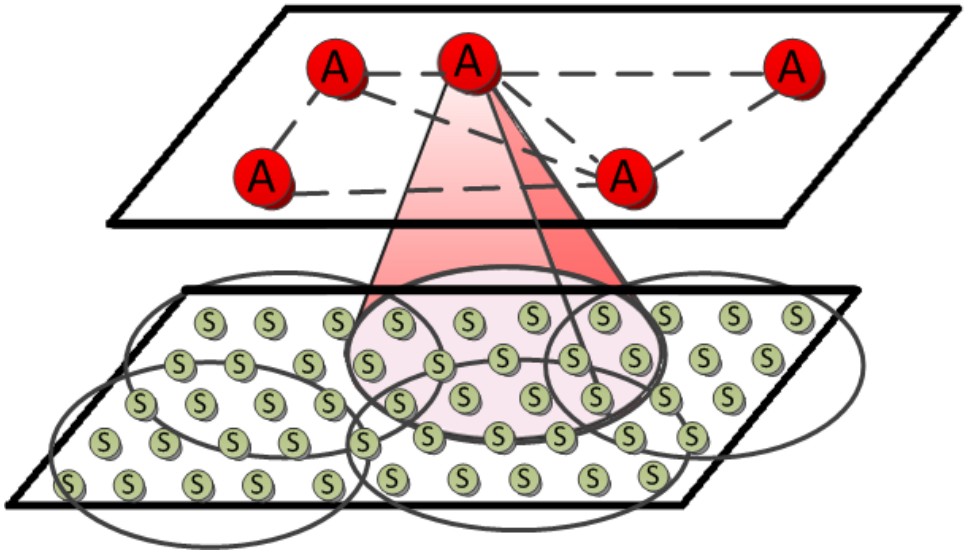


Figure 1.1: Network Layers of WSANs

- Self-management: WSANs are independent systems which means once they have been deployed to the ROI, they are away from external decision systems like control centers or technicians. Therefore, they should manage all the issues such as the network topology, energy cost, event response and coordination by themselves. Besides, we always wish to prolong the lifetime of our system once it has been deployed. Hence, both sensors and robots are expected to implement some energy-efficient schemes to finish different tasks.

- Communication of robots: usually robots in WSANs are not directly connected,

unlike mobile sensor networks. Instead, they are linked through sensor networks, as shown in Figure 1.2 (a), where the dashed line stands for an undirected link. With this connection feature, much transmitting energy of robots is saved because it requires a smaller communication range. It is valuable to WSANs because of their inherent energy-bounded attribute and it makes the network more scalable if necessary. It also balances the load of nodes and thus extends the work time of the system. However, communication coordination, either sensor-to-sensor or sensor-to-robot becomes more challenging because it involves more intermediate nodes. Three types of coordination are in the WSANs: sensor-sensor, sensor-robot and robot-robot coordination. The sensor-sensor coordination decides which sensor should report the event. Because of the overlapping of the sensing regions, a few sensors near one event may detect it together but usually, by negotiating, only one sensor will report the event. This works the same as in classic WSNs in order to eliminate redundant energy costs. The second type of coordination highlights how a sensor reports the event through the sensor networks, which is the service search process in service discovery. Robot-robot coordination decides the strategy about how to collaboratively deal with requests and give feedback.

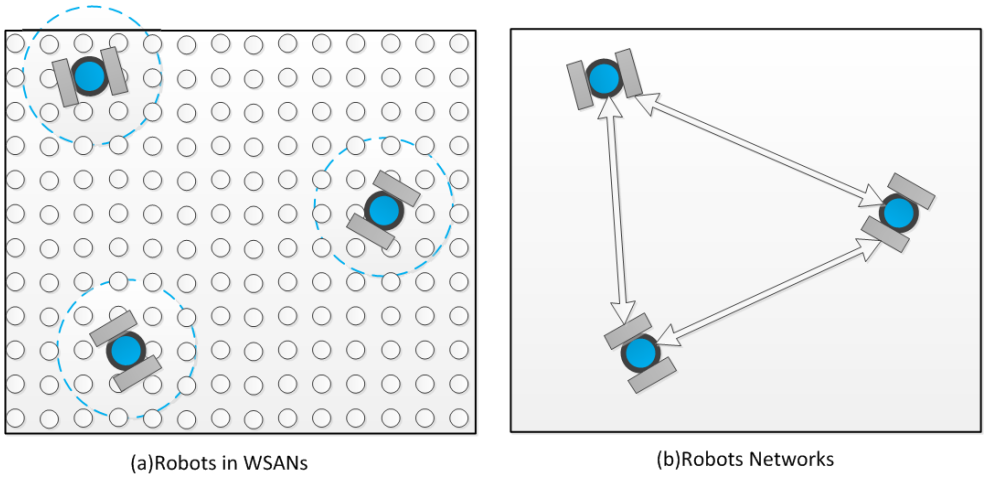


Figure 1.2: Communication Range Comparison in Different Networks

- Real-time requirement: Incorporating robots, WSANs need to automatically make

decisions and give reactions to events because the applications of WSANs are usually delay-sensitive. In the example of fire extinguishment, robots should move to the fire site with the least cost of time in order to minimize the loss. In some extreme environments, the real-time property endows WSANs with a high application potential. Even though we have the requirement of achieving good real-time performance, we need to make a trade-off between real-time performance and energy cost since it sacrifices the bounded power of robots.

1.2 Existing Solutions on Location Service

Location service is one of the top crucial and challenging components in any distributed mobile network because of network scalability and node mobility. As the prerequisite of position-based routing, it provides the source node with the location of the destination so that geographic routing can be applied. Generally, location services can be classified into two categories: flooding-based and rendezvous-based. Flooding-based location service implements location update through a specific method to a predefined area (if applicable). The message overhead in the flooding-based method is usually high and the process may even include all the nodes in the network. Furthermore, a relatively long delivery path increases the delay for the corresponding service. Alternatively, the rendezvous-based method involves two phases: location update and service search. *Service Providers* (SPs) delicately propagate their location information in the network. When *Service Customers* (SCs) require services, they query the network to search the pre-stored locations of SPs. Once these query messages "meet" the location information in SPs, SCs will get the position of SPs and ask for corresponding services directly from SPs using the geometric routing protocol. This mechanism decreases the communication overhead and eliminates the possibility of flooding in the whole network.

However, a few of these protocols guarantee an event being routed to the closest robot.

For real-time performance and energy saving, we should have distance-sensitive location service. The Voronoi division is the intuitive way to enable sensors to have access to their nearest robots. This division enables every sensor to be informed of its closest robot. If all the sensors know the location of every robot, they will be aware of the closest robot and the whole network will form several Voronoi diagrams generated by the robots (like DDMA in [25]). However, location update in these schemes leads to a high message cost for sensors because they require global computation. In [21], the authors formally define a closest service selection algorithm and propose a localized distance-sensitive location service protocol called iMesh which intentionally updates the locations or robots vertically and horizontally. However, these algorithms only guarantee one event finding its nearest robot. In the scenario where multiple events happen during a time period, the constant mesh structure is no longer a suitable way for a sensor to find the closest service provider. Once an event happens and a robot moves to the location of it, the location information of that robot in the mesh becomes inaccurate. This negative influence will spread to all the sensors that regard this robot as the closest robot. As more events continuously happen, the whole mesh will completely lose its value.

1.3 Motivation and Problem Statement

Consider the procedure of location service and how a robot responds to it in WSANs. Naturally, after we place sensors and robots in the assigned area by implementing deployment schemes [17, 36], they start to monitor designated variables in the physical environment of the ROI. Meanwhile, robots may update their location information due to the possible locomotion. When an event happens, sensors selectively report it and robots coordinate and collaboratively decide how to respond to it. Distance-sensitive location service is appealing because sensors report the event to the nearest robot. Hence, the event will be disposed of by the nearest robot to save time and energy.

However, when dealing with a sequence of multiple service requests in the network, a new problem arises. Specifically, after an event detected by sensors, a certain number of robots will move to the event and begin to deal with it. Therefore, the initial optimized topology of robots is changed and the new topology is no longer efficient to cover the whole area. As for later events reported by sensors, robots in the new deployment may need to travel a long distance to reach the new event. Even though sensors try to approach the closest robot, this undesired deployment may lead to an increasing delay for handling the event. A possible situation is shown in Figure 1.3. At some moments, after dealing with several events, robots may become clustered at the corner of the grid sensor network. Then a danger may be detected by sensors at another corner of the network. Although the closest robot will be selected, it still has to travel diagonally through the whole area to arrive at the danger spot. In some delay-sensitive applications, it can cause a catastrophic result.

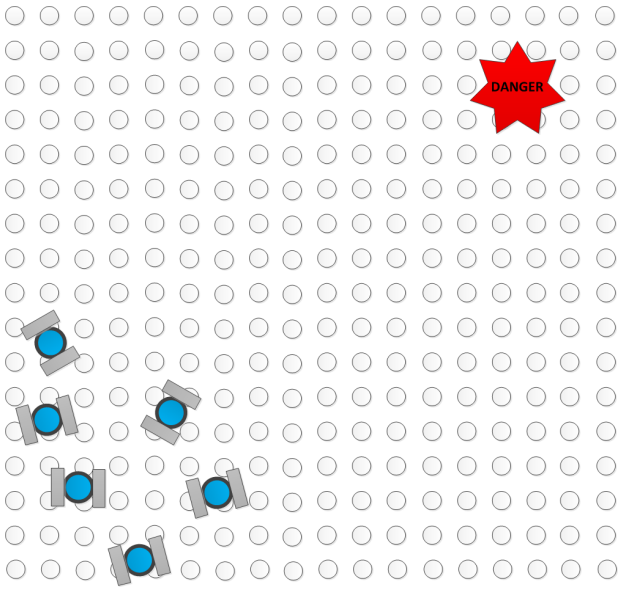


Figure 1.3: One Possible Situation of the Deployment of Robots in the Network

We take fire extinguishing as an example again where each robot works as an extinguisher. Suppose self-managed mobile extinguishers are randomly distributed in a city. Fire detectors are scattered in different zones of the city, e.g., every building and public

facility. Once a fire is discovered, actions are required as soon as possible. Otherwise, it will lead to tremendous damage to our properties. Only if the extinguishers are located evenly in the city can losses from fire be reduced to a minimum.

For small-size and less-dynamic networks, it may be realistically assumed that each sensor is aware of the existence of all the robots. However, when robots move and the network topology changes frequently, maintaining and refreshing this information requires significant communication overhead to spread the topological changes throughout the network. Thus, a network containing more than a dozen or so robots requires different assumptions and solution paradigms. Topology changes may be frequent and unpredictable. Broadly speaking, communication overhead is reduced and longevity is promoted by the use of localized protocols. Basic characteristics of sensor and robot networks, including distributed operations and a dynamic topology, make it imperative to design protocols that are localized rather than centralized. The local information must suffice for a sensor or robot to make protocol decisions; otherwise, the increased communication overhead could offset the energy savings and increase latency. Therefore, it is necessary to propose a localized efficient event coverage protocol for multiple events responses during a time period. Even though there are a few distance-sensitive protocols for location service, it is non-trivial to make such static structures dynamic and respond to a sequence of events with real-time properties. Our goal is to achieve the desired outcome with a minimal communication cost, minimal energy for moving, minimal time delay, and high benefits of the action such as guaranteeing coverage, and timely reaction to events in the area. Now, we begin to confront the following two sub-problems:

1. How does each sensor find the nearest robot to handle the reported event?

2. How do robots coordinate to cover the ROI after the optimal deployment has been damaged by events which have occurred?

These two sub-problems seem to be separate but they are connected. The former one

is the classic location service problem. Because we want to improve real-time performance, our protocol should be inherently distance-sensitive to make sure that if the robots are "nicely" distributed, sensors should always route event requests to their nearest robot. The latter one is the problem that once the deployment of robots changes due to the movement for event handling, we need a way to recover from this undesired malformation, refresh a new version of the static structure and still remain distance-sensitive for upcoming events. In order to guarantee reporting requests from sensors to their closest robot, to route data to the robot when the topology of the robot changes during movement and to avoid collision of the new and old locations of the robot, there are some questions we need to solve:

- how to propagate location information of robots with less message overhead, so that sensors always have access to their the closest robots for services
- how to nicely distribute robots, so that they can respond to random events in the whole network with minimal delay
- how to route events to robots during the interval of static structure updates when robots are moving
- how to update our structure once the locations of robots change, so that sensors are guaranteed to find their geometrically nearest robot even when the locations of robots change
- how to synchronize our robots and sensors in the network, so that they cooperate to properly handle the sequence events

These specifications provide us with deep understandings and explicit perspectives of our challenges. We will first outline some assumptions of this thesis and then claim our contributions.

1.4 Assumptions

Here are the assumptions needed to present our idea in this thesis:

- Sensors and robots do not fail during the experiments. Sensors are inferior and static in the network; robots remain static in the network and only move with constant speed based on requests.
- The position of sensors and robots in the network can be acquired by GPS devices or any other positioning services.
- Sensors are uniformly deployed in the Euclidian plane and form a quasi-grid structure. We also assume that they are connected with each other. Before the construction of the mesh, robots are randomly scattered and distributed in the network.
- Sensors and robots use omni-directional antennas so that they are connected to all their neighbours. Their communication models are Unit Disk Graph (UDG) and all the signals are without interference.
- The speed of movement of robots is much slower than the speed of transmission so that the messages can be delivered to moving targets.
- We also assume that any event in the network will be handled by single robot so that sensors only need find the closest robot to the event. In the situation where one event requires more robots, we would need to find nearby robots which is the problem of K-Nearest Neighbour (KNN). We will discuss this in our future work.

1.5 Contributions

In this thesis, we focus on how to find the closest robot in a dynamic wireless sensor network and investigate a new strategy for reacting to a sequence of events. We propose a

novel dynamic mesh-based location service protocol called DMesh. Our idea is to make a team of robots dynamically position themselves in the network to minimize response time in case of a sequence of events. Meanwhile, it preserves the ability to respond, possibly with an increasing number of robots as potential event progresses. Our protocol consists of two parts: DMesh Construction and Update (DMCU) and Distance-sensitive Routing for Multiple Mobile Robots (DRMMR). Following the analysis in the Section 1.3, the contributions of our protocol are as follows:

1. We modify the static location service protocol - iMesh, to help sensors find its closest robot. We change the registration message of the robot in a mesh construction process in order to make it suitable for a dynamic robot distribution and location update. Also, we add more procedures to collision sensors so that the robot can be aware of its nearby robots and calculate the centroid of its Voronoi Diagram.

2. After the mesh structure is finished, we dedicate a robot coordination scheme based on the Voronoi Diagram so that the nearest robot moves to the event while others move to their Voronoi centroids. Therefore, the event gets the fastest possible response while the event coverage in the network remains. During the interval of the mesh update, we propose a flooding-based location update protocol for multiple robots to guarantee that sensors can still track moving robots during the interval between the new and old version of the mesh.

