HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

——————*——————

THESIS

SUBMITTED FOR PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

ENGINEER

IN

INFORMATION TECHNOLOGY

STRUCTURED LIGHT USING DE BRUIJN SEQUENCE FOR RANGE DATA ACQUISITION

Author:	**Nguyen Thanh Trung**
	Class AS-K56
Supervisor:	**Prof. Yamaguchi Masahiro**
	Prof. Huynh Quyet Thang

HANOI 08-2016

ACKNOWLEDGEMENTS

Though only my name appears on the cover of this thesis, a great many people have contributed to its production. I owe my gratitude to all those people who have made this thesis possible and because of whom my graduate experience has been one that I will cherish forever. My deepest gratitude is to my supervisor, Prof. Nguyen Khanh Van. I have been amazingly fortunate to have an supervisor who gave me the freedom to explore on my own, and at the same time the guidance to recover when my steps faltered. He taught me how to question thoughts and express ideas.

Mrs. Nguyen Phi Le's insightful comments and constructive criticisms at different stages of my research were thought-provoking and they helped me focus my ideas. I am also thankful to her for reading my reports, commenting on my views and helping me understand and enrich my ideas.

I am also indebted to the members of SEDIC Lab with whom I have interacted during the course of my studies. Particularly, I would like to acknowledge Mr. Nguyen Duc Trong, Mr. Truong Thao Nguyen for the many valuable discussions that helped me understand my research area better.

I am also grateful to the current lecturers at School of Information and Communication Technology, for their various forms of support during my college study.

Many friends have helped me stay sane through these difficult years – Đô, Lù, and Na and especially #Reiju Vinsmoke. Their support and care helped me overcome setbacks and stay focused on my study. Thank you for giving me hope and bringing light to my life.

Most importantly, none of this would have been possible without the love and patience of my family. My immediate family to whom this thesis is dedicated to, has been a constant source of love, concern, support and strength all these years. I would like to express my heart-felt gratitude to my family.

REQUIREMENTS FOR THE THESIS

1. Student information

Student name: Nguyen Thanh Trung

Tel:	0977366197	Email:	20102380@student.hut.edu.vn
Class:	IS-K56	Program:	HEDSPI

This thesis is performed at: Hanoi University of Science and Technology
From 05/09/2015 to 27/05/2016

2. Goals of thesis

- Study related works about Geographical routing in Wireless Sensor Network with holes
- Propose a hole bypassing routing scheme for the near hole routing problem ensuring both requirements: energy efficiency and load balancing

3. Main tasks

- Study related works about Geographical routing in Wireless Sensor Network with holes
- Propose a hole bypassing routing scheme for the near hole routing problem ensuring both requirements: energy efficiency and load balancing
- Modify the BK-WisSim toolkit to adapt more requirements of network simulation
- Implement proposed protocol and do experimental evaluations to compare it with current protocols

4. Declaration of student

I – Nguyen Thanh Trung - hereby warrants that the Work and Presentation in this thesis are performed by myself under the supervision of Assoc. Prof. Yamaguchi Masahiro.

All results presented in this thesis are truthful and are not copied from any other work.

Hanoi, 27/05/2016
Author

Vu Quoc Huy

5. Attestation of the supervisor on the fulfillment of the requirements of the thesis

Hanoi, 27/05/2016
Supervisor

Prof. Yamaguchi Masahiro

TÓM TẮT NỘI DUNG ĐỒ ÁN TỐT NGHIỆP

Một vấn đề quan trọng trong nghiên cứu xây dựng các thuật toán định tuyến cho mạng cảm biến không dây là giải quyết bài toán định tuyến trong mạng có hố. Các hố này có thể được sinh ra bởi một vài lý do, bao gồm do trong môi trường có các vật cản tự nhiên dẫn đến không thể triển khai, hoặc do các thảm họa tự nhiên gây nên. Do đó hình dạng hố rất ngẫu nhiên, có thể là các hình đa giác lõm phức tạp. Một số thuật toán đã được đưa ra để giải quyết bài toán định tuyến với hố bằng việc định nghĩa một vùng cấm quanh hố, kết hợp với một số yếu tố ngẫu nhiên qua đó giúp mạng cân bằng tải và tiêu thụ năng lượng tốt hơn. Tuy nhiên, đối với bài toán định tuyến gần hố, tức là bài toán tìm đường định tuyến có nguồn hoặc đích nằm trong các vùng cấm, các nghiên cứu này đều chưa đưa ra được một thuật toán định tuyến hiệu quả, mà chủ yếu dựa vào thuật toán định tuyến tham lam. Trong đồ án này, chúng tôi đề xuất một thuật toán định tuyến giải quyết bài toán định tuyến gần hố, sử dụng sơ đồ Voronoi để xây dựng lên các đường định tuyến "thoát hố". Kết quả thực nghiệm cho thấy giải pháp đề xuất có kết quả tốt hơn các thuật toán định tuyến tham lam và thuật toán BOUNDHOLE trên các phương diện: năng lượng, cân bằng tải, độ dài đường định tuyến.

ABSTRACT OF THESIS

Geographic routing is well suited for large-scale wireless sensor networks (WSNs) since it is nearly stateless. It can achieve a near-optimal routing path in the networks without of holes because of its simplicity and scalability. With the occurrence of holes, however, geographic routing faces the problems of hole diffusion and routing path enlargement. The hole shape could be very complex with many cave regions. Several recent proposals attempt to fix these issues by deploying a special, forbidding area around the hole, which helps to improve the congestion on the hole boundary but still are deficient since they use the perimeter routing scheme to bypass the hole within the forbidden area. In this thesis, we introduce a novel approach which is the first to target and solve the near hole routing problem, while ensuring both two requirements: energy consumption and load balancing. Our simulation experiments show that our scheme strongly outperforms the existing schemes in several performance factors, including route stretch, efficient use of energy and load balancing.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

1.1.1 Wireless Sensor Network Overview

Recent advances in micro-electro-mechanical systems, embedded processors, and wireless communications have led to the emergence of Wireless sensor networks (WSNs) [4], which consist of a large number of sensing devices each capable of sensing, processing and transmitting environmental information. Applications of wireless sensor network include battlefield surveillance [6], biological detection, smart spaces [6], industrial diagnostics, and so on. Another important application of wireless sensor network that cannot be left out is environmental monitoring, analyzing and forecasting disasters such as Forest fire detection, Flood detection, ... We envision that, in near future, wireless sensor networks will be more popular and be an integral part of our lives.

One of the most important constraints on sensor nodes is the low power consumption requirement. The wireless sensor node can only be equipped with a limited, generally irreplaceable, power source. Therefore, while traditional networks aim to achieve high quality of service (QoS), sensor network protocols must focus primarily on power conservation. In an ad hoc sensor network, each node plays the dual role of data originator and data router. The dis-functioning of few nodes can cause significant topological changes and might require re-routing of packets. Hence, power conservation and power management take on additional importance.

Another essential property of wireless sensor networks is its scalability and distribution. The number of sensor nodes deployed in studying a phenomenon may be in the order of hundreds or thousands. Depending on the application, the number may reach an extreme value of millions, for instance, the forest fire detection system may contain thousands sensor nodes.

In the wireless sensor network, communicating nodes are linked by a wireless medium. Thus, a sensor node can only communicate with sensor nodes which lie within its communication range. In case of communicating two sensor nodes which do not lie within each other's communication range, the packet have to be transmitted via intermediate nodes. The routing path of packets is calculated depending on the routing protocol that network uses.

1.1.2 Geographical routing in Wireless Sensor Network

Routing protocols in wireless sensor network recently has attracted much concerns from researchers. The designing of routing protocols in wireless sensor network is challenging due to its mentioned properties. For example, in the traditional network such as Ethernet, a node can analysis whole network to find out the optimal routing path, however, it is impossible to do this task in wireless sensor network. The requirements of routing protocols in wireless sensor network not only includes low latency, fault tolerance, but also has to guarantee following factors:

- *Computational complexity*: sensor nodes have very limited and low hardware power so the complexity of the algorithm has to be as simple as possible. For example, the wide use sensor node, Mica 2 Mote [1], has only 128K bytes of program flash memory and uses a 8-bit micro-controller.
- *Energy consumption*: because of limited and irreplaceable power source, the routing protocol has to be energy efficiency.
- *Load balancing*: the load balance has a big impact on the energy consumption of sensor nodes. Network congestion causes some sensor nodes carry more data than the others and quickly die. Consequently, it significantly changes the connectivity and topology of the network. Thus, load balancing plays an important role in routing in wireless sensor networks.

A common approach is using geographical location information of sensor nodes to design the routing protocols. It is well suited for large-scale wireless sensor networks because it is nearly stateless. The class of these protocols is named *geographical routing protocols*. This approach assumes that exact location of sensor nodes is known beforehand. In other words, the routing path of the packet is conducted from the geographical location information of sensor nodes. Notice that, in geographical routing, each sensor node only needs to know its neighbors' location information (a 1-hop neighbor of a sensor node S is a node which is inside the communication range of S). When a sensor node receives a packet, it decides to forward this packet to one of its neighbors based on current location information of its neighbors.

A classical algorithm in this type of protocols is the greedy algorithm [10]. The greedy algorithm is described as follows: the packet is forwarded to a neighbor which is closer to the destination node than current node and closest to the destination node among all neighbor nodes. Greedy forwarding's great advantage is its reliance only on knowledge of the forwarding node's immediate neighbors. The state required is negligible, and dependent on the density of nodes in the wireless network, not the total number of destinations in the network. However, the biggest problem of greedy algorithm is the occurrence of the local minimum phenomenon, i.e. there is no satisfying neighbor to forward the packet anymore. Figure 1.1 demonstrates this phenomenon. This problems is usually caused by the existence of *routing holes* in the network topology.

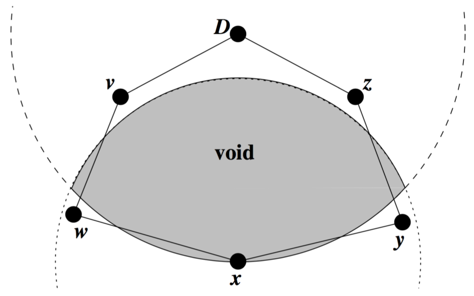


Figure 1.1: Node x's void with respect to destination D.

A *routing hole* in a wireless sensor network is a region where either there are no sensor nodes or the available nodes cannot participate in the activity of the network, i.e. they cannot communicate with the rest of the network. These holes can be formed either due to voids in sensor deployment or because of failure of sensor nodes due to various reasons such as malfunctioning, battery depletion or an external event such as fire, flood or structure collapse physically destroying the nodes. The local minimum phenomenon likely occurs when a routing hole is encountered. In the next chapter, we will discuss several previous works attempting to detect routing holes and to route the packet around them in greedy forwarding.