We analyse our protocol performance on a special uniform distributed sensor network, quasi-grid, where the sensor is located randomly within a tiny round area, as opposed to a specific fixed point in a strict grid sensor network. We present five metrics to evaluate the simulation results. By comparison with DDMA [25] and the static mesh-based location service protocol-iMesh [21], our protocol remarkably decreases the distance that the nearest robot has to move to the incident position for continuous events. It indicates an

improvement in real-time performance of the location service, which is the principal goal of this thesis.

1.6 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, we provide a review of related works. In Chapter 3, we propose our general idea as to how the robots coordinate to solve the problem and we give a detailed description of the algorithm. In Chapter 4, we present the experiments and analysis of our ideas. Finally, in Chapter 5, we give our conclusion and discuss the future work.

Chapter 2

Related Work

2.1 Location Service

Location service, or service discovery, in WSANs is a problem dealing with the trade-off between refreshing frequencies of the locations of mobile service providers and routing for stationary service customers to the latest position. The objective is to design a scalable distributed service so that sensors can track the locations of mobile nodes. In order to minimize the cost of location updating and searching, numerous protocols have been presented. Usually, location service algorithms are required to be low-cost in bandwidth, scalable to large systems, locality-aware of service providers and robust with different topologies and networks. According to whether it involves a looking-up process or not, we can divide location service into flooding-based location service or rendezvous-based location service (Figure 2.1). The former depends on the flooding of the location of service providers to all nodes or to nodes in a certain area. This location refresh may happen during the period when the locations of service providers have changed or when neighbours of service providers have changed. In order to improve its performance, many constraints have been applied. The latter category contains two phases: location update and service search. In the update phase, a set of sensors are selected to store the latest locations of service

providers; in the search phase, the requested sensor finds the location information from sensors and routes it to service providers. In the rest of this section, we will review some of the classic algorithms.

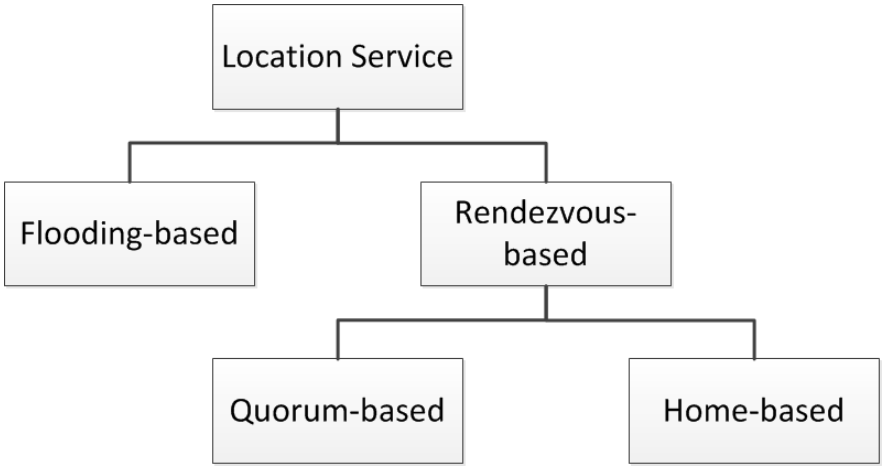


Figure 2.1: Classifications of Location Service

2.1.1 Flooding-based Algorithms

The main idea of flooding-based algorithms is to make all nodes in a specific area informed about actors and their latest locations. The highly accurate location of a robot requires a large update area (e.g., the whole network) and message overhead. Usually, location refreshes cause severe network overhead or even congestion. The flooding can be triggered by an actor; it also can be initialized by a sensor. If the process is started by actor, it is called proactive flooding such as DSDV [32]. The actor location updates periodically or when some metrics changes such as the distance or the connection. On the other hand, reactive flooding, like DSR [14] and AODV [33], means that sensors search the location of an actor when it needs to report data or other requests. The flooding mechanism usually requires high bandwidth and energy.

2.1.1.1 Doubling Circle

In [3], an actor divides the whole network into a sequence of circular zones centered round itself with radii $2^i R$ for i = 1, 2, 3, ... For each of the circle, a refreshment timer is set. When the timer expires or the actor goes across the border of circle C(t) for some t, the actor propagates its new location to a circle with radii $2^{t+1} R$. For sensors, once they need to find actors, they follow these circle zones. Every sensor routes packets to its latest updated location of an actor. As packets are forwarded closer to the actor, they will be diverted toward the center of the circle with half the radius of the previous one. Eventually, packets will be delivered into the smallest circle that the actor is in and reach the actor. For example, sensor S needs to report an event to the actor in Figure 2.2. In this example, actor moves from D" to D through D'. Sensors in different sizes of circles contain different locations of the actor. According to the latest update of the actor location in S, it will send the search message to the previous actor's location D". As the message is delivered closer to D", it will find a newer actor location when it gets into the next circle with half the radius of the previous circle. Thus, the message will be redirected to D'. These steps are repeated. Finally, the search message will arrive at the current precise actor's location D. Although the Doubling Circle can be applied to a scalable network, inherently its propagation process may involve the whole network if the actor keeps moving for a long time.

2.1.1.2 Integrated Location Service and Routing

In [22], the authors present two localized guaranteed-delivery location service algorithms for nodes to route to a low-speed mobile sink (e.g., the vehicular robot). Considering the pattern of movement, those two schemes are dedicated to unpredictable moves and controllable moves, respectively. The main idea is to update the location of the sink with a slow-varying position instead of the precise location when there is a linkage breakage or

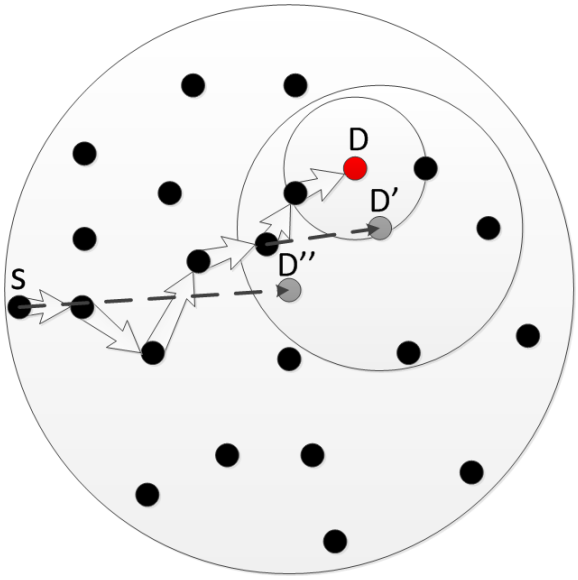


Figure 2.2: Double Circle Location Service. The Mobile Destination D Changes Its Position From D" to D' and Stays at D.

creation.

In random move, the mobile robot periodically exchanges a HELLO message with its neighbouring sensors and monitors the condition of linkages. After any current linkage break or new linkage creation, the robot updates its location information by broadcasting flooding-type location update messages. This message contains the latest location of the actor and a list of relay neighbours which is the smallest subset of the relay neighbours by the MPR [34] method. The retransmission will happen if a node is in the relay list or if the next hops of a node to the new and last-reported positions of the robot are different. If the robot loses neighbours, which means that such nodes are not the neighbours of any of the robot's current one-hop neighbours, the robot will send a routing-type to such nodes by GFG protocol.

While in controllable move, because the destination of the robot is known, it can predict link breakage of its neighbouring sensors and estimate the minimum broken time according to its direction of movement and speed. When a sensor is about to lose contact with a robot or a new sensor is detected, it updates its current location and the endpoint location

which it forwards to. Sensors receive location information and retransmit only if the new endpoint is different from the old one. In case the robot changes its destination during movement, the actor sends a recovery message to the old endpoint. Because this point does not actually exist, according to the facing property in the GFG protocol, it is natural for the closest node to the endpoint to receive this message twice. Once that happens, the traverse will be stopped and this node will perform as an anchor point. Later, packets routed to this old endpoint will be received by this node and forwarded to the new endpoint. Comparing ILSR with the Doubling Circle algorithm, both of them are localized structure-free location service protocols and the ILSR considerably reduces the message cost.

Figure 2.3 shows how these two protocols work. The blue circle is the communication range of the robot in both figures. Figure (a) is the ILSR for unpredictable movement. The robot moves from position R1 to R2 and loses contact of node L1. During this process, a new flooding type message is created. Node a, Node b, Node c and Node d will retransmit the packet because the next hop of a, c and d to the robot changes and b is the relay neighbour. Node e will not retransmit since its next hop to the robot remains the same (Node d). The robot at R2 will also start a routing-type message to update the location in the lost Node L1. The transmission path using GFG is marked by the red dashed line. (b) illustrates ILSR for controllable movement. The robot almost lost its neighbour L2 at R2 along the way to its endpoint E1. It will flood its current location as a location update message. A node will retransmit the message if its endpoint is different from the one in the message. At position R3, the robot changes its endpoint to E2 because something with higher priority happens at E2. The robot will send a location recovery message including the new endpoint E2 to its old endpoint E1. Because node A is the node that is closest to the virtual point E1, it will receive this message twice due to the nature of GFG. Node A will be treated as an Anchor Node and will redirect any request that is attempted to be routed to the old endpoint to the new endpoint.

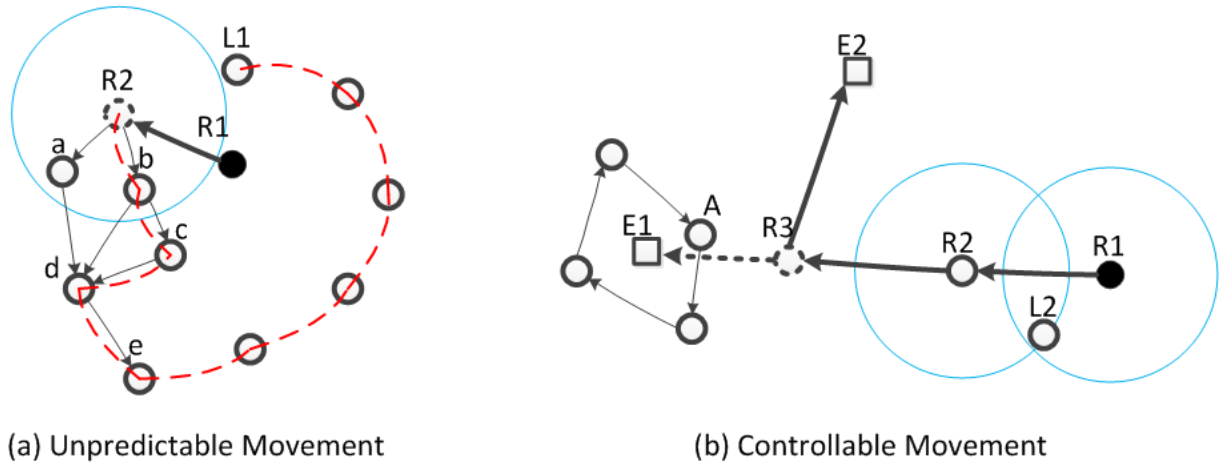


Figure 2.3: Location Update in ILSR

2.1.1.3 Distance Sensitive Flooding

In [25], Mei et al. present a distributed sensor management algorithm (DDMA) to monitor node failures by dynamically constructing the Voronoi diagram. Robots broadcast their location update message to the whole network; sensors receive such messages from all the robots, compare their location information to decide which one is the closest and retransmit it. When any event happens, the closest robot takes the event and informs all the sensors so that the sensors will re-choose the closest robot. Once the event is finished, the robot will broadcast its new location and the sensor selects the new closest robot again. This scheme enables the event to be handled by its closest robot. However, the global computation creates a high message cost and consumes the limited energy of the sensor.

2.1.2 Rendezvous-based algorithms

The rendezvous-based algorithms use optimal structure to update the actor's location information and the sensor searches this location by purposely sending a request message to make the joint point. When the request message hits the part of the structure that stores actor's location information, the reply message will include the actor's location.

2.1.2.1 Information Mesh Quorum

In [21], Xu et al. propose a novel localized planar quorum-based scheme to discover "nearby" actors. Nearby is defined as twice the distance from the event to the closest actor. This scheme significantly improves the message complexity of the well-known strip quorum method [38]. It creates an information mesh (iMesh) structure to update the location information of the service providers (SPs). Specifically, each SP delivers its location message to service customers (SCs) in four geographic directions, i.e., East, West, South and North using the GFG routing protocol. When these messages of different service providers collide with each other, a blocking rule is applied. That is to say, in the collision points, SCs receive multiple versions of the location of SPs and will only propagate the message of the closest one to the following SCs. However, a farther SP message may arrive at the collision point earlier and this wrong location will incorrectly be sent to others. In this asynchrony situation, the node in the collision point sends a revocation message following the former path of nodes which store wrong location updates and erase undesired information. In order to make iMesh effective in a few uncommon situations where the residing grid row or column of the closest robot is not a part of the perimeter of the mesh cell of the sensor, an extension rule is also proposed to help find nearby SPs. Nodes in collision points which block others' messages orthogonally, continually deliver the closest SPs location message along the backward transmission path of the location messages from nodes being blocked. Examples with one SP and seven SPs in arbitrary sensor networks are shown in Figure 2.4. To find the nearby SPs, the sensor, which detected the event, conducts a cross lookup process in four directions. When any of these search messages reaches the border of the sensor mesh cell, it replies with the closest SP's location information recorded in it.

Through analytical study, the authors show that iMesh has significantly lower message complexity than the strip quorum method [38] and that it generates a constant per node

storage load, which is a unique property that no other quorum-like algorithm possesses. Extensive simulation shows that iMesh guarantees nearby (closest) service selection with probability $> 99\%$ (resp. 95%). Moreover, iMesh eliminates global computation and has a constant node storage load. In [27], the authors point out some flawed situations of iMesh, especially in the sparse network, where it cannot find the nearest robot and add localized auction aggregation protocol to improve the efficiency of finding the closest actor. Even though the probability of finding the closest actor is guaranteed for every single event, as we mentioned in Chapter 1, iMesh has drawbacks when dealing with a sequence of multiple events.

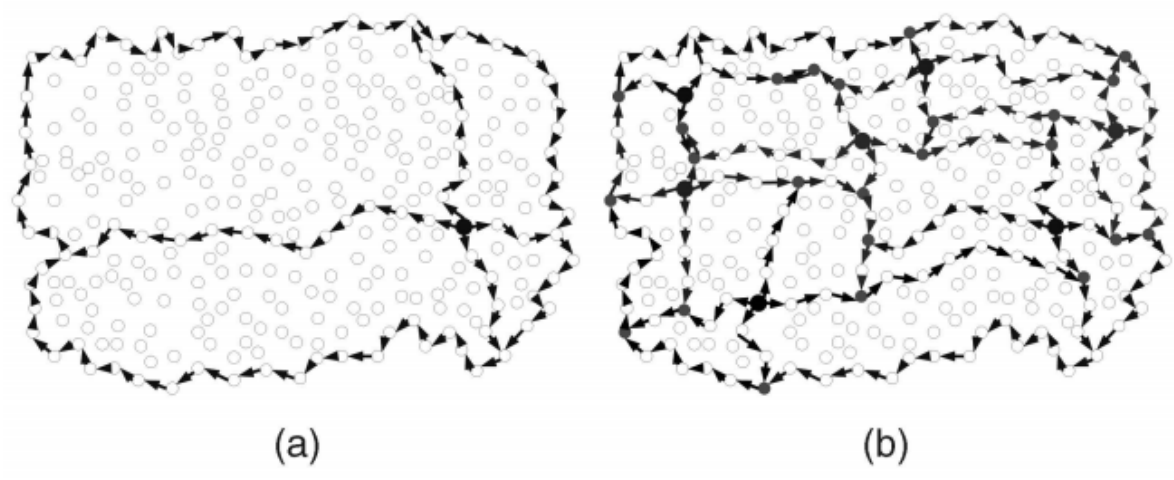


Figure 2.4: Figure from [21]. Information Mesh in Arbitrary Sensor Networks. (a) One SP (b) Seven SPs

2.1.2.2 Home-based Location Service

Home-based location service can be regarded as a special kind of quorum where the update and search regions are the same. In [37], the idea is derived from the home agent in the cellular network. When the moving sink M finds any lost node of its neighbouring list, it will broadcast its initial location within a circle with radius R where R is related to the transmitting radius. All nodes in this area will make up home agents of M. When

the counter of connecting changes between the sink and its neighbouring node exceed the threshold value, M will send a location update message to its home agents using some greedy routing protocols until there is no closer neighbour to the home agent than itself. The intermediate nodes on the path and their neighbours will also update the new position of M. For the intermediate node belonging to the home agent, the message will be resent only if it finds another node closer to the center of the home agent. As for the destination search, source node S will send one message targeted at the current location of destination D, and another one for the center of home agents of D. The former message will be redirected according to a more precise location of D through the way closer to D. The latter message will find the first node C in the home agents. C will get a newer updated location of D in the message (if applicable) and a request for the location of the destination within a circle with radius R. Using the most recent information of D, C will route the message from S to destination D. Even though this scheme reduces the communication overhead by abandoning unnecessary flooding, its message cost is still high if the moving sink is far away from its home agents. This limits the applications to a small scale with a low speed of motion.

Figure 2.5 is an example of a home-based location service. The mobile sink starts at $D1$, going through $D2$ and finally arriving at $D3$. At $D2$, the sink sends a location update message to its initial position marked by the white arrows. Because the node v is the closest node to $D1$, it will receive the location update message and broadcast this message within the circular area centered at itself. When the source S is looking for a robot, it will dedicate a message for position $D1$ according to the information stored in itself. This message will change its destination when it is received by p, because p is informed that the robot has gone to $D2$ (illustrated as the black arrows in the figure). Then node q will route the message with the latest position of robot $D3$. After the robot receives the request from S, it will notify S with its feedback.

Another kind of protocols that use the home base concept was applied in the grid hier-

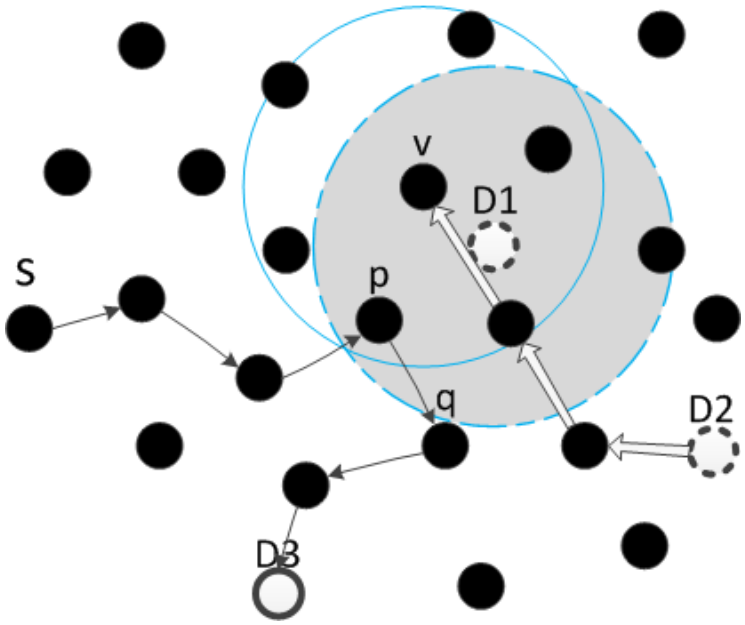


Figure 2.5: Home Based Location Service

archical network. GLS [20] is a famous, fault-tolerated and scalable region-based location service protocol. It divides the whole network into squares of different orders. Adjacent four order-k squares make up a bigger order-(k+1) square and any order-k square is only allowed to be part of one certain order-(k+1) square to avoid overlapping. This partition is known by all the nodes. A node D selects a node from each of the other three regions of each level of grid hierarchy as a location server. This selection rule is to choose the node with the least ID greater than itself and the ID space is circular. Specifically, D sends a location update message to its order-k square and the first node that receives that message in the square will forward it to the node with the least ID greater than D. This location update process will end when it reaches the square-wide least ID greater than D and set it as the location server. When a node S wants to find the location of node D, it will send a request to the node with the least ID greater than D. The node which receives such a request will forward it in the same way until it reaches the location server of D. Then this request will be directly routed to D using geographic forwarding since the location of D is known. Figure 2.6 is an example of GLS. The whole network is partitioned into 64 order-1 squares and some of the nodes are ignored. Suppose Node 17 is the source node. Black

nodes are the location server in their squares. When Node 50 makes a request to find D, it finds the closest node in term of ID to the D in each order square until the request message gets Node 37 which has the location information of D.

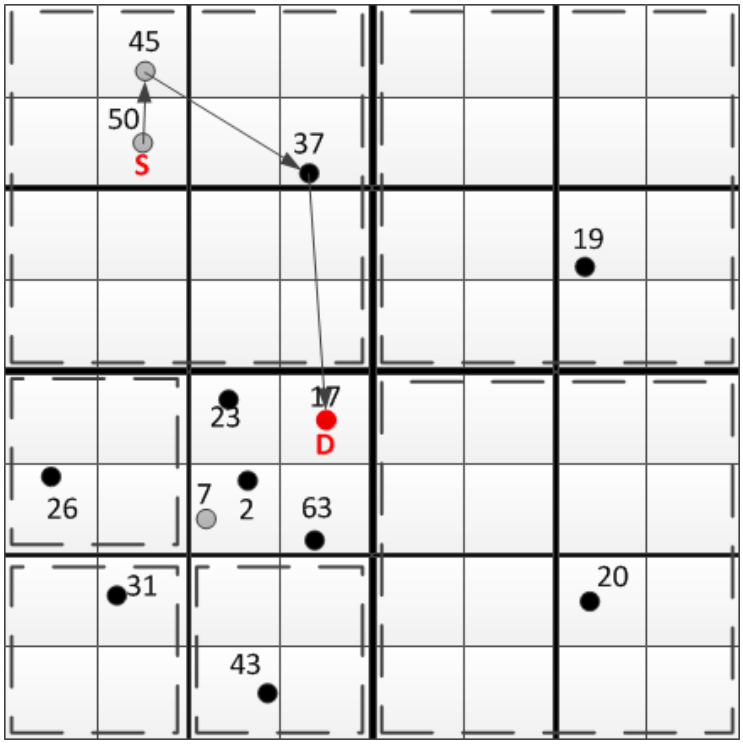


Figure 2.6: Grid Location Service (GLS)

GLS is a hierarchical scheme so that it can be applied to a scalable ad-hoc network. However, sometimes the routing path is unnecessarily long and may lead to a zigzag line which will increase the message delay. Moreover, when the nodes frequently move, the overhead of the location update message is very large and the latency for the location search is high. HLS [15] amends the partition of the network in GLS and a personalized tree by using the same hash function to improve the performance in situations with high node mobility. However, both GLS and HLS require frequent global computation and have inconstant storage load.

2.2 Robot Coordination in WSANs

Coordination in WSANs includes sensor-actor coordination and actor-actor coordination. The former class concerns how to collaboratively build a path between the sensor and the actor. Its objective is to help the sensor to deliver the data. The latter class regards how a group of actors make a decision and perform cooperatively for a certain purpose. These objectives include sensor placement, boundary coverage, robot dispersion and dynamic task location. Simply put, sensor-actor coordination deals with the way to report an event; actor-actor coordination deals with the way to handle an event.

A lot of research has been done in the area of robot coordination in recent decades. In [26], a scheme for sensor-actor and actor-actor communication is presented. The actors are stationary and sensor networks in an event area are partitioned in different groups, each reporting to a different actor. Within each group, a tree routed at the corresponding actor is built for data gathering with low energy cost and high reliability. The reliability is defined as the percentage of the packets generated by sensor networks in the event area and received from an actor not later than a predefined latency bound. The authors provide centralized as well as distributed solutions for the tree construction and, more precisely, they present a distributed protocol where the trees are constructed on-the-fly and adjusted according to the current reliability level. Also, the actor-actor coordination problem is formulated as a mixed integer non-linear program and a localized auction protocol is proposed for deciding which actor will handle the current event.

In [45], after an actor receives the most urgent event according to the predefined event priority, the actors coordinate with each other in the event area about their acting capabilities. This value is related to the distance to the event and the residual energy: the farther the distance to the event is and the less residual energy an actor has, the less acting capacity the actor possesses. The actor with the highest value will handle the event. In [29], Ngai et al. increase the reliability in a cell network where actors run a relocation algorithm

to move to the area with high event frequency in order to provide fast event response. However, actors should periodically communicate with each other by direct one-hop communication for their relocations which raise energy consumption. Additionally, different strategies may be applied for diverse applications due to different requirements. In [40], the authors focused on the surveillance of a sensor network for tracking and capturing tasks. Another application like fire fighting was discussed in [16]. However, in this thesis, we will mainly review the actor coordination based on the Voronoi Diagram [4]. As a fundamental concept in the geometry, we will present the Voronoi Diagram as the prerequisite knowledge. In the reminder of this section, we will discuss a few typical solutions in actor coordination.

2.2.1 Voronoi Diagram

Voronoi Diagram or Voronoi Tessellation (Figure 2.7) is a famous optimal partition of n-dimensional space for a set of fixed points in computational geometry. A survey of the theory and application of the Voronoi Diagram is given in [4]. The set of points $P = (p_1,...,p_n)$ is called sites or generators. Let Q denote a convex polytope in $\Re^N$. The division makes space into a collection of discrete Voronoi polygons $V = (v_1,...,v_n)$ which satisfies the condition:

$$v_i = \{q \in Q | \parallel q - p_i \parallel \leqslant \parallel q - p_j \parallel, \forall i \neq j\} \tag{2.1}$$

In the Voronoi Diagram, any two generators of adjacent polygons are called neighbours. The vertex of a Voronoi polygon is the point that is equidistant to three or more of its corresponding generators.

2.2.2 Centroidal Voronoi Diagram

A Centroidal Voronoi Diagram (Figure 2.8) is a special case of Voronoi Diagram where generators are located at the centroid of every Voronoi polygon. The centroid of a polygon

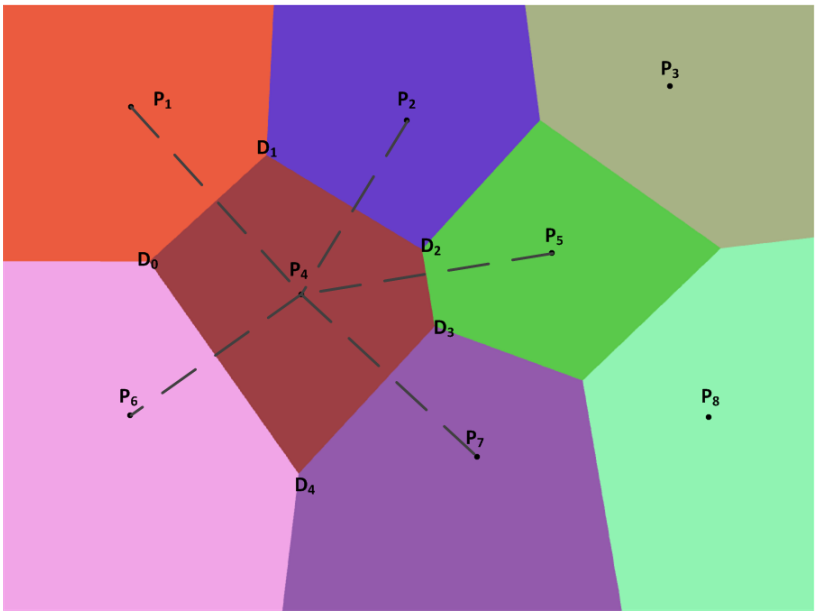


Figure 2.7: Voronoi Diagram with Randomly Distributed 8 Generators

is the average position of all the points in it; the centroid has the average x and y coordinates of all the points on the polygon. Given the density function $\rho(q)$ of point q, by definition, the centroid C_v can be expressed by:

$$C_v = \frac{1}{M_v} \int_v q\rho(q)dq \tag{2.2}$$

where M_v is the mass of the polygon defined by:

$$M_v = \int_v \rho(q)dq \tag{2.3}$$

For uniform density, where the $\rho(q)$ equals to 1, there is another way to calculate the centroid on a plane based on the coordinates of its polygon vertices. Consider a two-dimensional n polygon diagram with vertices $((x_0, y_0),...,(x_{n-1}, y_{n-1}))$ in Figure 2.9. For convenience, we assume (x_0, y_0) is the same as (x_n, y_n), then the mass and centroid can

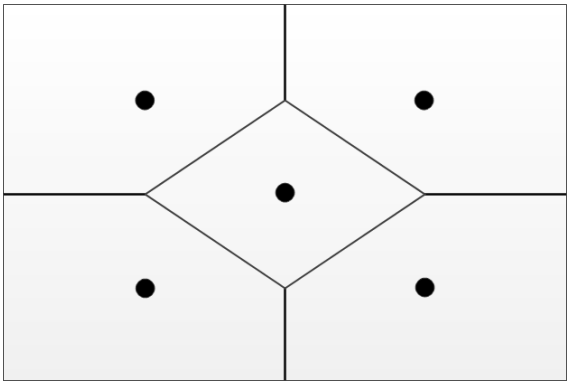


Figure 2.8: Simple Centroidal Voronoi Diagram with 5 Generators

be expressed as:

$$M_v = \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i) \tag{2.4}$$

$$C_{v,x} = \frac{1}{6M_v}\sum_{i=0}^{n-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \tag{2.5}$$

$$C_{v,y} = \frac{1}{6M_v}\sum_{i=0}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \tag{2.6}$$