1.2 Thesis statement

1.2.1 Thesis statement

Several solutions has been proposed to deal with the routing problem in wireless sensor network with holes [11, 9, 7, 12, 16, 14]. The most well-known solution is the Greedy Perimeter Stateless Routing (*GPSR*) protocol, which was proposed by B.Karp in 2000 [11]. This algorithm computes a planar graph for the network in which its vertices are sensor nodes and its edges are edges connecting sensor nodes and its neighbors. The packet is then forwarded based on edges of the graph. In 2005, Qing Fang and her colleagues proposed a novel algorithm (the BOUNDHOLE algorithm) to detect the boundary of the routing hole [9]. In addition, these authors also proposed a routing protocol based on the determined hole boundary: when the packet reaches a *stuck node*[1] S, it is forwarded along the boundary of the hole until it reaches an intermediate sensor node that is closer to the destination node than S.

Other solutions based on the BOUNDHOLE algorithm has also been proposed. The common idea is that these protocols construct a forbidden area covering the hole and broadcast the information of this area to the network. The source node then uses these information to calculate the routing path. The forbidden area is usually a simpler shape such as circle [7], ellipse [16], parallelogram [12], or octagon [14]. In [14], the authors also proposed a non-trivial approach that is to use some random factors to ensure mentioned requirements of routing in wireless sensor network such as load balancing, energy efficiency and route stretch.

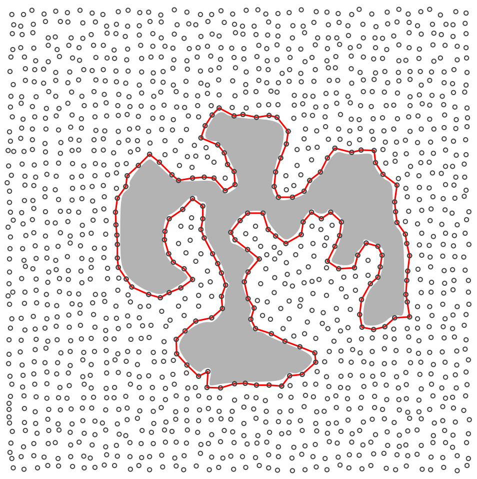


Figure 1.2: An example of hole polygon.
In this figure, the gray region is the hole and the red line is the hole boundary.

However, the routing hole in wireless sensor network can be a random, complex shape and thus, may have a lot of cave regions. Figure 1.2 shows an example of hole

[1]A stuck node is a node where a packet may get stuck and is more likely to occur the local minimum phenomenon. We will discuss further more about stuck node definition in section 2.1.

shape. *Near hole routing* problem is defined as a problem to find the routing path in which the source node or the destination node, or both of them lie nearly the hole, in the boundary of the hole or inside the cave areas. The difficulty of this problem probably is to construct a path for packet to route out the hole when the source node is inside a cave or to construct a path to forward packet from outside to a destination which is inside a cave. Most of the above algorithms have been ignored the near hole routing problem. In such a case, they all use the GPSR algorithm to compute the routing path. This generally degrades the efficiency of these protocols in case there requires a lot of near hole routings. This leads to the necessity of a good *near hole routing protocol* to fulfill requirements of routing in wireless sensor networks.

1.2.2 Objectives

Based on the described problem in subsection 1.2.1, the objectives of my thesis are:

1. *Study related works about geographical routing in wireless sensor network with holes.* We need to make a survey about previous works: their advantages and disadvantages. Generally, a routing protocol consists of two phases: the initial phase when the network learns about the hole shape and computes parameters of the protocol and the data forwarding phase when the network starts sending data packets. Most of previous works use different shapes to approximate the hole polygon in the initial phase. Thus, it is supposed to analyze these shapes to find out the most appropriate shape for the hole approximation since the approximate hole impacts the load balancing, routing stretch and hence, network's energy consumption.

2. *Propose a solution for the near hole routing problem.* Currently, as our knowledge, there are no protocols that can deal with the near hole routing problem except GPSR and BOUNDHOLE. We will propose a protocol which targets to achieve both two main requirements of routing in wireless sensor network with holes, that is load balancing and energy consumption. This is also the main objective of my thesis.

3. *Modify the BK-WisSim toolkit to adapt more requirements of network simulation. BK-WisSim* is a visual network simulation tool developed by SEDIC Lab [3]. We continue developing this tool to support more requirements of network simulation, such as improving the current Energy Model of *ns-2* and developing new features for *BK-WisSim*.

4. *Implement proposed protocol and do experimental evaluations to compare it with current protocols.* The last mission is implementing proposed protocol and embedding it into the core of *BK-WisSim*. Then the proposed protocol will be compared

with current exist protocols (GPSR, BOUNDHOLE) to evaluate its performance and proof that it totally satisfies the requirements.

1.2.3 Results and Contributions

Studied related works about geographical routing in wireless sensor network with holes.

We have provided a survey about previous works. In this survey, firstly, we described the BOUNDHOLE algorithm and explained the reason why we need to use this algorithm to detect the routing hole instead of other algorithms; secondly, we described an overview about each protocol and pointed out their pros and cons. We have also come up a conclusion that octagon is the most appropriate shape to use to approximate the hole based on theoretical analyses given by its authors.

Proposed the near hole routing protocol.

This algorithm is my primary research, conducted from studies and research to find a solution for near hole routing problem. This algorithm solves the difficulty of the near hole routing problem by constructing the Voronoi diagram [5] of the hole polygon and then creating a graph which is a *"skeleton"* of this polygon based on the Voronoi diagram. This graph is used to make the routing path for the packet to get out or get in the cave. It also inherits the previous result, that is using octagon to approximate the hole in initial phase and using random factors in routing phase. We also provided some theoretical analyses to prove the validity of the protocol, a discussion about the impact of protocol's parameters. This proposed algorithm not only inherits the advantages of previous algorithm but also completely solves the near hole routing problem.

Updated BK-WisSim toolkit.

We have improved the Energy Model of *ns-2* to support more experimental scenarios and to extract more information about the simulation process. We also developed some new feature for *BK-WisSim* client including: supporting graph algorithms (Voronoi diagram and Dijsktra algorithm) in WisSim Editor, simulating the simulation process via animation in WisSim Visualizer and more.

Evaluation of proposed protocol.

We proposed some evaluation metrics and scenarios for experiments. These metrics are: energy consumption, deviation of energy consumption, load balancing, routing path length and network lifetime. The experiments were performed to compare the proposed protocol with GPSR and BOUNDHOLE algorithm. The results shown that our proposed protocol is better than both the other algorithms in all evaluation metrics. We also performed some specific experiments for our proposed protocol to analyze the impact of protocol's parameters on energy consumption and routing stretch.

The remainder of this thesis is organized as follows. In chapter 2, we discuss about the BOUNDHOLE algorithm and related works. The details of proposed protocol is presented in chapter 3. In this chapter, we firstly describe the protocol details and then provide some theoretical analysis of proposed protocol. In chapter 4, we give an overview about *ns-2* and *BK-WisSim* toolkit. We also introduce new features of *BK-WisSim* in this chapter. The source code and detailed implementation of proposed protocol is provided in chapter 5. In chapter 6, we present the simulation result to evaluate the performance of the protocol. Finally, we conclude the thesis in chapter 7.

Chapter 2

Routing protocols in Wireless Sensor Network with Holes

We noted that the existence of routing holes in network's topology usually causes the local minimum phenomenon. Thus, a routing protocol has only two options in this scenario, either to drop the packet or to solve the routing hole problem, by using a certain mechanism. There are different methods of dealing with this problem. These protocols have helped us to further define some aspects of our design for the near hole routing protocol. In this chapter, we present these works, and point out their advantages that have been derived in our proposed protocol.

To approximate the hole polygon, firstly, we have to detect the hole boundary, or, those nodes that lie on the boundary of the hole. This problem is known as the hole boundary detection problem. The detection of hole boundary is important for improving wireless sensor networks operations.

There is an intuitive way to detect a routing hole. Given a network presented as a Cartesian coordinate system in which each sensor node is a point in the plane, we construct Delaunay triangulation of this network, then with a pre-defined threshold η, we remove all edges whose length is greater than η. Remaining edges create some polygons which have more than 3 edges. These polygons may be routing holes. However, the problem of this algorithm is that it is not a distributed protocol, since it requires a central node that has location information of all sensor nodes in the network. Hence, applying this algorithm in case of wireless sensor network may not be effective. Some distributed version of Delaunay triangulation have been proposed, but according to our current knowledge, there is still no truly distributed algorithm.

In 2004, Q.Fang proposed a novel algorithm called *BOUNDHOLE* to detect the hole boundary in a distributed manner. Many routing protocols have adopted the *BOUNDHOLE* algorithm for their hole boundary detection phase. In this chapter, before going into detail of current proposed routing protocols, we will give an overview of *BOUNDHOLE* algorithm.

2.1 The TENT rule and BOUNDHOLE algorithm

2.1.1 The TENT rule

The *TENT rule* is used locally by each sensor node, that is, a node p needs only its 1-hop neighbors and itself location information to determine whether it lies on the boundary of a routing hole. We exemplify the *TENT rule* using representation in figure 2.1. At each node p, its neighbors are sorted angularly. For every pair of neighbors u and v, where $\overrightarrow{pu}$ is left of $\overrightarrow{pv}$, we consider the perpendicular bisectors of each link (labeled $l1$ and $l2$ in figure 2.1). If the intersection of the bisectors falls outside of the range of p, then p has no neighbors closer to the region bounded by the communication range and the bisectors (i.e., the black region in figure 2.1). p is called *"stuck node"* and $\angle upv$ is called *"stuck angle"*. Note that this method locates regions, rather than nodes, where no neighbors are closer. Every node must evaluate its 1-hop neighbors using the *TENT rule*, and only at nodes that the *TENT rule* fails may packets get stuck. If a node p has at least 1 stuck angle, p has to be on the hole boundary.

This rule also elicits that if angle upv is less than 120°, then there is no stuck angle of p corresponding with pair (u, v). Thus, a sensor node has at most 3 stuck angles.

Upon detecting a failure, stuck nodes are responsible for initiating the *BOUNDHOLE* algorithm.