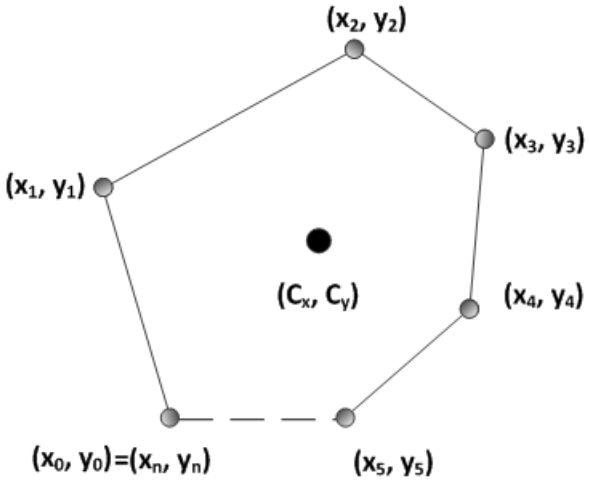


Figure 2.9: Closed-form Expression to Calculate the Voronoi Centroid

2.2.3 Voronoi Diagram in Robots Coordination

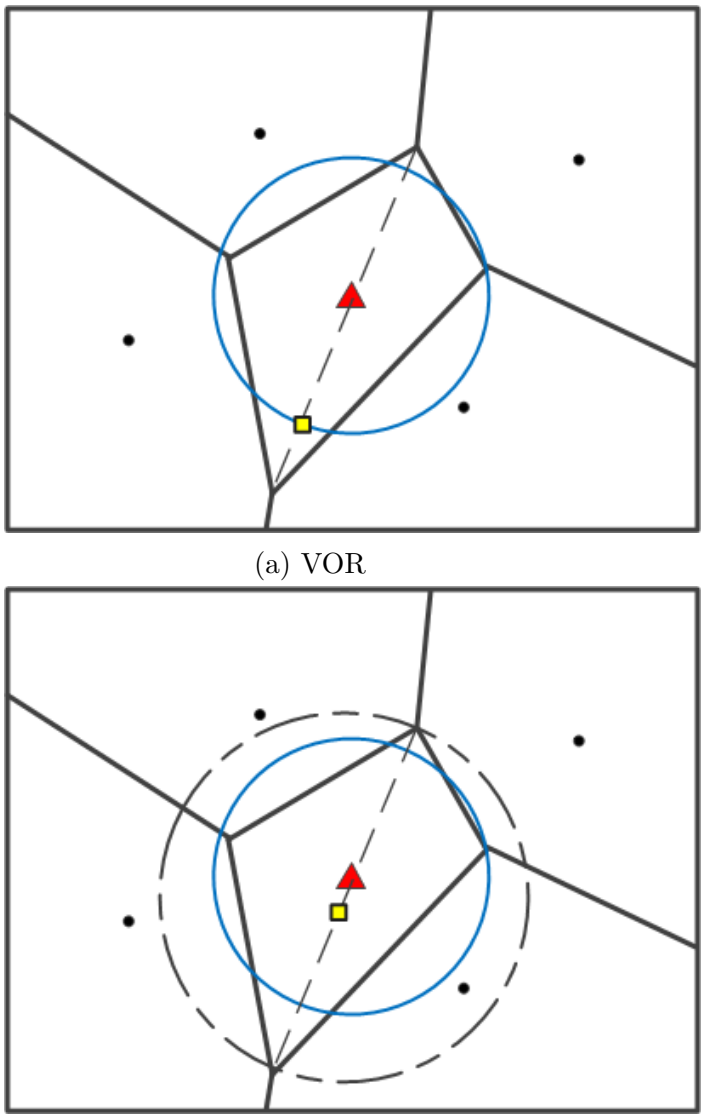
The structure of the Voronoi Diagram has been widely applied in many disciplines like geography, biology, vision science and engineering. In the distributed sensor network, researchers use this structure in node coordination to achieve optimized topology for specific topics such as sensing coverage, task assignment, path planning and energy management. Nodes, sensors, or robots in these networks act as generating points to construct their Voronoi Diagrams. According to the property of the Voronoi Diagram in equation 2.1, points in each Voronoi polygon are closer to their own generating point than other generating points. This gives a way to improve the performance of nodes when it takes distance into consideration. In the coverage problem, sensors target at collaboratively reducing or eliminating the sensor hole. With the unit disk sensing model, one sensor has to make sure that more area in its Voronoi diagram will be covered within the sensing range. In task assignment, robots are responsible for reacting to events in the network. In the path planning and energy management, nodes should minimize their moving distance because the energy cost for moving is much higher than transmission. Finding the shortest path to arrive at targets extends the lifetime of the network. To avoid global computation, some technology about proximately localized Voronoi Diagrams [39] should be used. We will review several applications of the Voronoi Diagram below.

2.2.3.1 VEC, VOR and Minimax

In [42], Wang et al. provide three strategies for sensors to change their deployment according to the assessment of sensors' vicinities location. A sensor generates its Voronoi polygon and moves to the uncovered area of the polygon. This is based on the simple fact that any point within its Voronoi polygon is closer to its sensor than other sensors; that is to say, each sensor should cover as much of the area of its Voronoi polygon as it can. Specifically, these methods are Vector-based (VEC), Voronoi-based (VOR) and Minimax.

VEC is inspired by electromagnetic fields and simply pushes sensors away from each other. Consequently, they are evenly distributed within the whole network. It uses the virtual force between sensor and sensor to adjust the distance among sensors to a desired value. Meanwhile, the boundary of the network tries to push sensors away. Compared to VEC, in VOR, once a coverage hole is detected, the sensor moves to its farthest Voronoi vertex. To prevent the required moving distance from exceeding the sensing range, the moving distance is limited, at most, to half of the difference between the communication range and the sensing range. Once the sensor leaves its current location to a new destination in one round, new sensing holes may be generated so that the sensor may move back and forth in the following several rounds. To avoid this situation, oscillation control is added. It checks the moving direction before it moves. If the direction of the destination is the same as the last round, it moves; otherwise, it stops in this round. Minimax is, in some ways, similar to VOR. But instead of choosing the farthest vertex of the Voronoi polygon, it moves to the point inside its Voronoi cell whose distance to the farthest vertex is minimized. This modification will lead the sensor to the centre of the smallest enclosing circle of the Voronoi vertices and thus keep most of the vertices of the Voronoi polygon within the sensing range.

A comparison of VOR and Minimax is provided in Figure 2.10. In this simple example, the communication range is three times larger than the sensing range. The red triangle is the targeted sensor that calculates its next round destination which is marked as a yellow square. In VOR, the sensor tries to move to the farthest vertex of its Voronoi polygon within its moving distance limit (marked as the blue circle) shown as the point circle. The dashed line is circle formed by the farthest two points of the Voronoi cell and the next destination is the center of that circle. Clearly, Minimax reduces the travel distance per round for sensors and eliminates the possibility or need for moving back and forth.



(a) VOR

(b) Minimax

Figure 2.10: Comparison of VOR and Minimax

2.2.3.2 Coverage Control using Centroidal Voronoi Diagram

In [8], Cortes et al. systematically provide and analyse the model of Centroid Voronoi Tessellation (CVT) applying it to wireless mobile sensor networks. It is based on the famous Floyd algorithm to propose distributed, gradient descent algorithms to optimally deploy sensors for both continuous and discrete time. The authors first explain the optimal coverage program as maximizing the detection probability in a desired area. It consider

the locational optimization function:

$$H(P,W) = \sum_{i=1}^{n} \int_{W_i} f(\parallel q - p_i \parallel)\phi(q)dq \tag{2.7}$$

where $P = (p_1, ..., p_n)$ is the location of nodes, $W = (W_1, ..., Wn)$ is the partition of convex polytope in $\Re^N$, $\phi(q)$ is the distribution density function, $f(\parallel q - p_i \parallel)$ is the sensing performance within the distance $\parallel q - p_i \parallel$. Equation 2.7 is related to the position of sensors and the partition of the space. For a Voronoi Diagram, it gets minimized if the sensor is located in the centroid of its Voronoi Diagram.

In [18], the authors also suggest taking advantage of a Centroidal Voronoi Diagram to boost the coverage area of a group of mobile sensors. They claimed that both VOR and Minimax only utilize part of the information of a Voronoi Diagram and thus it slightly degrades in performance in its result. Based on the observation of distance between nodes eventually becoming equal, the authors propose a Dual-Centroid Scheme when the Voronoi polygon is completely covered by Voronoi neighbour polygons. It sets the destination of sensors in each step to a linear combination of Voronoi Centroid and the center of the sensor's Voronoi neighbours. As shown in its simulation, this scheme accelerates the coverage process compared to VOR and Minimax.

Even though moving to the centers of Voronoi diagrams optimizes the deployment of mobile nodes in the network, global information and computation make these protocols unsuitable for robot coordination in the WSANs because the connection of a robot in WSANs is undirected and dependent on sensors. Therefore we prefer to use localized information and make decision in WSANs to drive our robots to their Voronoi centres to save energy.

2.2.3.3 Max-Min

In [6] Caicedo and Zefran propose a stable and scalable way called Max-Min to best monitor a convex area and localize an event which balances the localization and coverage task for mobile agents in the network. Before assigning tasks, the number of agents is given by a distributed counting algorithm. The count variable in agents are initialized to 0 except the "leader" to 1. By using some consensus algorithms, they will be averaged and go to $\frac{1}{n}$. According to this number n, the number of agents needed for tasks can be given by $P(n)$. After an event happens, agents are assigned to take either an event-response task or a cover task according to its urgency and the local urgency associated to the event. The urgency represents the level of satisfaction of a certain event; the local urgency means the importance of an agent for a particular event. If an event happens, the agent selects the larger one between its own urgency and local urgency, and compares it with that of others by sending invitations to its neighbours. The first given number of agents will respond to the event, while others with less urgency move to the centroid of Voronoi polygons to keep the network fully covered. The final configuration of the network guarantees that no more invitations can be accepted and all the agents that do not go to events are evenly distributed in the network if there are enough agents at all. This protocol deals with multiple events in the network happening at the same time, but does not aim at monitoring events for a period of time which it may suffer from collisions about the urgency among robots.

Chapter 3

Dynamic Mesh Structure For Location Service

This chapter presents our dynamic mesh algorithm to efficiently handle a sequence of multiple events in the network. First of all, the network model will be presented. Then our general idea will be described briefly. More details and essential steps of our protocol will be discussed in two different parts:

1. DMesh Construction and Update
2. Distance-sensitive Routing for Multiple Mobile Robots

For convenience, we explain our proposal in strict grid sensor networks where every sensor is located right at the corner of the square. But in the simulation, we will implement our protocol in uniformly distributed quasi-grid sensor networks.

3.1 Network Model

Before discussing our idea, we should consider our network model first. The WSANs consist of N robots which are randomly deployed in a two dimensional ROI, denoted by $A = X \times Y$, and n sensors, each of which are connected with at least one of the others, guaranteeing that there is no isolated sensor. Sensors form quasi-grid networks where the sensors are located randomly within a circle of radius, called the uncertain range, centered at grid points, as in Figure 3.1 (b). In this Chapter, we will illustrate our protocol in grid networks for simplicity, but we will implement quasi-grid networks in our simulation. Also, we assume sensors are trouble-free and have adequate power in the experiments. Hence, the sensors in the network are always connected and all possible events can be detected by some sensors. The communication model for both sensor and robot is the unit disc model. The sensor and robot have the communication range R_s and R_r respectively. In real applications, the transmitting power for a robot is usually stronger than that of the sensor. But in WSANs, since there is no requirement of a direct path between any two robots, it is less practical to set the communication range of a robot to a higher value. This adaptation also reduces the energy cost of the robot. Although we consider energy consumption, we suppose that both robot and sensor have enough energy in our simulation so that they work properly. Also any malfunction or breakage of a node is excluded.

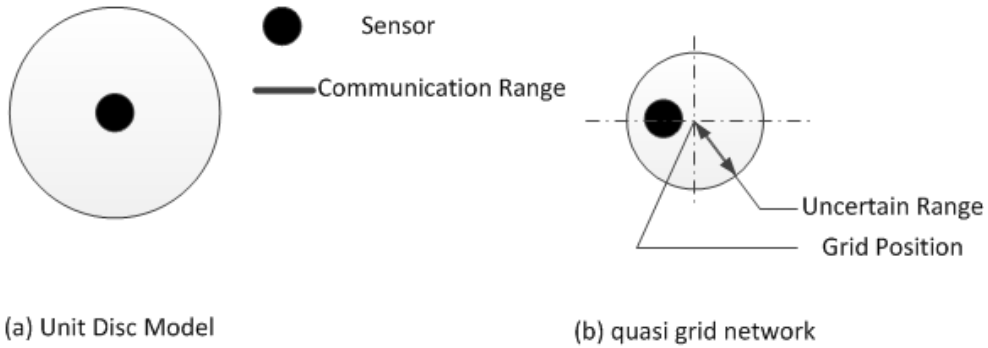


Figure 3.1: Sensor Model

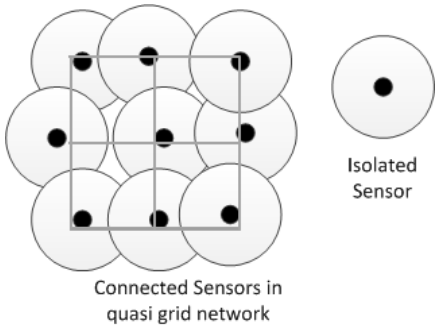


Figure 3.2: Quasi-Grid Networks Model

3.2 The General Idea of DMesh

Initially, we suppose robots are randomly placed in the network. Each robot sends mesh construction packets in four directions to construct the mesh structure so that sensors which detect any event are able to find the nearest robot through the cross lookup to request for services. At a collision point where multiple messages from different robots arrive, the sensor applies a blocking rule to choose the closer robots as its *Service Providers* (SPs). Instead of dropping the location information of other robots, the sensor collects all robot locations into a neighbourhood potential SPs set. This set will be sent to all of the potential SPs in the set (usually it will contain two robots information from either a collinear or vertical collision, as shown in the examples in the rest of this chapter, but for some situations, one sensor may compare three or four robots location theoretically). In this way, a robot will obtain the location information of its neighbouring robots. Then every robot uses its neighbouring robots' locations to construct a localized Voronoi Diagram (VD), showing the area that it will be responsible for. When there is no event, every robot moves toward the centroid of its Voronoi region for even deployment. This network deployment will make the robot better prepared for possible events that follow.

When one event happens, due to the overlapping of the sensing area of multiple sensors, it will be detected by nearby sensors. They will coordinate with each other and the closest one to the event will query the robot location and find its geometrically closest SP by

doing the cross lookup in its mesh cell. It will then inform that robot about the event. After accepting the request, the robot will stop moving toward its Voronoi centroid and will move ahead to the event to handle it. Meanwhile, other robots will keep moving toward the centroid of their Voronoi regions. The selected robot will be blocked from any other events in the future until it finishes its current task. We assume that any event could be disposed of by a single robot and define the cost of time to deal with events as T_{event}. Apparently, as more robots get involved for one event, the time spend on that event will decrease, but in this thesis, we only consider one robot as the event dealer. T_{event} is triggered at the beginning of handling the event and the robot will be inactive until its goes to zero. If the robot cannot finish its task before the next round of mesh update (usually the event requires more than one mesh update cycle), it will be eliminated in the new mesh construction. Once the event has been accomplished, the robot will wait at its location until the next round of mesh update. During moving, the mesh structure is not updated. Sensors have to report the event to mobile robots because the recorded locations in the mesh are out of date and cannot be used for finding SPs, thus a routing scheme for mobile sinks is introduced for service requests during this time interval. After all robots arriving at their endpoints, either an event or the centroid of a Voronoi polygon, a new mesh will be constructed with the current location of available robots.

3.3 DMesh Construction and Update

In order to propose our dynamic location service, we should first construct a static version of the mesh structure for the closest robot. We follow the general steps in [21] to construct the mesh structure in our scheme. However, we will extend and modify its construction process in order to acquire more necessary information to make such a structure dynamically.

3.3.1 DMesh Construction

We assume that sensors and robots are synchronized before the starting moment. At the time T_{begin}, each robot generates four *registration messages* in different directions: East, West, North and South respectively. These messages consist of the location of the robot, direction of the message and a timer called T_{mesh_update} which is used for synchronizing the updating of the mesh structure. To be specific, when the timer ends, sensors will delete the old version of the robot location information and leave space for the new one. This value is related to several parameters of the network, like the scale of the network, number of robots, the moving speed of the robot, the delay of transmitting a message, etc. If the value is too small, it will cause a collision between the previous and new mesh structure; otherwise, it will make the location information out-of-date. Both of these two situations will deteriorate the performance of our mesh. However, optimization of this variable is out of the scope of this thesis. We will give the value in our simulation configuration.

After the furthest sensor in one direction of a robot neighbourhood receives a message, it will store the location and timer to its local storage. Then, the sensor will retransmit the message to its foremost neighbour according to the direction information by using the Greedy-Face-Greedy (GFG [5]) protocol. In some situations where two *registration messages* from the same robot reach the same sensor, the sensor will forward these messages in different directions. Consequently, these messages will hit the border of the network in corresponding directions. The sensor at the border will deliver the *registration message*, switching from greedy to facing because there is no local node further than itself in the direction of message. Due to the nature of GFG, these *registration messages* will be forwarded in a clockwise (or counter-clockwise) direction and will eventually cover the whole boundary of the network. Figure 3.3 shows how it works when there is only one robot in the network.

When two or more *registration messages* from different robots arrive at the same sensor,

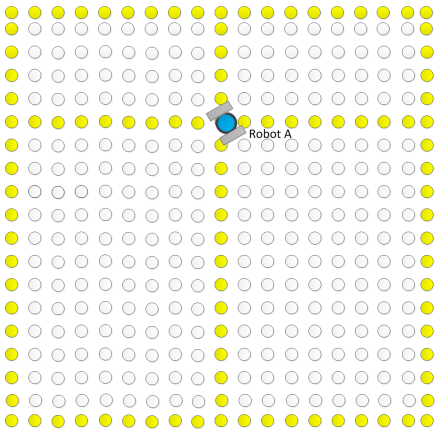


Figure 3.3: A Robot Sending Its Register Message to the Grid Network

this sensor will be called as *collision sensor* and the block rule will be applied which is officially formalized in [21] as:

" A common node u of the residing rows/columns of two SPs a and b(a $\neq$ b) stops the further propagation of the information of a, iff$|ua| > |ub| \vee |ua| = |ub| \wedge colline(a,b) \vee |ua| = |ub| \wedge \neg colline(a,b) \wedge horizon(b)$, where *colline*(a,b) and *horizon*(b) denote the case that a and b are (vertically or horizontally) collinear and the case that the involvement of b is along the horizontal direction, respectively. When this blocking happens, we say 'b blocks a at u' and denote it by a $\overset{u}{\leftarrow}$ b or b $\overset{u}{\rightarrow}$ a."

Simply, the *collision sensor* will compare the location of every robot and choose the closest one as its SP. Moreover, it will put all robot locations into a set as *Neighbouring Robots*. It will send the set to all the robots in this set. The transmitting process will be greedy to these robots using GFG. Hence, the robot is capable of capturing the location of its neighbouring robots. If a sensor receives a new *registration message* but finds the previous message of the further robot has arrived earlier, this *collision sensor* will send a *revocation message* to erase the previous robot location. This message will track the path of the earlier-arrived *registration message* until it reaches the endpoint of its propagation. In Figure 3.4, three robots try to construct the mesh structure.

In the construction process, there are three collision sensors in the figure which are

marked out as black nodes. Collision sensor 2 will compare the distance from itself to robot A and robot B. Since the sensor is closer to robot A than to robot B, it will orthogonally block the westbound message from robot B. That is because for sensors farther west than collision sensor 2, robot A is geometrically closer than robot B. Therefore these sensors have no need to store the location information of robot B. In the asynchronous scenario where the *registration message* from robot A arrives earlier, collision sensor 1 will retrieve its message along the delivery path of the *registration message* from robot A shown as sensors with a red perimeter. Collision sensor 1 will forward the message from robot B in advance. Collision sensor 3 applies the block rule in a collinear situation. It is located almost equally from robot A and robot C. Thus, sensors farther south than collision sensor 3 store the location of robot C while sensors farther north of there conserve the location of robot A.

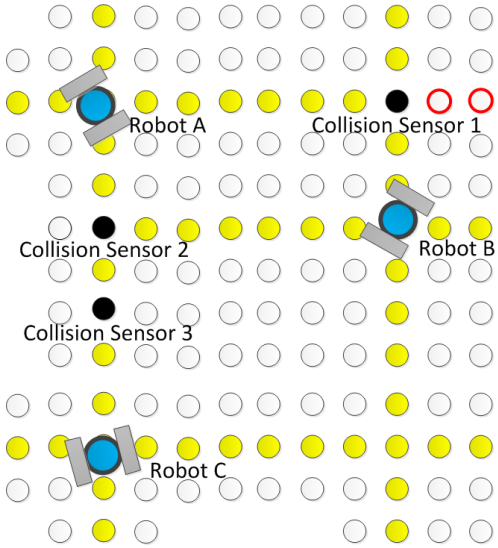


Figure 3.4: Collision Sensors During the Construction of the DMesh

According to the location of neighbouring robots, a robot will start to calculate the Voronoi Diagram and its center. This division of a network can more clearly show which sensor is closest to which robot. The benefit of moving a robot to its Voronoi centroid is that it minimizes the expected distance for a robot to any point within its Voronoi polygon. We assume that the robot knows the size of the ROI and the relative location

of itself in the ROI. If our ROI is defined by 0 to *LENGTH* in the X coordinate and 0 to *WIDTH* in the Y coordinate, robots create an array of virtual pixels p_{ij} for every integer location in the ROI, where i and j are integers indicating the X and Y coordinates of point p, $0 < i < LENGTH$ and $0 < j < WIDTH$. A robot compares the distance of all p_{ij} to itself with the distance of all p_{ij} to its neighbouring robot. Then the centroid of the Voronoi polygon of robot r can be calculated by using distributed versions of the equations 2.2 in Chapter 2:

$$Centroid_x = \frac{\sum_i^{p_{ij} \text{ is close to r}} \text{x of } p_{ij}}{\text{number of these } p_{ij}} \tag{3.1}$$

$$Centroid_y = \frac{\sum_i^{p_{ij} \text{ is close to r}} \text{y of } p_{ij}}{\text{number of these } p_{ij}} \tag{3.2}$$

Each robot will regard the centroid as its endpoint and move to it for better performance of event coverage until there is an event reported to a robot. In order to make the movement of all robots synchronized, we predefine a time called T_{wait}. A robot will stay at its location until the T_{wait}. The value of T_{wait} should be neither too large nor too small. The minimum value should allow robots to accomplish the mesh structure which is related to the scalability and transmitting delay of the network. On the other hand, since $T_{period\_update} - T_{wait}$ is the time for robots to move, T_{wait} should be adequately small so that robots have enough time to reach their endpoint. For convenience, we define the *time ratio of construction* as:

$$k_t = \frac{T_{wait}}{T_{period\_update}} \tag{3.3}$$

This ratio is constrained to be from 0 to 1, which exactly describes how much time robots will stay at their locations in the current mesh structure. During movement, the robot will change its endpoint to the event location whenever it receives an event report.

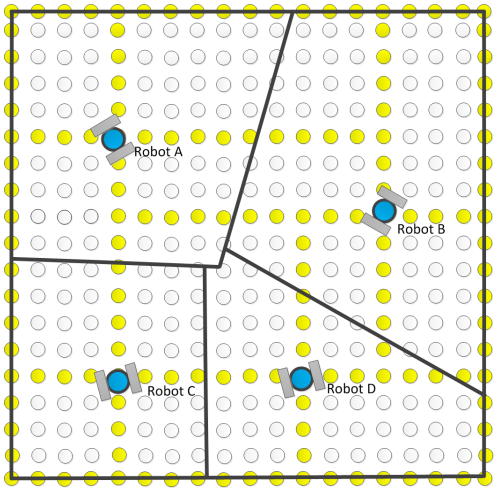


Figure 3.5: Voronoi Diagram above a Mesh Structure with Four Robots in a Grid Network

Figure 3.5 shows how a mesh structure is constructed with four robots in grid sensor networks. At a certain time, the robots deployed themselves separately as shown in the figure. The yellow sensors received the *registration messages* from the robots and became parts of the mesh structure at the current update cycle. The black lines in the network shows the Voronoi Diagram of the network above the mesh structure. This division will help the robot find its own centroid within its Voronoi polygon.

3.3.2 Pseudo Code of DMesh Construction and Update for a Robot

Here we present the pseudo code in Algorithm 1 for robot R.

Algorithm 1 DMesh Construction and Update, robot R

```
1: while true do
2:     if Timer = T_mesh_update and R is not busy then
3:         R sends registration messages to four directions
4:     end if
5:     if Timer = T_mesh_update + T_wait then
6:         R calculates its Voronoi diagram, sets its endpoint
7:         as the Voronoi centroid and starts to move
8:     end if
9:     if R receives event report then
10:        endpoint ← event location
11:        R moves to endpoint and becomes busy
12:    end if
13:    if R is on move then
14:        runs Distance-sensitive Routing for Multiple
15:        Mobile Robots (Algorithm 3)
16:    end if
17:    if event is accomplished then
18:        R becomes available for next events
19:    end if
20:    if R gets neighbouring robots position then
21:        adds these positions in Neighbouring Robots set
22:    end if
23: end while
```

3.3.3 Cross Lookup in DMesh

After the mesh structure is completed, the target for a sensor is to find the closest *Service Provider* through the mesh structure if there is an event detected. Suppose a sensor S detects an event and after coordinating among its neighbouring sensors, it will request for service if and only if it is the one closest to the event. It will send a *Service Request Message* which simply contains the event request, to its neighbours in four directions (See Figure 3.6). These search messages will be delivered using GFG until they reach the border of the mesh cell. Then the sensor where the search message stops will send the recorded location of service robot back to the sensor S. If it does not find any mesh sensor, it will reply to S with a failure notice. The sensor S will locally compare the locations of robots in these reply messages and choose the robot closest to the event. S will also send an *Event Report Message* to the robot, using GFG. The robot that receives such a message will set its status to busy and block itself from future request until it finishes the current event. If S has the location information of a robot, which means that the robot has already moved away from its mesh constructing point, S will directly sent an event report to the robot. This message may get redirected on its path to the robot.

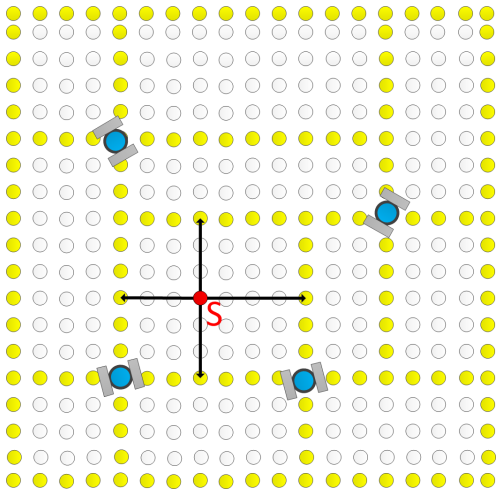


Figure 3.6: The Lookup Process in a Mesh Structure with Four Robots in a Grid Network

3.3.4 Pseudo Code of DMesh Construction and Update for a Sensor

Here we present the pseudo code in Algorithm 2 for sensor S.

3.3.5 DMesh Update

3.3.5.1 Stable Deployment

Once the mesh structure is accomplished, the robot will prepare to move to its endpoint, either the event or its Voronoi centroid. Supposing that one event happens at a specific time in the completed mesh structure, the closest robot to the event will move to the event. Meanwhile, it will become unavailable for upcoming events until it finishes its current one. Therefore, possible subsequent events in its Voronoi region will keep requesting in the current mesh update period but cannot get a response. In this situation, the event response delay will be at most T_{mesh_update} because other robots will construct a new version of the mesh structure. The busy robot will not send a *registration message*, so it will be eliminated from the structure until the first update cycle after it finishes the current event. Consequently, other robots cannot capture its location information through the collision sensor in the mesh structure. Thus, the new division of the Voronoi Diagram will lose this robot and lead to the location adjustment of all other robots.

Definition 1. *The stable deployment of n-robots in the network is the one where every event-available robot is located in its Voronoi centroid and requires no further movement. The number of robot n is called the rank of stable deployment.*

With the stable deployment, a robot reaches an optimal point where it does not have to move until an event happens in the network which breaks this balanced structure. Figure 3.7 a) shows the stable deployment of 4 robots. In the stable deployment, robots

Algorithm 2 DMesh Construction and Update, sensor S

```
1: while true do
2:     if Timer = T_mesh_update then
3:         reset S
4:     else
5:         if S gets registration message then
6:             if not receive before then
7:                 stores robot location and updates timer
8:                 retransmits the message
9:             else
10:                S applies blocking rule and checks whether
11:                to send revocation message
12:            end if
13:        end if
14:        if S detects event then
15:            if movingSP is not null then
16:                S informs its movingSP with event location
17:            else
18:                does cross look-up
19:            end if
20:        end if
21:        if S gets cross look-up message then
22:            if S is part of mesh then
23:                reports SP location in S to request source
24:            else
25:                retransmits the message
26:            end if
27:        end if
28:        if S gets SPs information then
29:            collects SPs location, chooses the closest one
30:            and sends event report to the SP
31:        end if
32:        retransmits the message
33:    end if
34: end while
```

are sparsely distributed and every robot is responsible for the event happening in its own Voronoi region in which all sensors are closer to this robot than to other robots. Moreover, being located at the centroid of the Voronoi region minimizes the expected distance of movement of the robot to any event in the whole Voronoi cell. The proof is provided in [8]. In Chapter 4, our result shows that the number of nodes in such a division is almost the same when we consider a uniform probability distribution of events. For message cost, maintaining the mesh structure is the same as the static mesh: for each robot, it will send four *registration messages*; for sensors in the residue row and column of robots, they will only deliver these messages. On the other hand, these messages have to be sent because this is how the robot gets information about other robots. However, periodic location updates of robots will waste the energy of robots and may deplete the energy of a certain group of sensors.

When no event happens in the network, the robot will stay at its position which means the location information of the robot stored in the sensor is accurate for the next T_{mesh_update}. In this situation, the robot will double its T_{mesh_update} in the *registration message* for the next update period to lower the frequency for location updates. Therefore, the message cost for both robot and sensor will be reduced while the whole system still keeps the latest accurate locations of robots. This process will continue if no event happens until the $2 \times T_{mesh\_update}$ exceeds the maximum tolerated delay of event for our system $Delay_{max}$. When any robot receives the event report, it will move to the event. In a small system with few robots, eliminating one robot from the mesh will lead to locomotion of all the other robots. But in considerable large systems, some robots may not be able to detect the movement of the robot which goes toward an event because their Voronoi cells do not change during the next period. As a result, robots geometrically adjacent to the robot which goes to an event will double their T_{mesh_update}, while other robots will still keep their T_{mesh_update} the same. This will lead to synchronized chaos throughout the whole mesh structure. To avoid such a mess, the robot receiving the event report will send a notice

message containing the moving status and its *Neighbouring Robots* to robots in this set. Robots receiving such a notice will stop the doubling process and set the T_{mesh_update} as the initial value. To avoid duplication of this notice, each robot will eliminate robots in *Neighbouring Robots* in this notice message from its own set. Then robots will forward this notice message to the robots left in its own *Neighbouring Robots* set. In this way, all robots will realize that an event happens and set T_{mesh_update} back to the initial value.