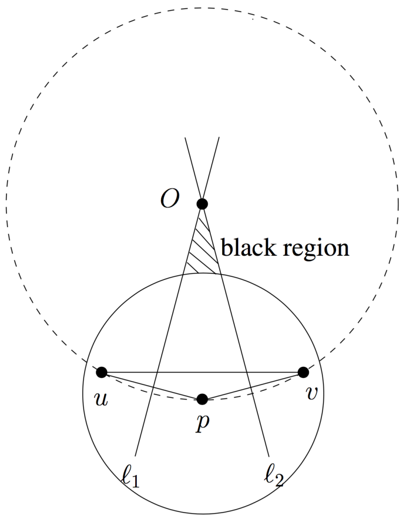


Figure 2.1: TENT rule: Node p has no neighbors closer to the back regions as determined by the intersection of bisectors of adjacent links.

2.1.2 The BOUNDHOLE algorithm

After being detected as a *stuck node*, the stuck node p can initiate the distributed algorithm, *BOUNDHOLE* that uses the right-hand rule to connect a node with its neighbors, to form a closed region. That closed region is a routing hole. The basic idea is shown in figure 2.2. The algorithm starts from node p and sweeps over the stuck direction by sending a packet to node t_1 in a counter-clockwise direction. Node t_1 then passes the packet to node t_2. The above process is repeated until the packet moves around the hole boundary and returns to p. Now that the boundary of a hole has been identified and stored locally.

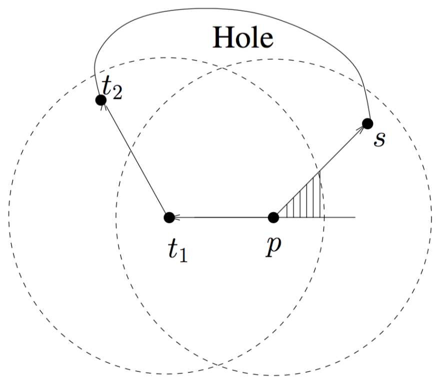


Figure 2.2: BOUNDHOLE algorithm: Greedy sweeping at t_1

A stuck node initiates a messenger packet in each direction this node is stuck in. Thus, there are a lot of packets the originated from all stuck nodes in the hole boundary. These packets are obviously redundant since we need only 1 packet to detect the hole boundary. To reduce networking overhead and possible packet collisions, especially when large holes exist in the network, the ID of the originator is recorded in each messenger packet and the messenger packet originated by the stuck node with the lowest ID is guaranteed to be overlaid at each hop. That is, all packets which have the originator ID greater than the lowest ID will be dropped.

Due to the distributed property of the *TENT rule* and the *BOUNDHOLE* algorithm, this protocol ensures the scalability even when the network topology is extended.

2.2 Hole bypassing protocols

As stated previously, all the proposed routing protocols using the *BOUNDHOLE* algorithm in its hole boundary detection phase construct a forbidden area which covers the hole polygon. The purpose of constructing these forbidden areas is to simplify the hole polygon, thus it reduces the memory and network overhead for storing and broadcasting the hole information to the network. The forbidden area could be represented as a simpler shape such as circle [7], ellipse [16], hexagon [8], parallelogram [12] or octagon [14]. We call these simpler shapes are the approximate hole. These protocols are all separated to 2 main phases. The first one is the network initialization phase, and the second one is data forwarding phase. However, there are some fundamental differences among these protocols.

In the network initialization phase, all mentioned protocols try to detect the hole boundary using the *BOUNDHOLE* algorithm. After finishing detecting the hole boundary, the hole boundary message originator calculates the approximate holes. After that, some protocols (which are [16], [8], [12], [14]) have an extra phase called broadcast hole information phase. The broadcast data could be the information of the approximate hole, and the broadcast area is controlled by a system parameter. In the contrary, some protocols (which are [7]) forward the hole information to all boundary sensor nodes, and let each boundary sensor node calculate the approximate hole itself.

In the data forwarding phase, although each protocol uses a different shape of forbidden area, the routing path construction algorithm still shares some common ideas. They all use the concept of *virtual anchor point*. These *virtual anchor points* create a hole-bypassing routing path for the data packet, and its determination depends on each routing protocol. There are two common approaches to determine these *virtual anchor points* which are: intersection points of tangent lines in case of [7, 16] or vertices of approximate polygon in case of [12, 14, 8].

Due to the limitation of the thesis, we only give thorough studies on [7, 14, 12] protocols.

2.2.1 The Efficient Hole Detour Scheme

A representative for the group using the tangent line to construct the routing path is the Efficient Hole Detour Scheme (*EHDS* protocol). In this proposal, the authors used a circumscribed circle to approximate the hole polygon.

In the network initialization phase, the protocol uses the *BOUNDHOLE* algorithm to detect the hole boundary. Then the creator of the hole boundary detection message calculates the circumscribed circle of the hole polygon using the algorithm described in algorithm 1. This circumscribed circle information would be sent to all hole boundary sensor nodes. Figure 2.3 represents the idea of this scheme.

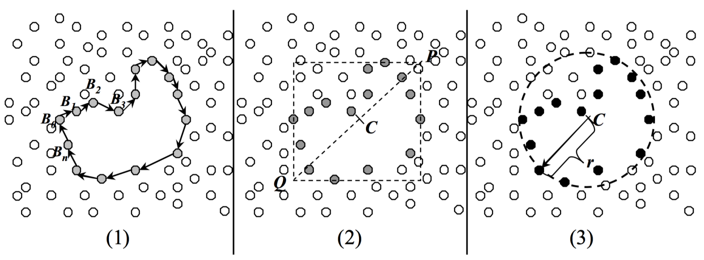


Figure 2.3: The Efficient Hole Detour Scheme (cited from [7]). Illustration for the hole approximation phase.

In the data forwarding phase, the packets are firstly forwarded by the greedy routing until it reaches a hole boundary sensor node. This node is called an intermediate node. This intermediate node would send the packet back to the source by greedy routing

Algorithm 1: Circumscribed circle constructing algorithm

```
Input  : $P = \{P_1, ..., P_n\}$: the hole polygon
Output : $S(C, r)$: the circumscribed circle of $P$
$X \leftarrow \{P_{i_x} \mid i = \overline{1, n}\}$;
$Y \leftarrow \{P_{i_y} \mid i = \overline{1, n}\}$;
$x_{min} \leftarrow min(X)$ ;
$x_{max} \leftarrow max(X)$ ;
$y_{min} \leftarrow min(Y)$ ;
$y_{max} \leftarrow max(Y)$ ;
$C \leftarrow (\frac{x_{min}+x_{max}}{2}, \frac{y_{min}+y_{max}}{2})$;
$r \leftarrow \frac{\sqrt{(x_{min}-x_{max})^2+(y_{min}-y_{max})^2}}{2}$;
return $S(C, r)$;
```

with extra information including the location of center point and the radius of the hole polygon's circumscribed circle. After finishing receiving back the packet, the source node resends the packet with new bypass routing path to avoid the circle. Figure 2.4 illustrates this routing protocol.

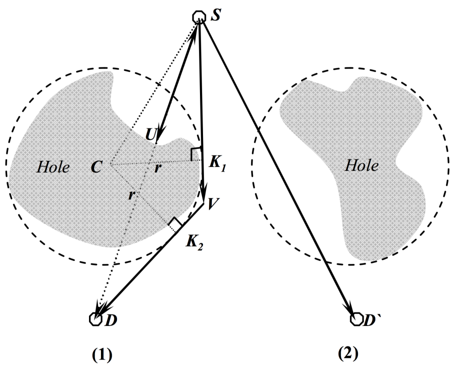


Figure 2.4: The Efficient Hole Detour Scheme (cited from [7]). Illustration for the data forwarding phase. The forbidden area is the circumscribed circle of the hole polygon.

In this figure, S, D are the source node and the destination node, respectively. Firstly, S greedily forwards the packet until it reaches a node U which is a hole boundary sensor. U then sends back to S the forbidden area information. Finally, S re-forwards the packet using new routing path, that is $\overrightarrow{SVD}$. The packet is forwarded along this path. V is called the virtual anchor point in this case.

It is obvious that in the *EHDS* protocol, the packet has to be forward twice: in the first time, it is forwarded by the greedy routing, in the second time, it is forwarded by the hole bypassing routing. Consequently, this increases the packet transmission time, as well as the energy consumption of nodes participating in the routing.

One more problem of *EHDS* protocol is that, this protocol leads to the routing path enlargement problem in case of near hole routing which is demonstrated in figure 2.5.

In this figure, the source node S is close to the forbidden area. Since this protocol uses the tangent line as the routing path, the closer to the forbidden area the source node and the destination node are, the more longer the routing path is.

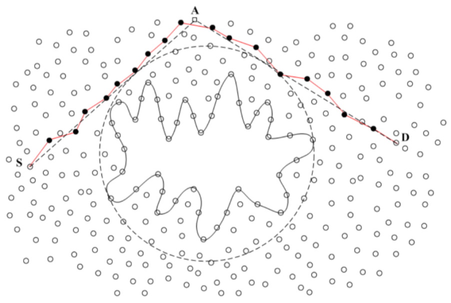


Figure 2.5: The routing path is enlarged in case of near hole routing.

More importantly, the author ignored the case where the source node or the destination node or both of them are inside the forbidden area. In such a case, there is no hole bypassing routing paths.

2.2.2 The Energy Efficient and Load Balanced Distributed Routing Scheme

The Energy Efficient and Load Balanced Distributed Routing Scheme (ELBAR protocol) is the one that simultaneously considers both energy efficiency and load balancing requirements. In this protocol, the authors proposed a novel algorithm to approximate the hole polygon. Instead of using a geometric shape, the authors use a square grid to approximate the hole [13]. Using a square grid ensures the convex/concave properties of the original hole polygon, and the error between the approximate hole and original hole is controlled by a system parameter which is the length of edges of grid's square units. The approximate hole is a polygon whose edges are the edges of the grid and vertices are the vertices of the grid. Its properties are:

- Covering: the approximate hole covers completely the hole polygon.
- Controllable approximate error: the upper bound of the total vertices of the approximate hole is controlled by a system parameter n. The larger the value of n is, the less the approximate error is.

Figure 2.6 illustrates the approximate hole using a square grid.