3.3.5.2 DMesh Degradation

Definition 2. *k-th mesh degradation happens when stable deployment of n-robots degrades to robot deployment of $(n-k)$-robots.*

The stable deployment will be broken only if there is an event happening. If an event happens, the closest robot will move to it. This robot will become unavailable and will lose contact with other robots during the event process. If at the next update cycle, the robot is not able to finish the event, the n-robots deployment will eventually regress to $n-1$-robots stable deployment as illustrated in Figure 3.7. Robot B will take the event since it is the nearest one to the event. It is not able to deliver its *registration messages* because of event handling in the next several update cycles. Thus, our 4-robots stable deployment on the left will become the 3-robots stable deployment in a few rounds of updates on the right.

3.4 Distance-sensitive Routing for Multiple Mobile Robots

Although the mesh structure allows a sensor to have a high probability of finding the closest robot, during the update of the mesh, the sensor is not able to track robots because the location of the robot changes during its movement to the event or the centroid. During the move, we keep our existing mesh structure and make no update. Therefore, a sensor in

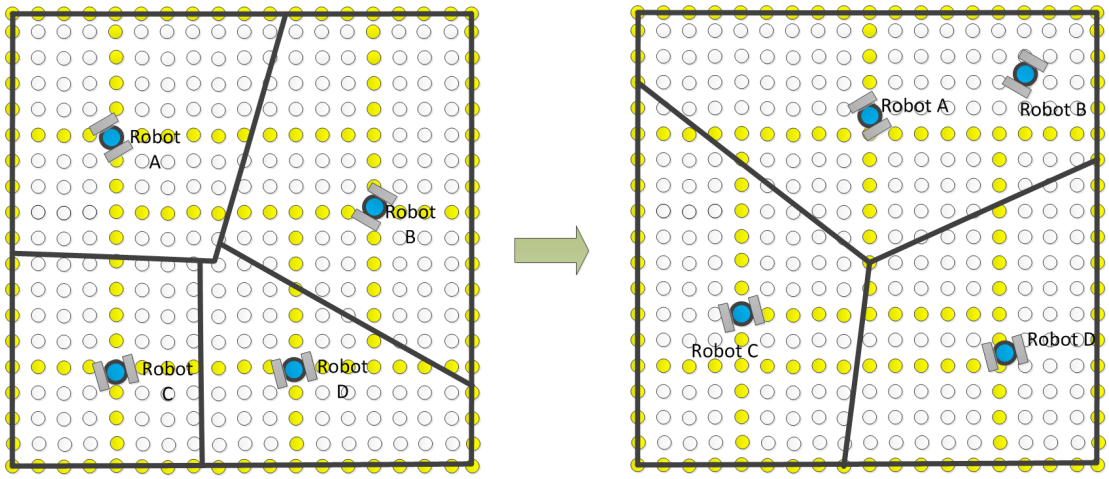


Figure 3.7: 4 Robots Degrade to 3 Robots

the mesh structure only stores the location of its closest robot before robots move. Since the event is totally random for any time, we need to guarantee that any event request in this interval of mesh update will be received by its closest robot. Thus, we should adapt our protocol so that it allows event monitoring during the movement of a robot.

After the robot decides its endpoint, it starts to move in the current mesh updating cycle. While moving, the robot monitors its neighbouring sensors and calculates the *minimal broken time* according to its direction of motion and speed. The *minimal broken time* will decrease as the real time increases. If it goes to zero, a sensor will lose contact with the robot; if the robot meets a new sensor, it will add this sensor to its neighbours list, sets the old *minimal broken time* to zero and re-calculates the new one. Then, in both situations, the robot will broadcast the *location update* message which contains its robot ID, its current position, its endpoint and the T_{mesh_update}. The sensor receiving the *location update* message will store the robot location as the moving service provider (*movingSP*), the endpoint and the T_{mesh_update}. The new neighbouring sensor or the broken one will refresh its neighbours list. The *location update* message will be retransmitted by the sensor if the location of the robot or endpoint is different from the location recorded by the sensor.

Our protocol for moving routing is based on flooding. Even though we decrease the frequency by predicting the connection of the robot, the robot may still potentially broadcast

the message to the whole network which will cause severe congestion or even a network crash. In fact, there is usually more than one robot in the network in order to reduce the burden per robot and to elevate the real-time performance of event response. Each sensor may receive multiple *location update* messages from different robots which may lead to unnecessary overhead. Therefore, the *location update* message of robot should be constrained within its Voronoi cell which eliminates collision among the different versions of *location update* messages and dramatically reduces the overhead of the message.

Before the robot moves, it will broadcast its neighbouring set and its own location with the corresponding ID. The sensor that receives them will choose its *movingSP* by calculating its distance to its neighbouring robots in the neighbouring set. It will record the robot ID and its location. Retransmission of the message will happen only if the chosen *movingSP* is the same as the source of the message based on the robot ID. In this way, a sensor is aware of which Voronoi polygon it belongs to and this Voronoi division will not be changed until the robot arrives at its current endpoint. Later, the sensor receives the *location update* message will compare the robot ID in the *location update* message and its *movingSP* ID. Only when the robot ID in new message is the same as the one of its *movingSP* will the sensor update the location of its *MovingSP* and re-transmit the new message.

If there is an event reported to the robot during its move to the centroid, the robot will abandon its endpoints and switch to the event location because of the higher priority. The robot also stops current *minimal broken time* and sends a *recovery location update* message which contains its current position and new endpoint to its previous endpoint using GFG. Because the endpoint is a virtual point, the message will be delivered to sensors around the endpoint. According to the property of GFG, the sensor that is the closest to the old endpoint will receive the same *recovery location update* message twice. This sensor will treat itself as the *Anchor Sensor*. The queries from sensors which lose contact with the robot after the robot changed its endpoint and try to report event to the previous

endpoint, will be re-routed to new endpoint of the robot by the *Anchor Sensor*. Once the robot accepts the event report, it will be blocked from any other requests from the sensor. Therefore, during this period of mesh update, any sensor that tries to access the robot will be denied and any other event report will be suspended. Since the complete execution of the event usually requires more than one mesh update cycle, the sensor will do the cross look-up in next cycle through the new mesh structure. In the meanwhile, other robots targeting coverage of the ROI in their own Voronoi polygon will still monitor the event during the interval of mesh updating.

Figure 3.8 illustrates the process of location update. The robot R will notice all the sensors in its Voronoi cell as the green area before it moves. During the move to the Voronoi centroid, it changes direction at point P_1 because an event happens. Therefore, at P_1, it will send a *recovery location update* message to the centroid. This message will be delivered around the centroid marked by the blue arrow. The closest sensor (the black node) will be selected as the Anchor Sensor.

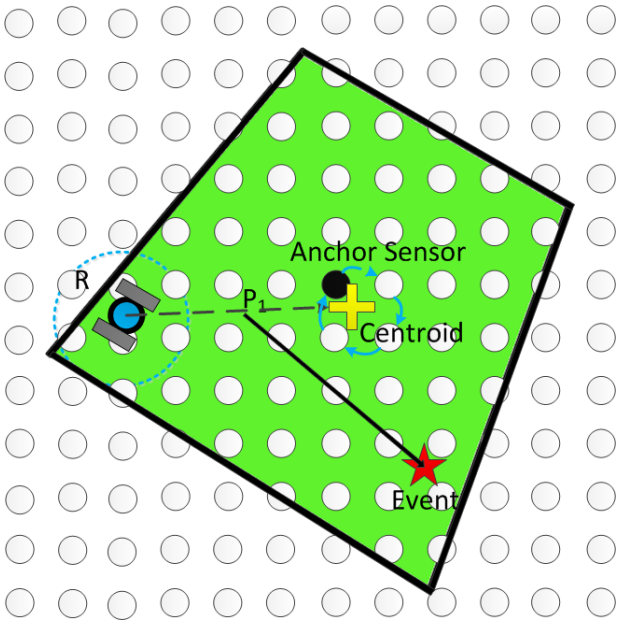


Figure 3.8: A Robot Updates Its Location within Its Voronoi Polygon

Before the sensor which is closest to the event does the cross lookup as illustrated in

Figure 3.6, it should check whether the *movingSP* is null or not. If it is null, it will do the cross lookup; if it contains a certain robot, it will use GFG to report its request. During the process of cross lookup, if any sensor contains the *movingSP*, it means the location of the robot has already been changed. As a consequence, the cross lookup will stop and this sensor will send the location stored as *movingSP* to the source. Then the source will route the event report to the closest robot directly. If the robot is not at that location, sensors will send this report to the endpoint of the robot.

3.4.1 Pseudo Code of Distance-sensitive Routing for Multiple Mobile Robots

Here we present our pseudo code in Algorithm 3 for robot R and Algorithm 4 for sensor S:

Algorithm 3 Distance-sensitive Routing for Multiple Mobile Robots, robot R

```
1: while R is moving do
2:     calculate the minimal broken time
3:     if minimal broken time = 0 then
4:         R broadcasts location update message
5:     else if new neighbour added then
6:         R stops current minimal broken timeout
7:     else if new event received then
8:         R sends recovery location message to old endpoint
9:         and stops current minimal broken timeout
10:    end if
11: end while
```

Algorithm 4 Distance-sensitive Routing for Multiple Mobile Robots, sensor S

```
1: while true do
2:     if S gets location update message then
3:         if S is a new node OR S is a broken node then
4:             if movingSP in message is from its Voronoi polygon then
5:                 keeps location in message as movingSP
6:             end if
7:         else if endpoint in the message is changed then
8:             retransmits the message
9:         end if
10:    else if S gets recovery location message then
11:        if S receives the same message twice then
12:            sets itself as anchor node and stops retransmitting
13:        else
14:            S forwards the message using GFG
15:        end if
16:    end if
17: end while
```

Chapter 4

Performance Analysis

In this chapter we will evaluate the protocol by implementing our protocol in JBotSim [7] which is a Java-based library, to realize and visualize the simulation. Because our protocol is derived from iMesh, we will compare our DMesh with iMesh. We will also compare DMesh with the Voronoi-based protocol-DDMA [25]. First of all, we will introduce some metrics for the estimation of performance. Then we will configure our network setting and give simulation steps. Finally, we will use these metrics for the analysis of properties of our protocol.

- Response Distance (ResD) assesses the needed distance for a robot to arrive at the event position once the event is reported by the sensor. Since the robot's moving speed is constant in our simulation, the distance can be considered as a measure of time required by a robot to respond to an event. Less Response Distance means less time required for robots to arrive at the event. As we suggested in Chapter 1, in some applications dealing with emergencies which are delay sensitive, a robot should arrive as soon as possible. Aside from that, the real-time property is the motivation for us to develop the DMesh protocol.

- Recovery Distance (RecD) estimates the distance for event-available robots to cover

the whole ROI with stable deployment after some of the robots react to events. Also, this distance represents the time for a robot to add itself to the currently existing mesh structure after it finishes dealing with an event. This distance value embodies the time consumption for deployment adaptation.

- Load of Robot (LR) indicates the number of sensors that a robot will be responsible for in the stable deployment. Because of the uniform distribution of our sensors, this also represents the size of the area that a robot is responsible for handling in an event. If the Load of Robot for every robot is the same, the energy cost for possible events among the robots will be balanced and the system lifetime will be prolonged.

- Robot Message Cost (RMC) and Sensor Message Cost (SMC) are the number of messages that one robot and one sensor uses per round in DMCU and DRMMR. Because we update our structure periodically and require robots to inform sensors of their new locations during a move, both RMC and SMC will increase as a result and they are regarded as the price to achieve our protocol.

4.1 Network Configuration

We implement our protocol in an ROI with size A using quasi-grid sensor networks where the location of sensors is random within a circle with a radius of 2 meters centered at a grid point. This is more realistic than the strict grid networks. We are always willing to generate grid networks in application because of its benefits, but when the sensors are about to be placed in some extreme scenario where it is hard to approach people, like a remote mountain area, under the sea or in another unmanned region, the carriers can only install sensors within a geometric error location. Moreover, it is easy for us to transfer this network to a random and uniform sensor network by adding another group of sensors and slightly adjusting the GFG protocol so that the mesh registration message can cover the

whole border of the network. This kind of network can be generated by adding a random variable with a positive value into both the X and Y coordinates of the robot's location.

The next thing that should be mentioned is that the probability distribution of events is uniform within the ROI. In some applications, we may treat the possibility of events happening as uneven based on technology or empirical analysis. This will be discussed in our future work. Events may happen any time.

Our experience will be conducted with the number of robot N being selected from 1 to 8 for number of sensors n. In this sensor network configuration, the communication range R_c of the robot and the sensor will be 35 m. This makes sure that each sensor can just connect with its orthogonal neighbours but not the diagonal ones, which simplifies our GFG protocol in implementation while embodying the attributes of our protocol. The simulation was repeated 20 times with different initial positions of robots and events, and the results were averaged. Below, all of the network parameters are shown in the table:

Table 4.1: Static Network Configuration

Parameters	Definition	Values
A	Network Size	470m * 470m, 620m * 620m, 770m * 770m, 920m * 920m
N	Number of Robot	1 to 8
n	Number of Sensor	225, 400, 625, 900
D_s	Distance of Sensor Gap	30m ± 2m
R_c	Communication Range	35m
d_c	Communication Relay	10ms
v_r	Speed of Robot	10m/s
k_t	Constructing Time Ratio	0.5

Here, we show different stable deployments of robots with N varying from 3 to 8 with a sensor network $n = 20 * 20$ in Figure 4.1.

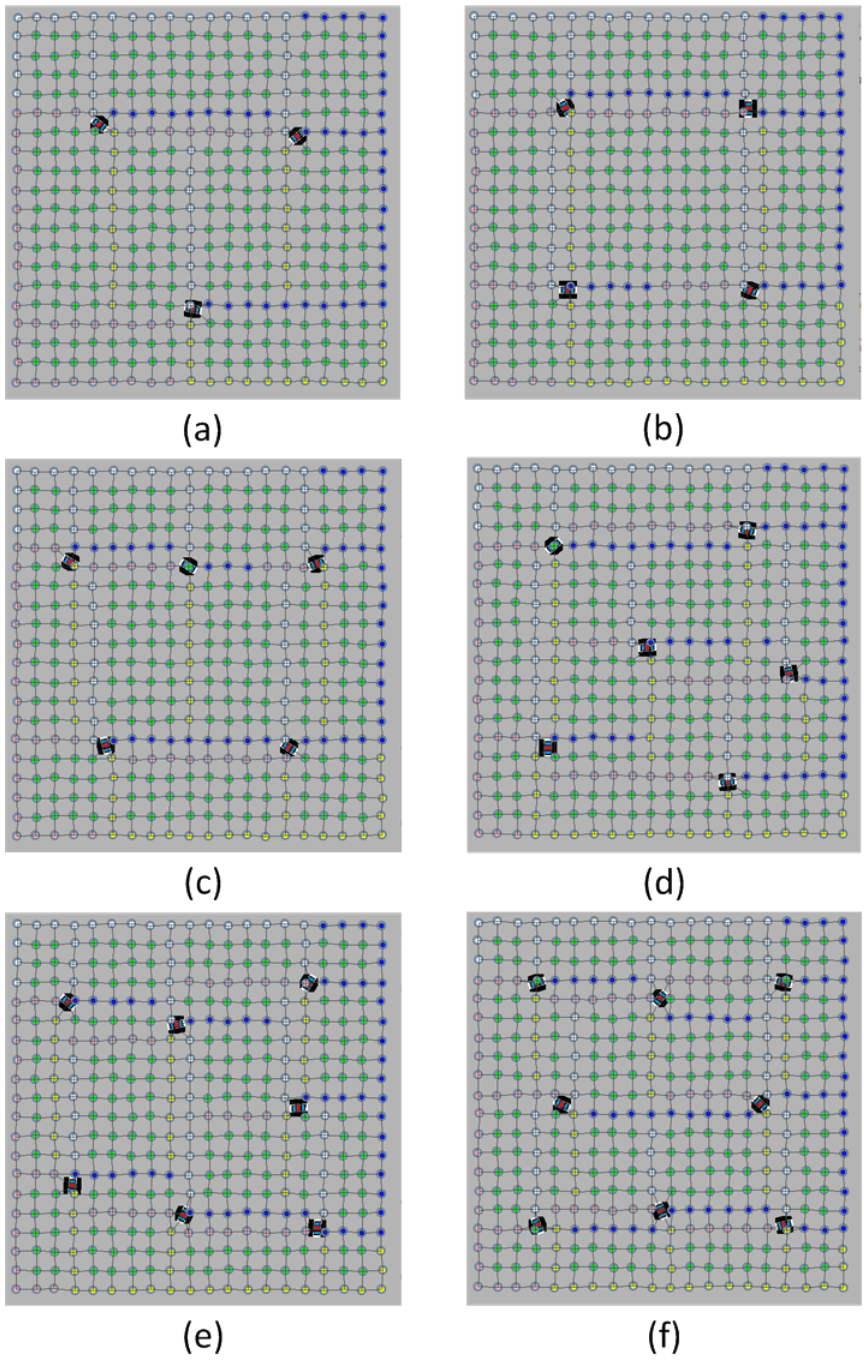


Figure 4.1: Stable Deployment of 3 to 8 Robots

4.2 Result and Analysis

4.2.1 Response Distance

In order to evaluate response distance, we randomly add an event to our network. For DMesh, we suppose the event happens after all the robots have moved to their Voronoi centroid which means that robots are in the stable deployment. For iMesh and DDMA, since they do not make any change to the robot locations if no event happens, we simply randomly deploy all the robots and the response distance for both protocols are the same. As shown in Figure 4.2, the averaged response distance for a robot to reach the event drops proportionally to the increasing number of robots. That is because the more robots that are involved in the network, the smaller area that an individual robot is going to be responsible for. If there is only one robot in the network, it has to respond to all the possible events in the whole network. If there are two robots, each of them only have to cover one of two areas separated by the perpendicular bisector of their connecting line. As more robots get involved, more divisions will exist in the network and there is less area for each robot to cover. Comparing the Response Distance between a random distribution of robots and our stable deployment, we prove that our stable deployment considerably decreases the distance a robot is about to move once the event happens. By leading a robot to the centroid of its Voronoi cell, there is about fifty meters less than in the random topology on an average of 20 experimental iterations for different numbers of robots. For example, if there is only one robot, by just driving it to the centre of network, the response distance drops from about 350 m to 280 m. When 8 robots are deployed, even though the response distance decreases substantially compared with a few robots, our stable deployment still decreases by about 50 m. In terms of percentage, it reduces the response distance from 17.8% (single robot) to 33.1% (eight robots). It is proven that our stable deployment in case of DMesh largely decreases the distance a robot has to travel once an event occurs.

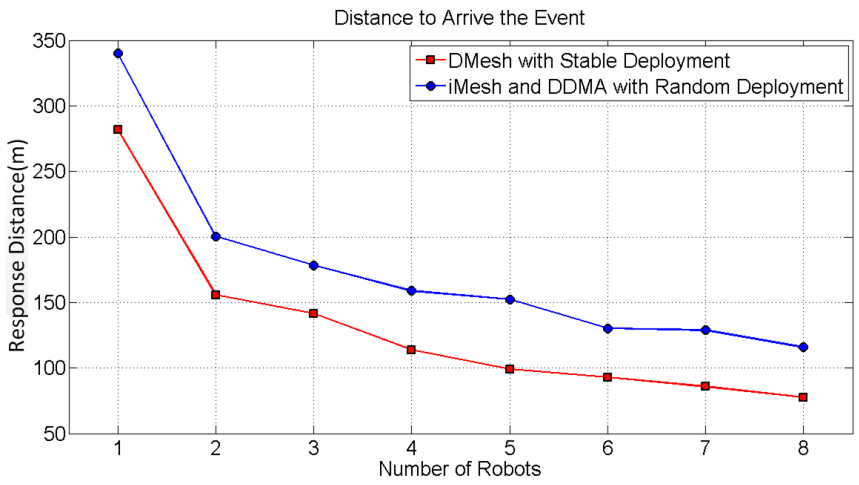


Figure 4.2: Moving Distance When the Number of Sensors is 400

For the ResD metric, we will also change our network area and total number of sensors with a constant interval distance between sensors. Figure 4.3 depicts the advantage of DMesh in response distance reduction for different sizes of sensor network with 5 robots. In the 225-sensor network, the random deployment causes 140 m movements; while in stable deployment of our protocol, such a value declines to around 80 which is less than the 60 percent of random deployment. Although this ratio rises a little to above 70 percent when the scale of sensors is enlarged to 900, it is reasonable for us to add more robots in large-size scenarios for the shorter responding time.

4.2.2 Recovery Distance

As we mentioned in Chapter 3, our mesh structure degrades if any robot receives an event request. In our simulation, with 1 to 8 robots, we observe that robots take about 5 mesh update cycles to recover and form a sub-class deployment. For a small number of robots (e.g., less than 5), the "unstable" status only lasts for 2 to 3 cycles. We suppose that servicing the event demands 5 mesh update cycles so that these robots which receive the event will not be able to go back to impact robot deployment. Hence the moving distance per robot to reach the stable deployment can be accumulated. Our experiments were

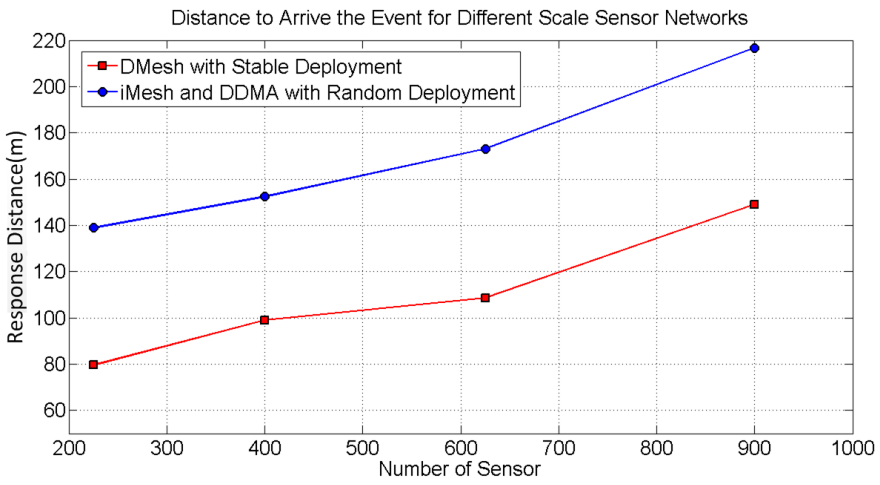


Figure 4.3: Moving Distance When the Number of Robots is 5

conducted in stable deployment with one or two events happening randomly. Because we need at least one robot left available in the network to represent the recovery property, our recovery distance analysis starts from 2 robots for one event and 3 robots for two events. The two gradually descending lines in the Figure 4.4, indicate that a robot moves a shorter distance to recover from mesh degradation because the interval distance between any two robots shrinks as the number of robots increases. If one event happens and the closest robot moves to the event, it leaves a coverage hole for event coverage in the mesh because in the robot's own Voronoi polygon there is no possible event handler and other robots need to cover this area. Now, with one less robot available for event coverage, other robots coordinate to repair the coverage hole by forming a sub-class stable deployment. With more robots in the network, required moving distance for other robots is reduced as depicted in the blue line in Figure 4.4. When two events happen, the two robots closest to the events get occupied by the two events during the same time, and the remaining robots travel longer distances to reach the sub-class stable deployment as shown in the red line in Figure 4.4. It is because that the other event-available robots need to recover more areas than the one with single event.

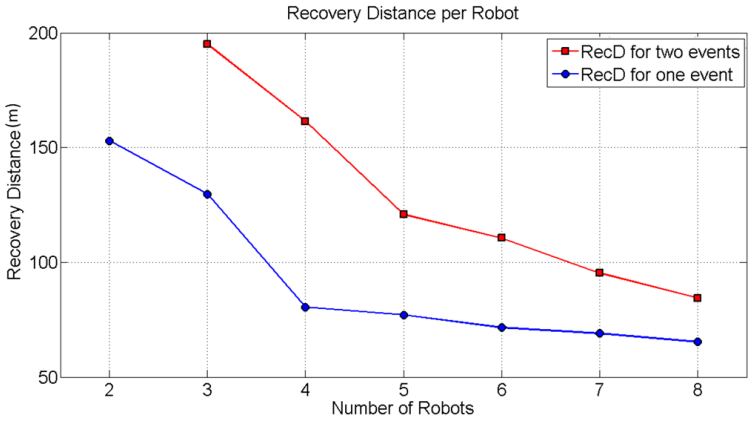


Figure 4.4: Recovery Distance per Robot

4.2.3 Load of Robot

A balanced load of robot is another valuable characteristic of DMesh. Figure 4.5 demonstrates how many sensors will potentially report the event to a certain robot. Because our sensors are uniformly distributed in the ROI, this also implies the size of area a robot will be responsible for. Actually, this is a side effect of setting the endpoint of event-available robots to the centroids of Voronoi cells and uniformly distributed sensors. The Voronoi division causes sensors to be informed about the area in which a robot is closer to them than other robots, but the size of these separated areas are not equal at the beginning. Nonetheless, after the mesh has evolved through a few updates, it will divide the ROI into approximately equal areas. In a one-robot scenario, the robot has to take the requests from all the sensors; for two robots, both of them take care of half of the sensors; for more robots, their sensor loads are almost the same. We point out that even through this value rises and falls in a small range except for 1 to 3 robots, in robots numbering 4 to 8 with a deviation of less than 5 sensors from the theoretical average value, the balancing effect for the sensor load of robot is acceptable. Notice that the event is totally uniform and random in the ROI and this property averages the chance for robots to take care of events which extends the lifetime of our system as a whole.

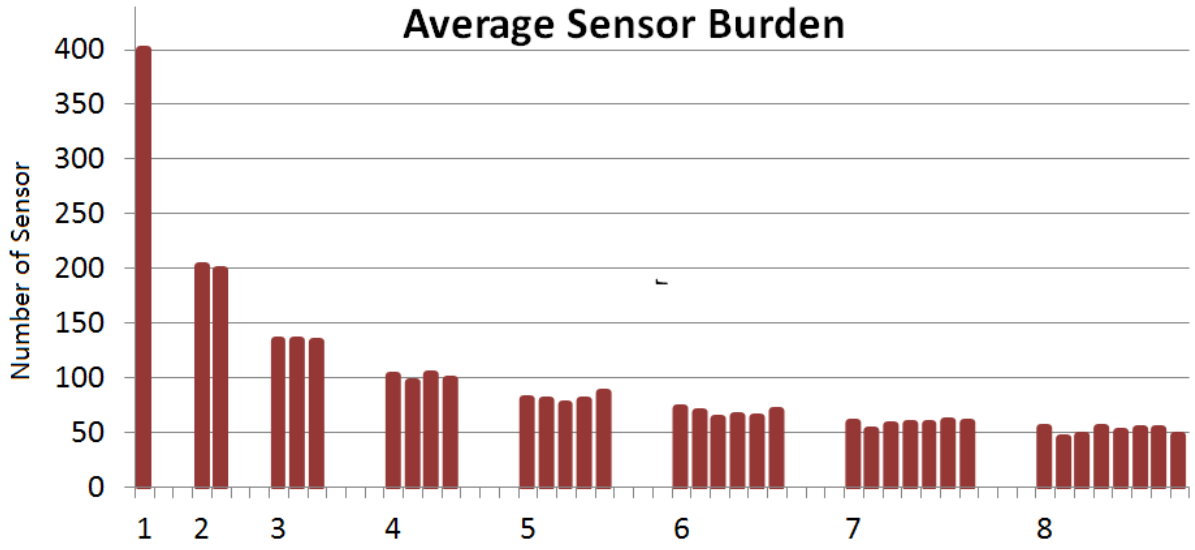


Figure 4.5: Load of Robot

4.2.4 Message Cost

In this section, we will look at the message cost for the robot and the sensor. In Figure 4.6, the green line, blue line and red line are the message cost of DDMA, iMesh and DMesh respectively. For the iMesh, we make robots update their mesh periodically by relocating robots' positions at the beginning of every mesh update cycle. Besides, we make robots in DDMA move to the centroid of Voronoi cells as the covering task, so robots will update their positions once their locations change. We calculate the message cost in 5 cycles before the robot in DMesh reaches the stable deployment and average them into the message cost per update cycle. Also, the message cost is accumulated without an event. Clearly, the red line is higher than the blue and green lines. For iMesh, the message cost is constant at 4 because every robot propagates its location information in four directions. In DDMA, a robot will broadcast its location to the whole network after it moves to a new location. Comparing DMesh with iMesh, except for the mesh construction, the message cost in DMesh for a robot is used for routing during moving (DRMMR). The message cost in DMesh demonstrates a slow decline: when the number of robots is 1, the message cost is about 10; when the ROI has 3 robots, the message cost is about 9; when the robots

were increased to 8, the cost goes to about 7 and tends to settle down. As the number of robots in the network increases, the distance for a robot to reach its position in stable deployment decreases. Since our sensors in the network have a uniform distribution, the chance for a robot to meet new sensors or lose sensors is proportional to the distance that a robot moves. Although our protocol seems to almost double the message cost of iMesh, it is important to mention that this process only lasts for the first few update cycles before stable deployment. If there is no event happening, the message cost of a robot will drop to 4 to maintain the mesh structure. More sophisticated design, like doubling the mesh update cycle discussed in Chapter 3, will decrease the message cost further.