This scheme propose additional modifications to the packet forwarding function and routing functions. With respect to specific routing hole H all the nodes are divided into three regions based on two predefined threshold values α_{min} and α_{max} as shown in figure 2.7. A particular node is in:

- region 1 if the hole view angle of the node is either greater than or equal to α_{max},
- region 2 (which is referred to as an escape region in what follows) if the hole view angle of the node is in interval $(\alpha_{min}, \alpha_{max})$,

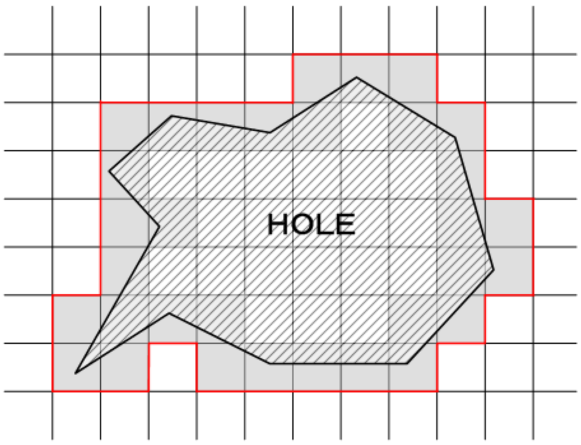


Figure 2.6: An example of approximate polygon.
Grey area represents the hole. Bold black line represents the boundary of the approximate polygon.

- region 3 if the hole view angle of the node is either less than or equal to α_{min}.

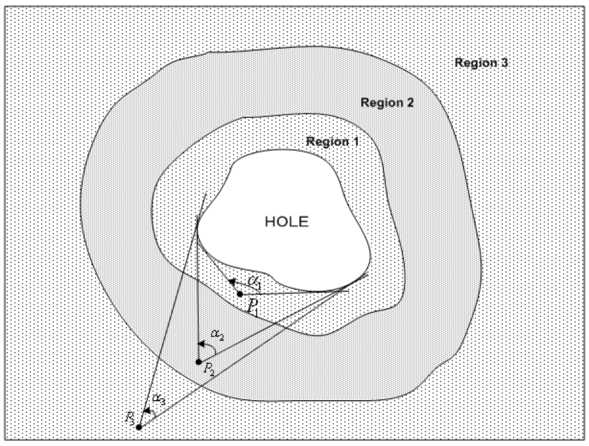


Figure 2.7: Regions with respect to a specific routing hole.

A parallelogram is used as the forbidden area, and these parallelograms are different from each sensor node. The construction of the parallelogram is illustrated in figure 2.8.

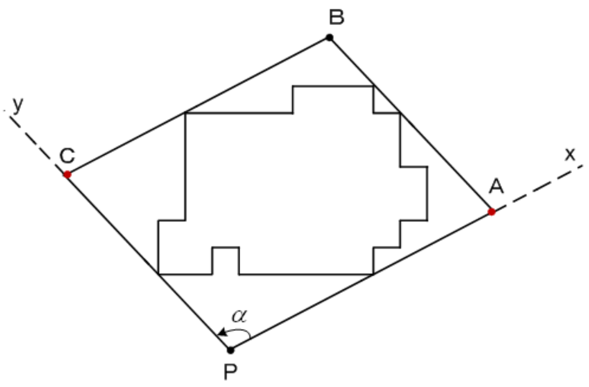


Figure 2.8: The parallelogram of the sensor node P

The routing protocol is described as follows. Two fields are proposed to be carried by a packet: the first field is the *forwarding mode* that can take either *default* or *escape* value, while the second field *anchor location* is to store the anchor point in the *escape* mode. Let D be the destination of packet p. The forwarding mode p_m is initialized to *default* and anchor location p_a takes D when packets are first generated. When node P is in region 3, it simply keeps the *default* mode. When P is in the *escape* region and

- if D lies either outside the hole view angle of P or inside the hole covering parallelogram at P, then the packet is forwarded by the rule of the GPSR protocol [11].
- otherwise, a hole escape probability function is executed to adjust the routing mode for the packet. In the escape mode, the route is "bended" around P's hole view parallelogram.

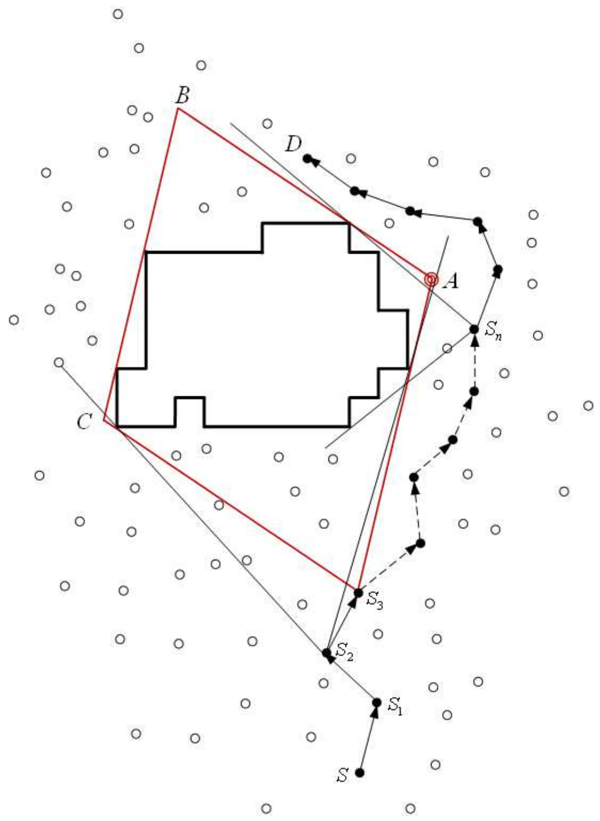


Figure 2.9: ELBAR packet forwarding example.

Figure 2.9 illustrates how a data packet would be routed from source node S in region 3 to destination node D. When in region 3 (far from the hole then has no information about it) the packet is forwarded in the *default* mode until it reaches a node in region 2, namely node S_2 in the figure. Note that reaching region 2 does not mean automatically switching to *escape* mode but to mean having a chance to switch to this mode. In this example, the adjust routing mode method at S_2 returns in *default* mode, thus the packet is forwarded to the neighbor of S_2 which is nearest to destination

D. However, at the next hop, i.e S_3, the adjust routing mode method returns in *escape* mode thus S_3 updates necessary fields and sends the data packet towards the virtual anchor A. Before arriving to the node which is nearest to A, the packet however reaches intermediate node S_n that can "see" D (without the view blocking of the hole). Now S_n forwards the packet in the default mode to destination D.

In the *ELBAR* protocol, we can think region 1 as the near hole region. For the source node belonging to this region, the *GPSR* routing protocol is used.

2.2.3 The Load Balanced Routing with Constant Stretch Scheme

The Load Balanced Routing with Constant Stretch Scheme (the *OCTAGON* protocol) takes a further step to target and solve two problems of hole diffusion and path enlargement. Its theoretical analysis proves the constant stretch property and its evaluation results show that this scheme strongly outperforms the existing schemes in several performance factors, including route stretch, efficient use of energy and load balancing.

This protocol also consists of 2 main phases: the initial network setup phase and the data forwarding phase. The initial network setup is designed to produce the core polygon, a compact approximation of the hole, which has 8 vertices and covers the hole boundary. The initial network setup process consists of three sub-phases. First, the nodes on the hole boundary are identified using the *BOUNDHOLE* algorithm [9]. Then, the core polygon is constructed. In the final phase, the core polygon information is disseminated to the nodes surrounding the hole. The dissemination area is not all of the network but only a restricted area in order to save the energy consumption. Once the initial network setup is finished, the nodes nearby the hole know about the location of the core polygon while the nodes far from the hole do not.

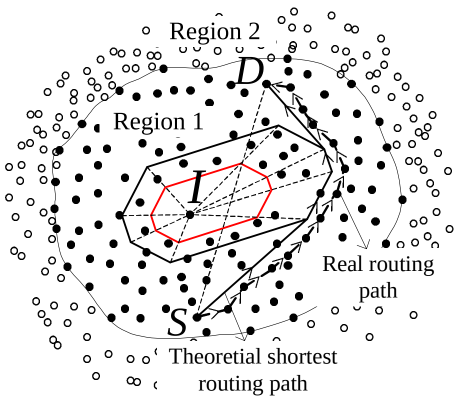
If the packet is initialized at a source node out of a dissemination area of a hole(not knowing about the hole and the core polygon), it is simply greedily forwarded towards the destination node until reaching an intermediate node inside this area. Once in this area the source node (or intermediate node) can construct a packet-specific polygon, seen as a forbidding area for this packet, which is the image of the core polygon through a homothetic transformation with the scale factor based on the source-destination distance. The packet is then greedily forwarded to the virtual anchor points which are the vertices of this packet-specific polygon.

The core polygon satisfies the following conditions:

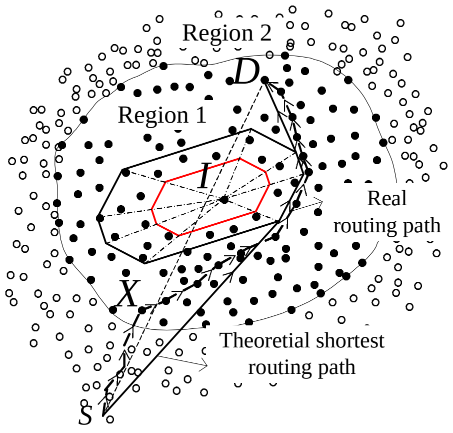
- It completely covers (the boundary of) the hole.
- It has at most 8 vertices and each edge of its contains at least one vertex of the hole.
- All angles at the vertices that are not the vertices of the hole, equal to $\frac{3\pi}{4}$.

After the core polygon information broadcast phase, the network is divided into 2 regions: region 1 contains nodes that receive the hole core polygon broadcast message, and region 2 contains the other nodes as shown in figure 2.10.

The main idea of the data forwarding phase is to find the shortest route that has to go around a larger polygon A that is the image of the core polygon through a homothetic transformation using a center I chosen randomly inside the core polygon and the scale factor $\xi > 1$ computed based on the distance between the source and the destination. This polygon A is being used as a dynamic forbidding area which is different per each packet and source node because of the randomization of center I. As a result, the routing path also changes dynamically for each packet and thus balance the energy consumption of the network. While, the scale factor ξ is designed so that the stretch of the routing path is bounded by a constant. Figure 2.10 shows examples of the routing scheme in both cases when the source node belongs to the region 1 and to the region 2.



(a) Source node belongs to region 1



(b) Source node belongs to region 2

Figure 2.10: Examples of the *OCTAGON* routing scheme

Since our near hole routing protocol adopted some advantages from this protocol, we give here more details about this protocol's theoretical analysis. Firstly, the authors proved that the upper bound of the stretch does not exceed $\frac{1}{sin(3\pi/8)} + \delta$ for every source and destination pair (S, D) staying outside the core polygon. More notably, the authors explained the reason why they choose a octagon to approximate the forbidden area. The reason is if P is a *n-gon* with equal angles such that P covers the hole and each edge of P contains at least one vertex of the hole, then the E-stretch of P to the hole is bounded by $\frac{1}{sin(\frac{(n-2)\pi}{n})}$, a decrease function which converges to 1. The value of this function decreases strongly when n increases from 3 to 8 and does not decrease much after that.