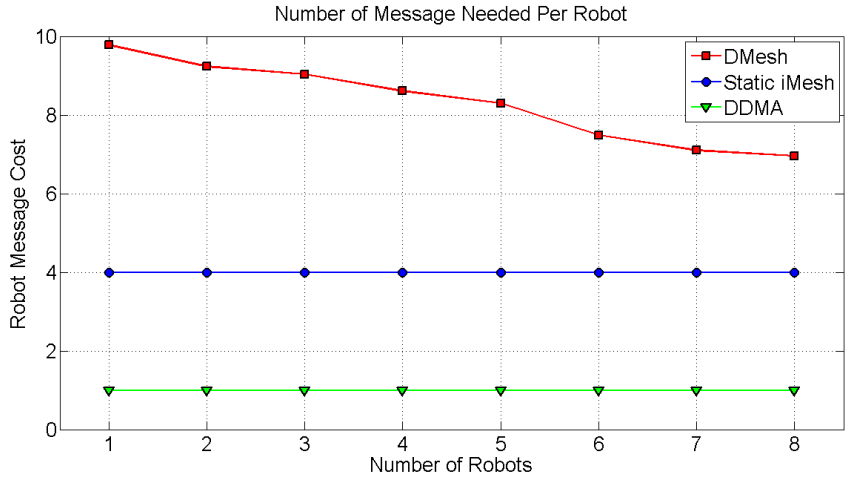


Figure 4.6: Message Cost per Robot

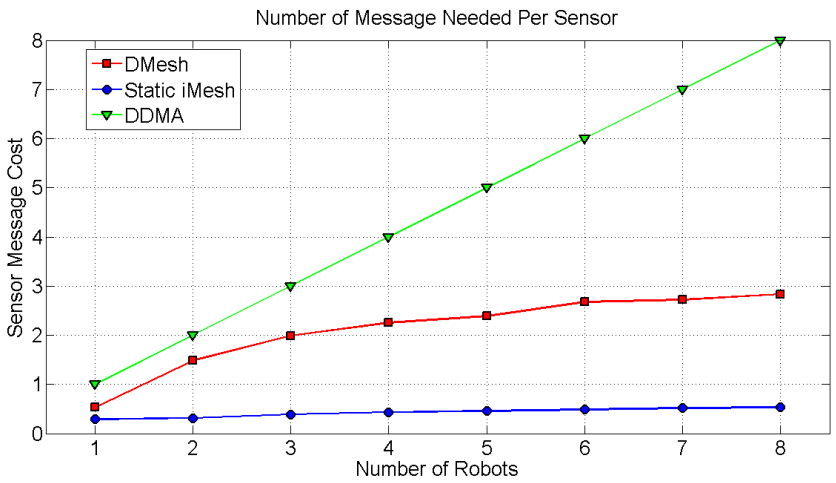


Figure 4.7: Message Cost per Sensor

Figure 4.7 depicts the sensor message cost for DDMA, iMesh and DMesh. Because DDMA requires a sensor to know information about all the robots, every robot will be flooding its location to the whole network. Thus the SMC in DDMA is proportional to the number of robots. In iMesh, the message cost for a sensor is just made up by the group of sensors in residual rows and columns of robots and the edges of network, which deliver the *registration messages*. Hence, for the whole network of numerous sensors, the cost is relatively low with a tiny rise. For DMesh, the message cost is in between the DDMA and iMesh. The augmentation of the message cost for a sensor stems from several perspectives. For the mesh construction, introducing more robots requires more registration message deliveries which is the same as iMesh and is shown in the growing tendency of the blue line. On the other side, message transmission as a Moving Service Provider is the major part which will be increased as the number of robots goes up. Similar to RSC, this extra cost will be cancelled as the robot's deployment becomes stable and kept in a relatively small value (below 0.5 as iMesh).

Compare to RMC and SMC, even though the robot message cost for DMesh is higher than DDMA, the message cost for sensors in DMesh is reduced which has more value in practice. Since the robot is superior to the sensor in WSANs, the energy cost of a few messages for the robot does not matter too much especially with such a short communication range. In contrast, the energy in a sensor is strictly limited and lower message cost has significance in extending the lifetime of the network.

Chapter 5

Conclusion

5.1 Conclusion and Future Work

In this thesis, dynamic location service for multiple events in a continuous period in WSANs was addressed. Motivated by the lack of a real-time property of existing protocols and the demands of practical applications, we developed a mesh-based dynamic location service protocol- DMesh. This protocol periodically refreshes the mesh structure which makes sure that requests from a sensor can approach its nearest robot. Depending on this static structure, robots collect vital knowledge about the network to calculate their Voronoi Diagrams. Moving to the centroid of this geometric division enables idle robots to obtain the efficient event coverage benefit. Simulation results prove that the distance to reach an event using our systematic updating is considerably diminished at a relatively small sacrifice. Additionally, the degradation and recovery attribute of our protocol make it possible for us to maintain the performance under changing location of robots. Even through our protocol increases the real-time property to handle events, intentionally moving robots to their Voronoi centroids for event preparation and the relatively high message cost for robots consume a considerable amount of energy. Therefore, its applications should be constrained to ones with a high delay-sensitive requirement and adequate energy robots.

Moreover, the choice of update cycle is related to the application. An improper value for the update cycle will have a negative impact on the performance of the whole network.

To extend our current research, there are several aspects of improvement we can think of. One thing to consider is the multiple-robot cooperation to deal with one detected event. By this means, the event dealing process will be accelerated proportionally to the number of robots. Consequently, real-time performance will be enhanced. This type of challenge is called K Nearest Neighbours (KNN) which can be found in [44] and [43]. We strongly believe it can be solved in iteratively reporting events in our mesh structure until the desired K robots is obtained in uniform event distribution. The other aspect is to study the uneven probability of events which is more realistic in real life. The deployment of robots should adjust to guarantee that the expected distance to the event is minimized.

References

[1] I. F. Akyildiz and I. H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351 – 367, 2004.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393 – 422, 2002.

[3] K.N. Amouris, S. Papavassiliou, and M. Li. A position-based multi-zone routing protocol for wide area mobile ad-hoc networks. In *Proc. 1999 IEEE 49th Vehicular Technology Conference*, volume 2, pages 1365–1369, Jul 1999.

[4] F. Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, September 1991.

[5] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *Proc. 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, DIALM '99, pages 48–55, New York, NY, USA, 1999. ACM.

[6] C.H. Caicedo-Nunez and M. Zefran. Distributed task assignment in mobile sensor networks. *Automatic Control, IEEE Transactions on*, 56(10):2485–2489, Oct 2011.

[7] A. Casteigts. The jbotsim library. *arXiv preprint arXiv:1001.1435*, 2010.

[8] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *Robotics and Automation, IEEE Transactions on*, 20(2):243–255, April 2004.

[9] R. Falcon, X. Li, A. Nayak, and I. Stojmenovic. The one-commodity traveling salesman problem with selective pickup and delivery: An ant colony approach. In *Proc. 2010 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, July 2010.

[10] G. Fletcher, X. Li, A. Nayak, and I. Stojmenovic. Randomized robot-assisted relocation of sensors for coverage repair in wireless sensor networks. In *Proc. Vehicular Technology Conference Fall (VTC 2010-Fall)*, pages 1–5, Sept 2010.

[11] M. Garetto, M. Gribaudo, C. Chiasserini, and E. Leonardi. A distributed sensor relocatlon scheme for environmental control. In *Proc. IEEE International Conference on Mobile Adhoc and Sensor Systems, 2007*, pages 1–10, Oct 2007.

[12] S. He, X Li, J. Chen, P. Cheng, Y. Sun, and D. Simplot-Ryl. Emd: Energy-efficient p2p message dissemination in delay-tolerant wireless sensor and actor networks. *Selected Areas in Communications, IEEE Journal on*, 31(9):75–84, September 2013.

[13] A. Howard, M.J. Matari, and G.S. Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Distributed Autonomous Robotic Systems 5*, pages 299–308. Springer Japan, 2002.

[14] D. B Johnson. Routing in ad hoc networks of mobile hosts. In *Proc. First Workshop on Mobile Computing Systems and Applications*, pages 158–163. IEEE, 1994.

[15] W. Kieß, H. Füßler, Jörg Widmer, and M. Mauve. Hierarchical location service for mobile ad-hoc networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 8(4):47–58, 2004.

[16] P. Kulakowski, E. Calle, and J.L. Marzo. Sensors-actuators cooperation in wsans for fire-fighting applications. In *Proc. 2010 IEEE 6th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 726–732, Oct 2010.

[17] J. Le Ny and G.J. Pappas. Sensor-based robot deployment algorithms. In *Proc. 2010 49th IEEE Conference on Decision and Control*, pages 5486–5492, Dec 2010.

[18] H.-J. Lee, Y.-H. Kim, Y.-H. Han, and C. Y. Park. Centroid-based movement assisted sensor deployment schemes in wireless sensor networks. In *Proc. Vehicular Technology Conference Fall, 2009 IEEE 70th*, pages 1–5, Sept 2009.

[19] H. Li, X. Liao, T. Huang, W. Zhu, and Y. Liu. Second-order global consensus in multiagent networks with random directional link failure. *Neural Networks and Learning Systems, IEEE Transactions on*, 26(3):565–575, March 2015.

[20] J. Li, J. Jannotti, D. SJ De Couto, D. R Karger, and R. Morris. A scalable location service for geographic ad hoc routing. In *Proc. 6th annual international conference on Mobile computing and networking*, pages 120–130. ACM, 2000.

[21] X Li, N. Santoro, and I. Stojmenovic. Localized distance-sensitive service discovery in wireless sensor and actor networks. *Computers, IEEE Transactions on*, 58(9):1275–1288, Sept 2009.

[22] X. Li, J. Yang, A. Nayak, and I. Stojmenovic. Localized geographic routing to a mobile sink with guaranteed delivery in sensor networks. *Selected Areas in Communications, IEEE Journal on*, 30(9):1719–1729, October 2012.

[23] B Liu, O. Dousse, P. Nain, and D. Towsley. Dynamic coverage of mobile sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 24(2):301–311, Feb 2013.

[24] R. Marin-Perianu, H. Scholten, P. Havinga, and P. Hartel. Energy-efficient cluster-based service discovery in wireless sensor networks. In *Proc. 2006 31st IEEE Conference on Local Computer Networks*, pages 931–938, Nov 2006.

[25] Y. Mei, C. Xian, S. Das, Y.C. Hu, and Y.-H. Lu. Replacing failed sensor nodes by mobile robots. In *Proc. 26th IEEE International Conference on Distributed Computing Systems Workshops, 2006*, pages 87–87, July 2006.

[26] T. Melodia, D. Pompili, V.C. Gungor, and I.F. Akyildiz. Communication and coordination in wireless sensor and actor networks. *Mobile Computing, IEEE Transactions on*, 6(10):1116–1129, Oct 2007.

[27] I. Mezei, M. Lukic, V. Malbasa, and I. Stojmenovic. Auctions and imesh based task assignment in wireless sensor and actuator networks. *Computer Communications*, 36(9):979–987, 2013. Reactive wireless sensor networks.

[28] A.N. Mian, R. Baldoni, and R. Beraldi. A survey of service discovery protocols in multihop mobile ad hoc networks. *Pervasive Computing, IEEE*, 8(1):66–74, Jan 2009.

[29] E. Ngai, Y. Zhou, M. R. Lyu, and J. Liu. A delay-aware reliable event reporting framework for wireless sensoractuator networks. *Ad Hoc Networks*, 8(7):694 – 707, 2010.

[30] R. Olfati-Saber, J.A. Fax, and R.M. Murray. Consensus and cooperation in networked multi-agent systems. volume 95, pages 215–233, Jan 2007.

[31] S. Parikh, V.M. Vokkarane, L. Xing, and D. Kasilingam. Node-replacement policies to maintain threshold-coverage in wireless sensor networks. In *Proc. 16th International Conference on Computer Communications and Networks, 2007*, pages 760–765, Aug 2007.

[32] C. E Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *Proc. ACM SIGCOMM Computer Communication*, volume 24, pages 234–244. ACM, 1994.

[33] C. E Perkins and E. M Royer. Ad-hoc on-demand distance vector routing. In *Proc. Second IEEE Workshop on Mobile Computing Systems and Applications, 1999*, pages 90–100. IEEE, 1999.

[34] A. Qayyum, L. Viennot, and A. Laouiti. Multipoint relaying for flooding broadcast messages in mobile wireless networks. In *Proc. the 35th Annual Hawaii International Conference on System Sciences, 2002*, pages 3866–3875, Jan 2002.

[35] W Ren and R.W. Beard. Consensus seeking in multiagent systems under dynamically changing interaction topologies. *Automatic Control, IEEE Transactions on*, 50(5):655–661, May 2005.

[36] B. Smith, A. Howard, J.-M. McNew, J. Wang, and M. Egerstedt. Multi-robot deployment and coordination with embedded graph grammars. *Autonomous Robots*, 26(1):79–98, 2009.

[37] I. Stojmenovic. Home agent based location update and destination search schemes in ad hoc wireless networks. *Computer Science, SITE, University of Ottawa, TR-99-10*, 1999.

[38] I. Stojmenovic, D. Liu, and X. Jia. A scalable quorum-based location service in ad hoc and sensor networks. *Int. J. Commun. Netw. Distrib. Syst.*, 1(1):71–94, February 2008.

[39] I. Stojmenovic, AP. Ruhil, and D. K. Lobiyal. Voronoi diagram and convex hull based geocasting and routing in wireless networks. In *Proc. Eighth IEEE International Symposium on Computers and Communication*, volume 1, pages 51–56, June 2003.

[40] S. Trigui, A. Koubaa, M. Ben Jamaa, I. Chaari, and K. Al-Shalfan. Coordination in a multi-robot surveillance application using wireless sensor networks. In *Proc. 2012 16th IEEE Mediterranean Electrotechnical Conference (MELECON)*, pages 989–992, March 2012.

[41] G. Wang, G. Cao, T. La Porta, and W. Zhang. Sensor relocation in mobile sensor networks. In *Proc. 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 2005*, volume 4, pages 2302–2312, March 2005.

[42] G. Wang, G. Cao, and T. Porta. Movement-assisted sensor deployment. *Mobile Computing, IEEE Transactions on*, 5(6):640–652, June 2006.

[43] S.-H. Wu, K.-T.and Chen C.-M. Chuang, and M.-S. Chen. Diknn: An itinerary-based knn query processing algorithm for mobile sensor networks. In *Proc. IEEE 23rd International Conference on Data Engineering, 2007*, pages 456–465, April 2007.

[44] Y. Xu, T.-Y. Fu, W.-C. Lee, and J. Winter. Processing k nearest neighbor queries in location-aware sensor networks. *Signal Processing*, 87(12):2861–2881, 2007.

[45] C.J. Sreenan Y. Zeng and G. Zheng. A real-time architecture for automated wireless sensor and actuator networks. In *Proc. Fifth International Conference on Wireless and Mobile Communications, 2009*, pages 1–6, Aug 2009.

[46] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292 – 2330, 2008.

[47] M. Younis and K. Akkaya. Strategies and techniques for node placement in wireless sensor networks: A survey. *Ad Hoc Networks*, 6(4):621 – 655, 2008.