Chapter 3

Near hole routing protocol

In this chapter, we describe our near hole routing protocol in details.

3.1 Proposed Scheme

3.1.1 Assumptions and definitions

To ease the comprehension, we present a short overview of terms and definitions related to routing holes. Following the approach in [7, 12, 14], we consider the same assumptions as follows.

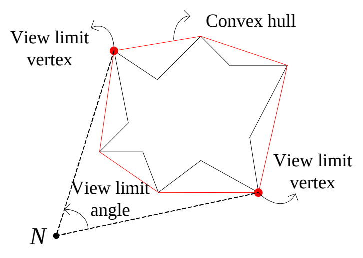
- Each node knows the position of itself and its 1-hop neighbors by using GPS or other positioning services.
- The location of a destination is known by a source.
- All the sensor nodes have the same radio hardware. Let R_c denote the transmission range of a node.
- The energy consumption of the computation performed for one packet at a node is negligible compared to the energy consumption of the transmission of one packet.

We start with the definitions of the notations and terms being used throughout this paper. Below, let P denote a polygon which is determined by a set of vertices $P_1, P_2, ..., P_n$.

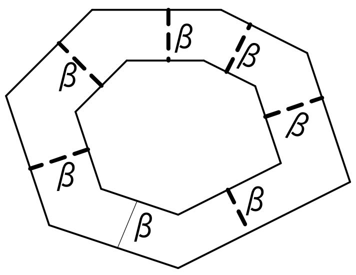
Definition 1 (Convex hull of polygon)**.** *The convex hull of polygon P is the smallest polygon whose vertices are vertices of P and completely covers P.*

The convex hull of polygon P is denoted as C_P.

Definition 2 (View-limit vertex)**.** *P_i is called a view-limit vertex from a point N outside P to P if and only if the line through N and P_i does not intersect P (see Figure 3.1a).*



(a) View-limit vertices of N



(b) β-covering polygon

Figure 3.1: Illustrations of definition 2, 3

Clearly, there are only two such view-limit vertices from a given node N.

Definition 3 (β-covering)**.** *Assume that P is a polygon and β is a positive number. Then, β-covering polygon of P is a polygon covering P and having each edge parallels to an edge of P while the distances between each parallel pair is the same β (see Figure 3.1b).*

Definition 4 (Point-to-Polygon Distance)**.** *Assume that P is a convex polygon, N is a point staying outside of P . $d > 0$ is defined as the distance from N to P if and only if N stays on the boundary of d-covering polygon of P.*

The distance from point N to polygon P is denoted as $d(P, N)$.

Definition 5 (Inner polygon)**.** *An inner polygon of polygon P is a set of sequentially ordered nodes denoted by $IP_{P_{jk}} = \{P_j, P_{j+1}, ..., P_k\}$, where $IP_{P_{jk}} \in P$ such that P_j and P_k are the extreme points of convex hull C_P. In particular, P_jP_k is called the **gateway** of $IP_{P_{jk}}$.*

Figure 3.2 shows an example with four inner polygons and their gateways. There is a polygon P (black edges). Polygon P has four inner polygons which are IP_1, IP_2, IP_3, IP_4. The gateway of IP_1, IP_2, IP_3, IP_4 are $IP_{1_1}IP_{1_2}$, $IP_{2_1}IP_{2_2}$, $IP_{3_1}IP_{3_2}$, $IP_{4_1}IP_{4_2}$ (red edges), respectively.

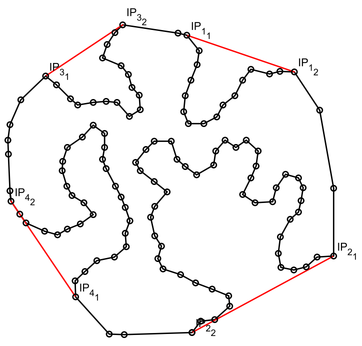


Figure 3.2: An example of inner polygons.

3.1.2 Overview

Our proposed scheme consists of two parts, namely the *initial network setup* and the *data forwarding*. The *initial network setup* is designed to construct a graph for each inner polygon of the hole polygon. It consists of three phases: *Hole Boundary Detection*, *Hole Information Dissemination* and *Internal Graph Construction*. Firstly, in the *Hole Boundary Detection* phase, the hole boundary is identified using the *BOUNDHOLE* algorithm [9]. During this phase, a leader node who is responsible for initiating the next phase: *Hole Information Dissemination* is selected based on node IDs. This leader node also determines the convex hull of the hole polygon. Then, in the *Hole Information Dissemination*, the hole polygon information is disseminated to the nodes surrounding the hole. The dissemination area is controlled by a system parameter δ. The dissemination area is called the **near hole region**. Finally, in the *Internal Graph Construction*, all nodes staying inside any inner polygons build an internal graph of its inner polygon. The internal graph is built based on the Voronoi diagram [5] of the inner polygon. It represents the internal structure of a inner polygon and is used for routing a packet optimally inside the convex hull of a hole.

Our data forwarding algorithm provides two routing mode, namely, *near hole* mode and *default* mode. The near hole routing consists of three cases:

- Both source node and destination node belong to the *near hole region*.
- Only source node belongs to the *near hole region*.
- Only destination node belongs to the *near hole region*.

In the first case, the source node s uses the *near hole* mode to guide a packet out of the convex hull. If s stays inside any inner polygons, it first finds an intermediate node in the *gateway* and deliveries the packet to it. We call this intermediate node the source's **virtual anchor point** (VA-point). The detail algorithm to compute the VA-point is described in the next subsections. Otherwise, s is the source's VA-point itself. In the source's VA-point, a *dynamic forbidden area*, seen as a packet-specific forbidden area, is constructed. It is the image of the octagon covering the hole polygon through a homothetic transformation with the scale factor based on the distance from the source's VA-point to the convex hull. It also checks that if the destination node d is inside any inner polygons. If the answer is "yes", it constructs the internal graph of the inner polygon containing d and determines the destination's VA-point. The packet is then greedily forwarded to the VA-points which are the vertices of this packet-specific polygon. When the packet reaches the last VA-points, it is keeps greedily forwarding the packet to the destination's VA-point. Finally, the destination's VA-point forwards the packet to the destination based on the constructed internal graph. Figure 3.3 shows an example of this case. In this figure, I_s is the VA-point of the source node s, I_1, I_2, I_3

is the VA-points which are the vertices of scale octagon, and I_d is the VA-point of the destination node d. The routing path is $\overrightarrow{sI_sI_1I_2I_3I_dd}$.

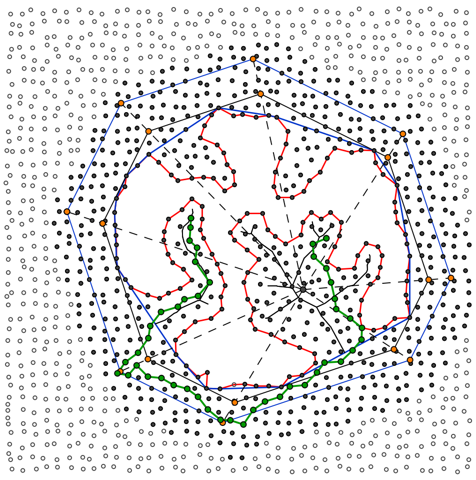


Figure 3.3: An example of proposed routing path.

In the second case, the packet is forward along the routing path constructed same as the first case, until it reaches an intermediate node that does not belong to the *near hole region*. Then the packet is greedily forwarded to destination node. In the third case, first, the packet is forward greedily to the destination, until it reaches an intermediate node belonging to the *near hole region*. The routing path from this intermediate node to destination node is constructed same as the first case. In two later cases, the near hole routing protocol is used to find the path from the source to the intermediate node or from the intermediate node to the destination.

The detailed scheme is described in the following subsections.

3.1.3 Hole Boundary Detection

The *TENT rule* is executed to identify the node on the hole boundary. Then, the *Hole Boundary Detection* (*HBD*) packet is formed and forwarded around the hole boundary using the *BOUNDHOLE* algorithm. Assume H_0 is the leader node of the hole. After the *HBD* packet came back to H_0, it has the information of all hole boundary sensor node $H_0, H_1, H_2, ..., H_n$. H_0 then uses the *Graham scan* algorithm to determine the convex hull of the hole polygon. The information of each hole boundary sensor node is stored as format described in table 3.1.

Table 3.1: Storage scheme of a hole boundary sensor node.

x	x-coordinate of node
y	y-coordinate of the node
isExtremePoint	true if it is a vertex of the convex hull and false otherwise

3.1.4 Hole Information Dissemination

After the convex hull construction, node H_0 then creates a hole polygon information (*HPI*) message that stores the information of the polygon vertices (which is the format described in table 3.1), and disseminates this *HPI* message to its neighbors. In addition, to reduce the overhead of transmission information, before disseminating, H_0 will try to eliminate the vertices of any inner polygons whose vertices number is less than the system parameter η (our experiments show that with an appropriately chosen value of η (e.g., $\eta = 10$), it reduces the message overhead and still ensures the routing mechanism). This eliminating algorithm is described in algorithm 2.

Algorithm 2: Hole polygon's vertices eliminating algorithm

```
Input  : $P = \{P_1, P_2, ..., P_n\}$: the linked list of the hole polygon vertices in counterclockwise
         order.
         $\eta$: pre-defined system parameter.
Output : $P$: simplified hole polygon.
for $i = 1$ to $n$ do
    if $P[i].isExtremePoint = false$ then
        Remove $P[i]$ from $P$;
        Append $P[i]$ to the end of $P$;
    else break;
end
for $i = 1$ to $n$ do
    if $P[i].isExtremePoint = true$ then
        $j \leftarrow 0$;
        $m \leftarrow i$;
        while $P[i+1].isExtremePoint = false$ do
            $j \leftarrow j + 1$;
            $i \leftarrow i + 1$;
        end
        $i \leftarrow i - 1$;
        if $j \leq \eta$ then Remove $\{P_{m+1}, P_{m+2}, ..., P_i\}$ from $P$ ;
    end
end
return $P$;
```

When an intermediate node N receives this *HPI* message, it does following tasks:

- Stores the information of the hole polygon to its local memory.
- If N is inside the convex hull of the hole polygon or its distance to the convex hull of hole polygon less than $\delta \times R_c$, it forwards the message to its neighbors.
- Otherwise, it drops the message.

After this phase finished, the network is divided into two regions, and all the nodes that have hole information formed the *near hole region*.

In addition, all sensor nodes also construct its *core polygon* which is an octagon. This core polygon has to satisfy the following conditions:

- It completely covers the hole polygon.

- It has at most 8 vertices and each edge of its contains at least one vertex of the hole.
- All angles at the vertices that are not the vertices of the hole, equal to $\frac{3\pi}{4}$.

The reason for approximating the hole by an octagon is explained in subsection 3.1.6. Algorithm 3 describes how to construct the core polygon of specific node N.

Algorithm 3: Core polygon construction algorithm

```
Input   : $P = \{P_1, ..., P_n\}$: the set of hole polygon vertices in counterclockwise order.
          $N$: current sensor node.
Output  : $A = \{A_1, A_2, ..., A_8\}$: the core polygon of $N$.
$P_j \leftarrow$ view-limit vertex of $N$ respect to $P$;
$i \leftarrow 1$;
$\alpha \leftarrow$ angle of $P_j$ with x-axis;
$L \leftarrow \{P_j N\}$;
for $k = 1$ to $n$ do
    $l \leftarrow$ line goes through $P_k$ and makes an angle of $\alpha + i \times \frac{\pi}{4}$ with x-axis;
    if All hole polygon nodes stay on the left side of $l$ then
        $L \leftarrow L \cup \{l\}$;
        $i \leftarrow i + 1$;
    end
    if $i > 8$ then break;
end
for $k = 1$ to $8$ do
    $A_k \leftarrow L[k] \cap L[k+1]$;
end
return $A$;
```

3.1.5 Internal Graph Construction

The most difficult part of building a near hole routing protocol is to find an optimal path for a packet to a destination node inside an inner polygon or from a source node inside an inner polygon to the outside of the convex hull. The *Internal Graph Construction* phase is responsible for building an internal graph so that nodes are able to find such an optimal path. To ensure the requirements of a routing protocol that are load balancing and route stretch, this optimal path need to be as short as possible and is different for each sensor node. Our idea is to let every sensor node inside an inner polygon learn about the internal structure of that inner polygon via representing it by a *skeleton*. This *skeleton* is presented as a undirected acyclic graph, and it is built from the Voronoi diagram of the polygon without any Voronoi edges that does not inside the polygon. Figure 3.4 shows an example of a *skeleton*.

Because this graph is calculated locally in each sensor node, node needs to know the location of only one point such that, if the packet is forwarded to that point, it still can find the way out the inner polygon by forwarding to the next point. We call such that point **endpoint**. On the other hand, from a point in the *gateway*, we can re-construct the graph to find the way in from the gateway to the destination point.

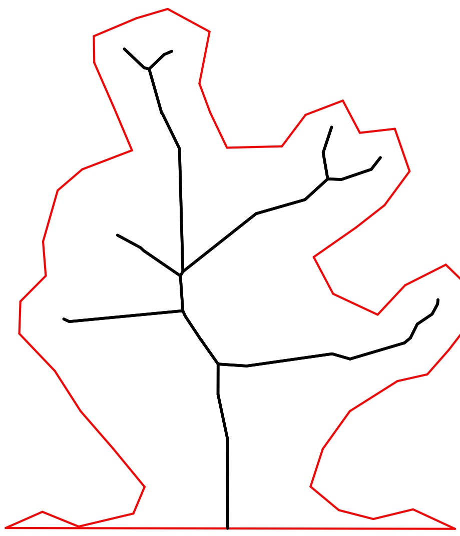


Figure 3.4: An example of polygon's skeleton.

We call such that point **gate-point**. We also note that to reduce the node state, this graph is calculated only for each node to determine its *endpoint*, and the nodes save the location of its *endpoint* to its local memory, not the graph. Algorithm 4 describes how to construct such a skeleton and determine the *endpoint* and *gate-point* of a sensor node.

By determining the *endpoint* and *gate-point* of a sensor node, we can route a packet to a destination node inside an inner polygon or route a packet from a source node inside an inner polygon to the outside of the convex hull. The details of this algorithm are presented in the next subsection. Note that this *Internal Graph Construction* is performed only by nodes staying inside an inner polygon.

3.1.6 Data Forwarding

In this subsection, we present the hole bypassing algorithm. As described above, the near hole routing path is constructed from VA-points. There are three type of VA-points: the source's VA-point, the VA-points which are vertices of scale octagon and the destination's VA-point. We use the octagon instead of the convex hull to reduce the number of VA-points' information stored in each packet, since the path constructed from the convex hull can contain a lot of VA-points and its number depends on the complexity of the convex hull. The routing path consists of three main periods: the first is from the source to its VA-point, the second is from the source's VA-point to the destination's VA-point and the last is from the destination's VA-point to the destination. Below we show how to determine these VA-points and the routing path of each period.

When a data packet is formed, the source node initiates its VA-point as its endpoint. The first period terminates immediately in case the source node is outside the convex hull. Otherwise, the packet is forwarded to this VA-point. At each intermediate node,

Algorithm 4: Internal graph construction, endpoint and gate-point determination algorithm

```
Input   : $N$: current sensor node
Output  : $E_N$: endpoint of $N$
          $G_N$: gate-point of $N$
if $N$ is not inside any inner polygons then
    $E_N \leftarrow N$;
    $G_N \leftarrow null$;
else
    $IP_{P_{hk}} \leftarrow$ inner polygon contains $N$;
    $P_hP_k \leftarrow$ gateway of $IP_{P_{hk}}$;
    $E_{IP_{P_{hk}}} \leftarrow$ set of edges of $IP_{P_{hk}}$;
    $l \leftarrow$ perpendicular line from $N$ to $P_hP_k$;
    $H \leftarrow l \cap P_hP_k$;
    if $NH \cap \{E_{IP_{P_{hk}}} \setminus P_hP_k\} = \varnothing$ then
        $E_N \leftarrow H$;
        $G_N \leftarrow null$;
    else
        Construct Voronoi diagram $\upsilon$ of $IP_{P_{hk}}$;
        $\gamma \leftarrow$ Voronoi site contains $N$;
        $V_j, V_{j+1}, ..., V_u \leftarrow$ Voronoi vertices of $\gamma$ inside $IP_{P_{hk}}$;
        $K \leftarrow P_{i_h}P_{i_k} \cap e$ such that $e$ has a vertex inside $IP_{P_{hk}}$ and the other vertex outside
          $IP_{P_{hk}}$;
        Eliminate all Voronoi edges not inside $IP_{P_{hk}}$ except $e$;
        Construct graph from remaining edges;
        Traversal the graph using BFS algorithm to find the shortest path from
          $V_j, V_{j+1}, ..., V_u$ to $K$;
        $V_w \leftarrow$ is vertex corresponding to the shortest path among these paths;
        $E_N \leftarrow V_w$;
        $G_N \leftarrow K$;
    end
end
return $E_N, G_N$;
```

the VA-point field is updated to the endpoint of current node. We note that the routing path inside the convex hull is different from each sensor node. It depends on the location of current sensor node, thus it still ensures the load balancing in inside the forbidden area. This period terminates when the packet reaches the *gateway*. In case the source node does not belong to the *near hole region*, it simply greedily forwards the packet the destination until it reaches an intermediate node belonging to the *near hole region*.

Assume I_s is this intermediate node. I_s then uses its core polygon information A to construct a scale polygon. This scale polygon is the image of core polygon A through a homothetic transformation using a center I chosen randomly inside the core polygon. The scale factor ξ is computed based on the distance of I_s to the convex hull and the system parameter δ as follows:
Let d is maximum distance from I to the vertices of A.

- $\xi = 1 + \frac{\delta \times R_c}{d}$ if I_s is inside or on the edge of the convex hull
- $\xi = 1 + \frac{d(C_P, I_s) \times R_c}{d}$ if I_s is outside the convex hull

I_s also computes the destination's VA-point I_d as follows:

- If the destination node d is not inside any inner polygons, I_d is the destination node.
- Otherwise, I_d is the *gate-point* constructed from the internal graph of the inner polygon containing the destination node.

Then the algorithm 5 is executed to determine VA-points which are vertices of scale octagon. Figure 3.5 illustrates the algorithm.

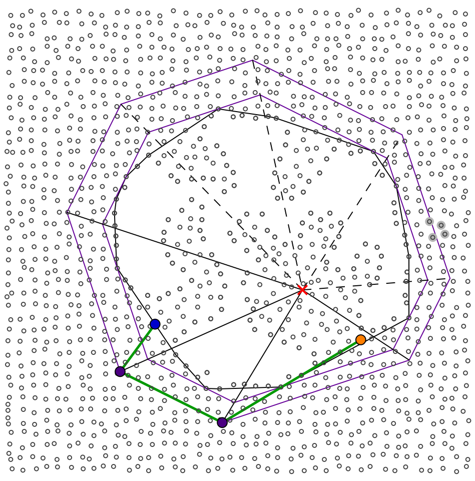


Figure 3.5: An example of choosing VA-points which are vertices of scale polygon.

Algorithm 5: Virtual anchor points determination algorithm

```
Assume $I_s$ is current sensor node, $I_d$ is the destination's VA-point and $A' = \{A'_1, A'_2, ..., A'_m\}$
  $(4 \leq m \leq 8)$ is the scale polygon with homothetic center $I$.
Four points $L_s, L_d, R_s, R_d$ are computed as follows:
if $I_s$ is inside $A$ then
  | $A'_i, A'_{i+1} \leftarrow$ two consecutive vertices of $A'$ such that $\triangle I A'_i A'_{i+1}$ contains $I_s$
  | $L_s, R_s \leftarrow A'_i, A'_{i+1}$
else
  | $L_s, R_s \leftarrow$ view-limit vertices of $I_s$ respect to $A'$
end
$L_d, R_d$ is computed same as $L_s, R_s$.
Assume $L_s, L_s$ stay on the left side of $I_sI_d$ and $R_s, R_d$ stay on the right side of $I_sI_d$. The hole
  bypassing routing path is chosen as the shorter path between $\overrightarrow{I_sL_s...L_dI_d}$ and $\overrightarrow{I_sR_s...R_dI_d}$.
```

The information of virtual anchor points is added to the packet header. The packet then is forwarded greedily to each virtual anchor point.

In the last virtual anchor point, i.e. the destination's VA-point, it re-construct the internal graph of the inner polygon containing the destination in case the destination is inside an inner polygon, or it simply use the greedy protocol to forward the packet to the destination. If the destination is inside an inner polygon, the internal graph construction is performed in some specific nodes, that are the Voronoi vertices of the graph, not all the nodes in the routing path. In this process, the BFS algorithm is executed to trace back the shortest path from the *gate-point* to the destination node. This process terminates when the packet reaches the *endpoint* of the destination node.

Algorithm 6 presents the hole bypassing protocol.

Algorithm 6: Hole bypassing protocol.

```
Input   : $N$: current sensor node
          $D$: destination node
          $AP$: the set of virtual anchor points
          $i$: current index of $AP$
Output  : $next$: the location of the next hop
Assume $getNextHopByGreedy(N, D$ returns the neighbor of $N$ which is nearest to $D$;
$period \leftarrow$ current routing period;
if $period = 1$ then
    $I_d \leftarrow$ endpoint of $N$;
    $AP[6] \leftarrow I_d$;
    $i \leftarrow 6$;
end
if $N$ is a virtual anchor point then
    $i \leftarrow i - 1$;
else if $N$ is its endpoint then
    $A \leftarrow$ core polygon;
    $A' \leftarrow$ scale polygon;
    $A'_1, A'_2, A'_3, A'_4 \leftarrow$ virtual anchor points computed from $A'$;
    $AP[2..5] \leftarrow A'_1, A'_2, A'_3, A'_4$;
    $i \leftarrow 5$;
end
$next \leftarrow getNextHopByGreedy(N, AP[i])$;
return $next$;
```

3.2 Theoretical Analysis

3.2.1 Impact of parameter δ over the protocol

The value of δ decides the area of the near hole region. The greater the value of δ is, the bigger the near hole region is. In the following, We show that the area of near hole region has a big impact on the two factors: energy consumption of the network and route stretch.

In case δ is big, the are of near hole region is big. Thus, the *HPI* message is broadcast further, and more. Consequently, the energy consumption of the network increases. If δ is small, the *HPI* message is broadcast lesser. Hence, the energy consumption of the network decreases.

In addition, if δ is small, the scale factor of the homothetic transformation is small. It makes all scale octagons almost the same. With a randomly deployed network, the density of sensor nodes is not really high, this would not create different routing path for different scale octagon. Thus, it decreases the load balancing. Vice versa, if δ is big, the scale factor is big, the virtual anchor points are far from the core polygon. It leads to the path enlargement problem.

According to mentioned analysis, obviously, the value of δ affects load balancing, energy consumption as well as route stretch. We also conducted experiments to prove these analysis and give a conclusion on the appropriate value of δ.

Chapter 4

The WisSim toolkit

The WisSim is a visual WIreless Sensor network SIMulation toolkit, which was developed by researchers at SEDIC Lab [2] for building a software tool that could help researchers and algorithm designers to test their newly proposed algorithms for wireless sensor networks.

The WiSSim architecture follows the Client-Server architecture where at the client side it provides the front-end components for the user to define the test networks, algorithms and execution scripts, and possibly ask for simulation then run the visualization and analysis tools to enjoy seeing the test analysis results provided with care towards the user convenience and deep understanding. At the back-end, it exploits a popular existing network simulator software, that is *ns-2*. The WiSSim server acts as a connector to work with *ns-2*: preparing the Tcl scripts to be executed in *ns-2* and retrieving and interpreting the trace files outputted from *ns-2*. Besides WisSim server also implements into the *ns-2* core certain selected routing protocols [2].

The remaining of this chapter describes an overview of *ns-2* simulator and some features which have been implemented in WisSim client by the author.

4.1 ns-2 simulator overview

The *ns-2* is a discrete-event simulation framework, which was developed by researchers at UC Berkeley from 1995. The general architecture of *ns-2* is presented in figure 4.1.

The simulation script is written in Tcl language. A simulation script must have:

- Network topology including network's size, number of sensor nodes, the configuration of each sensor nodes, network infrastructure, network protocol, and simulation time.
- Declaration of events: the start time of the event, type of events, events' commands ...

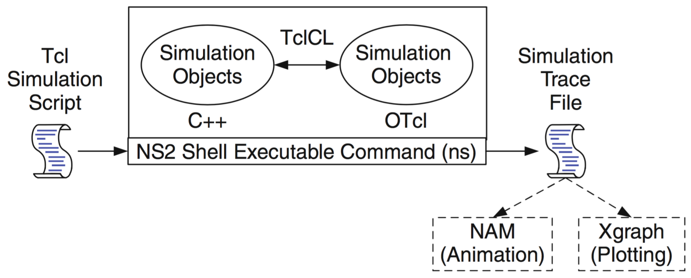


Figure 4.1: The architecture of *ns-2*.

- Output format.

The output of simulation process called trace files. By analyzing these trace files, we can calculate network's metrics such as energy consumption, drop ratio or average path length, ...

The *ns-2* uses two programming language: Tcl and C++. All the routing algorithms which are deployed into *ns-2* core must be written in C++. The component of *ns-2* could be classified into 4 groups based on their principal functions:

- Simulation-related Objects: controls and manages the simulation time. These objects are events, handlers, scheduler and simulator.
- Network Objects: manages packet (transmit, receive, or drop).
- Packet-related Objects: includes supporting packets.
- Helper Objects: such as a routing protocol.

As mentioned above, the *ns-2* is a discrete-event simulation framework. More particularly, each event contains the time it will execute, its tasks and the link to the next event. These events create an event-chain, that is a single linked list and is created during the simulation process. Figure 4.2 shows an example of event-chain.

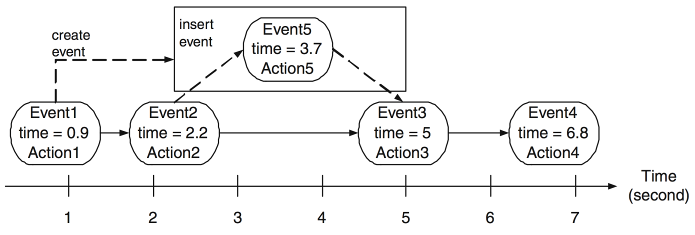

create event
insert event
Event5 time = 3.7 Action5
Event1 time = 0.9 Action1
Event2 time = 2.2 Action2
Event3 time = 5 Action3
Event4 time = 6.8 Action4
Time (second)
1
2
3
4
5
6
7

Figure 4.2: An example of event-chain in *ns-2*.

4.2 WisSim Client

WisSim client is a software in the *WisSim* toolkit which targets to have users create a network topology and analyze its properties. WisSim client consists of three main feature: WisSim Editor, WisSim Visualizer and WisSim Analyzer. In this section, we describe our addition features and improvements for WisSim client software.

4.2.1 Functional requirements

Table 4.1: Functional requirements

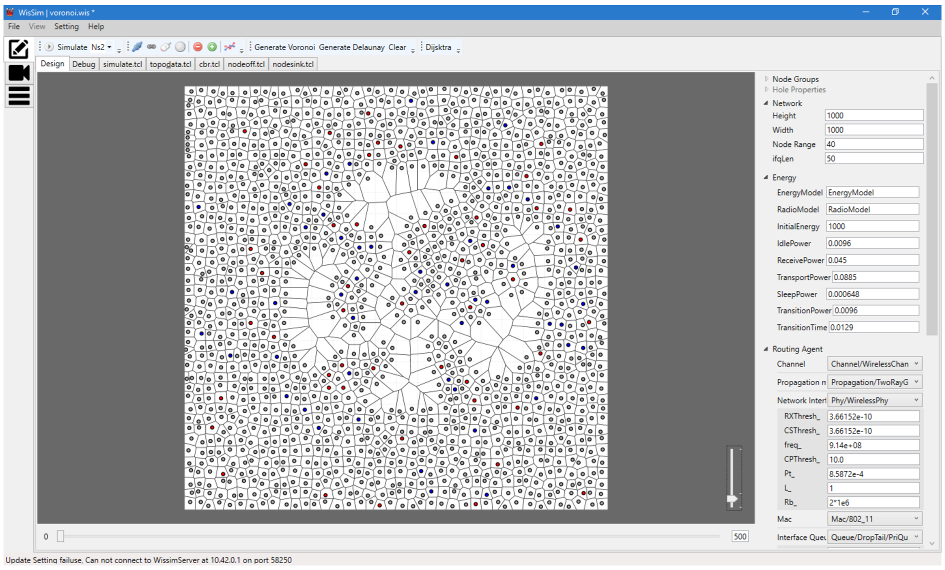
Function	Condition	Input	Output	Module
Voronoi Diagram	Working project	Node list	Display	WisSim Editor
Graph	Working project	Node list	Shortest path computed from Dijkstra algorithm	WisSim Editor

4.2.2 Class design

Table 4.2: Packet list.

Packet	Implementation
Graph/VoronoiDiagram	This packet contains implementation of Voronoi diagram construction algorithm and computational geometry library (such as line, point, circle, parabola)
Graph/Graph	This packet implements primitive graph algorithms such as BFS, Dijkstra, ...

4.2.3 Screen design



Network

Height	1000
Width	1000
Node Range	40
ifqLen	50

Energy

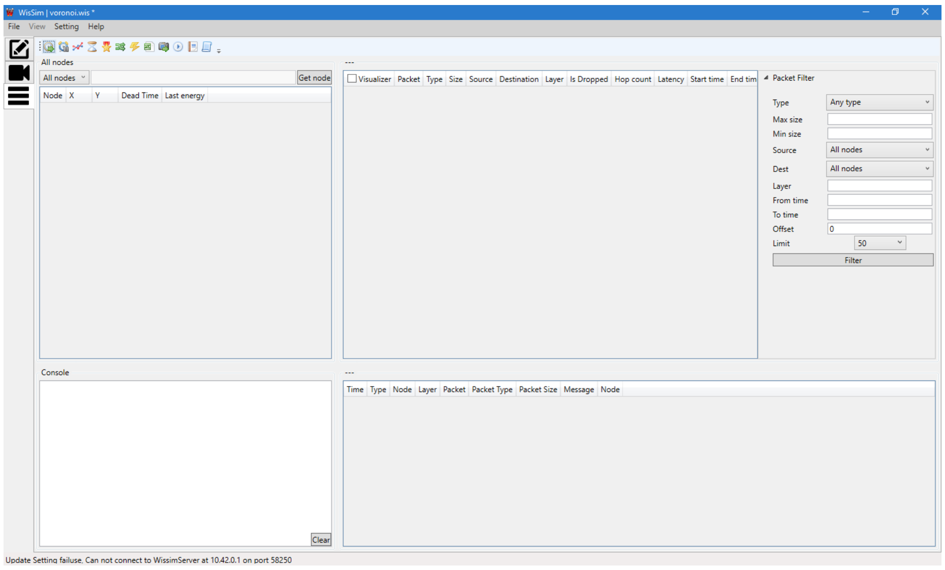
EnergyModel	EnergyModel
RadioModel	RadioModel
InitialEnergy	1000
IdlePower	0.0096
ReceivePower	0.045
TransportPower	0.0885
SleepPower	0.000648
TransitionPower	0.0096
TransitionTime	0.0129

Routing Agent

Channel	Channel/WirelessChan
Propagation m	Propagation/TwoRayG
Network Interf	Phy/WirelessPhy
RXThresh_	3.66152e-10
CSThresh_	3.66152e-10
freq_	9.14e+08
CPThresh_	10.0
Pt_	8.5872e-4
L_	1
Rb_	2*1e6
Mac	Mac/802_11
Interface Queu	Queue/DropTail/PriQu

Update Setting failuse. Can not connect to WissimServer at 10.42.0.1 on port 58250

Figure 4.3: Screen of Voronoi diagram construction.



Type	Any type
Max size	
Min size	
Source	All nodes
Dest	All nodes
Layer	
From time	
To time	
Offset	0
Limit	50

Update Setting failuse. Can not connect to WissimServer at 10.42.0.1 on port 58250

Figure 4.4: New design of WisSim Analyzer screen.

Chapter 5

Protocol implementation

5.1 Implementation details

The proposed protocol is implemented in C++ and embedded into the core of current *WisSim* toolkit. The source code is organized as follows.

- *geometry*: contains implementation of geometry objects such as Point, Line, Circle, Vector, Parabola and its operators.
- *graph/voronoi*: contains implementation of Voronoi diagram construction using *Fortune* algorithm.
- *nearholerouting*: contains implementation of the proposed protocol.

  −*nhr.cc*: definition of class *NHRAgent*. This class is the core of the protocol.

  −*nhr.h*: header of class *NHRAgent*.

  −*nhr_graph.cc*: definition of class *NHRGraph*. This class implements operators to construct the internal graph and compute the shortest path.

  −*nhr.cc*: header of class *NHRGraph*.

  −*nhr_packet.h*: declaration of packet header of protocol.

  −*nhr_packet_data.cc*: definition of class *NHRPacketData*. This class provides methods to work with the packet payload such as add, remove, count, get size operator.

  −*nhr_packet_data.h*: header of class *NHRPacketData*.

In order to integrate our routing protocol into *ns-2* core, we defined a new packet type for our proposed protocol and gave default values for binded attributes. Firstly, we added new *NHR* packet type into *tcl/lib/ns-packet.tcl*. We also added *PT_NHR* to the *packet_t* enumeration in *common/packet.h*. Default values for binded attributes are given inside *tcl/lib/ns-default.tcl*. Finally, we modified *tcl/lib/ns-lib.tcl* to add procedures for creating a node with *NHR* protocol.

In *nhr.cc*, we implemented the core of our proposed protocol. The class *NHRAgent* inherits class *Agent* of *ns-2* and overrides *command()* and *recv()* method.

5.2 Source code listing

Packet header

The packet header of our proposed protocol is defined in *nhr_packet.h* source file.

```
struct hdr_nhr {
    nsaddr_t daddr_;
    Point dest_;
    Point anchor_points[10];
    uint8_t ap_index;
    uint8_t type;

    inline int size() {
        return 11 * sizeof(Point) + 2 * sizeof(uint8_t) + sizeof(
    nsaddr_t);
    }

    static int offset_;

    inline static int &offset() {
        return offset_;
    }

    inline static struct hdr_nhr *access(const Packet *p) {
        return (struct hdr_nhr *) p->access(offset_);
    }
};
```

nhr_packet.h

The meaning of each field is described below:

- *daddr_*: destination node' ID
- *dest_*: destination node's location information
- *anchor_points*: array of virtual anchor points
- *ap_index*: current index of *anchor_points* array
- *type_*: packet type. It receives one of following values:
  - *NHR_CBR_GPSR*: indicates greedy forwarding mode
  - *NHR_CBR_AWARE_SOURCE_ESCAPE*: indicates near hole routing mode, from inner polygon to outside

- *NHR_ CBR_ AWARE_ SOURCE_ PIVOT*: indicates near hole routing mode
- *NHR_ CBR_ AWARE_ OCTAGON*: indicates near hole routing mode using scale octagon
- *NHR_ CBR_ AWARE_ DESTINATION*: indicates near hole routing mode for routing to destination

The *size()* function returns the total size in byte of all above fields.

Core protocol

Our core protocol is implemented in *nhr.cc* source file. In this file, the *recvData()* method is source code of the hole bypassing protocol.

```
void NHRAgent::recvData(Packet *p) {
  struct hdr_cmn *cmh = HDR_CMN(p);
  struct hdr_nhr *edh = HDR_NHR(p);
  node *nexthop = NULL;

  if (cmh->direction() == hdr_cmn::UP
      && edh->daddr_ == my_id_) // up to destination
  {
    port_dmux_->recv(p, 0);
    return;
  }

  if (edh->type == NHR_CBR_GPSR && !hole_.empty()) {
    edh->type = NHR_CBR_AWARE_SOURCE_ESCAPE;
    edh->ap_index = 1;
    edh->anchor_points[edh->ap_index] = endpoint_;
  }

  nexthop = getNeighborByGreedy(
              edh->anchor_points[edh->ap_index]);
  if (nexthop == NULL) {
    switch (edh->type) {
      case NHR_CBR_AWARE_SOURCE_ESCAPE:
        edh->anchor_points[1] = endpoint_;
        if (isPivot_) {
          edh->type = NHR_CBR_AWARE_SOURCE_PIVOT;
        }
        nexthop = getNeighborByGreedy(
                    edh->anchor_points[edh->ap_index]);
        if (nexthop != NULL) break;
      case NHR_CBR_AWARE_SOURCE_PIVOT:
        if (determineOctagonAnchorPoints(p)) {
          edh->type = NHR_CBR_AWARE_OCTAGON;
        } else {
            edh->ap_index = 0;
            edh->type = NHR_CBR_AWARE_DESTINATION;
        }
        nexthop = getNeighborByGreedy(
                    edh->anchor_points[edh->ap_index]);
        if (nexthop != NULL) break;
      case NHR_CBR_AWARE_OCTAGON:
        while (edh->ap_index > 2 && nexthop == NULL) {
          --edh->ap_index;
          nexthop = getNeighborByGreedy(
                      edh->anchor_points[edh->ap_index]);
        }
        if (edh->ap_index == 2) {
          edh->ap_index = 0;
          edh->type = NHR_CBR_AWARE_DESTINATION;
        }
        nexthop = getNeighborByGreedy(
                    edh->anchor_points[edh->ap_index]);
        if (nexthop != NULL) break;
      case NHR_CBR_AWARE_DESTINATION:
        routeToDest(p);
        nexthop = getNeighborByGreedy(
                    edh->anchor_points[edh->ap_index]);
        break;
      default:
        drop(p, DROP_RTR_NO_ROUTE);
        return;
    }
  }

  if (nexthop == NULL) {
    drop(p, DROP_RTR_NO_ROUTE);
    return;
  } else {
      cmh->direction() = hdr_cmn::DOWN;
      cmh->addr_type() = NS_AF_INET;
      cmh->last_hop_ = my_id_;
      cmh->next_hop_ = nexthop->id_;
      send(p, 0);
  }
}
```

nhr.cc

Tcl hooking

The procedure to initiate our protocol is written in Tcl language in *ns-lib.tcl* source file. This procedure first creates an instance of routing protocol (line #3), then sets up the parameters for each sensor node (line #4 - #11). Our protocol receives 2 commands to start specific events. In the beginning of the simulation process (0.0 second), all the sensor nodes start the protocol and send the HELLO packet to collect their neighbors information (line #12). Then at second 30.0, the *BOUNDHOLE* algorithm is started (line #13).

```
1  # NHR
2  Simulator instproc create-nhr-agent { node } {
3    set ragent [new Agent/NHR]
4    set addr [$node node-addr]
5    $ragent addr $addr
6    $ragent node $node
7    if [Simulator set mobile_ip_] {
8      $ragent port-dmux [$node demux]
9    }
10   $node addr $addr
11   $node set ragent_ $ragent
12   $self at 0.0  "$ragent␣start"    ;# start updates
13   $self at 30 "$ragent␣boundhole"
14   return $ragent
15 }
```

ns–lib.tcl

The default binding attributes for the protocol is declared in *ns-default.tcl* source file. Our protocols takes 4 parameters. The name and meaning of each parameter are described in table 5.1.

Table 5.1: Protocol parameter

Parameter	Meaning
limit_boundhole_hop_	Upper bound of vertices number of the hole polygon
hello_period_	Period time to send HELLO packet (0 to send one time only)
range_	40m
delta_	value of δ parameter
n_	value of η parameter

```
Agent/NHR set limit_boundhole_hop_ 100
Agent/NHR set hello_period_ 0
Agent/NHR set range_ 40
Agent/NHR set delta_ 2.1
Agent/NHR set n_ 10
```

ns–default.tcl

Chapter 6

Evaluation

In this chapter, we compare our proposal (called NHR) with two existing proposals, GPSR [11], BOUNDHOLE [9]. The metrics considered in our evaluation are the average energy consumption, the deviation of energy consumption, the energy consumption of individual sensor nodes, the network lifetime, the average path length and the routing path stretch. We assume that 1500 sensor nodes with 802.11 MAC protocol are deployed randomly in an area of 1000m x 1000m.

All parameters are summarized in table 6.1. Nodes follow the energy model suggested by [15] (see table 6.2). We investigate the cases until the operation of a network reaches 1000s and a time when the first node fails (denoted by FNF).

Table 6.1: Simulation parameters.

Parameter	Value
Area	1000m $\times$ 1000m
Number of nodes	1500
Node communication range	40m
Number of CBR source	20/25/30/35/40/50
Bandwidth	2Mbps
Packet sending period	3s
Data packet size	50 bytes

6.1 Average energy consumption

The average energy consumption is calculated by dividing the total consumed energy of all the nodes to the number of the nodes. Figure 6.1 plots the average energy consumption for six different routing protocols. It can be observed that the energy consumption for *NHR* and the curve slope are smallest.

Table 6.2: Node parameters.

Parameter	Value
Type of channel	Channel/WirelessChannel
MAC type	MAC/802.11
Network interface type	Phy/WirelessPhy
Interface queue model	Queue/DropTail/PriQueue
Transmission of radio	Propagation/TwoRayGround
Energy model	EnergyModel
Routing protocol	GPSR/BOUNDHOLE/NHR
Node initial energy	1000J/10J
Length of the network interface queue	50
Node idle power	9.6mW
Node receive power	45mW
Node transmit power	88.5mW

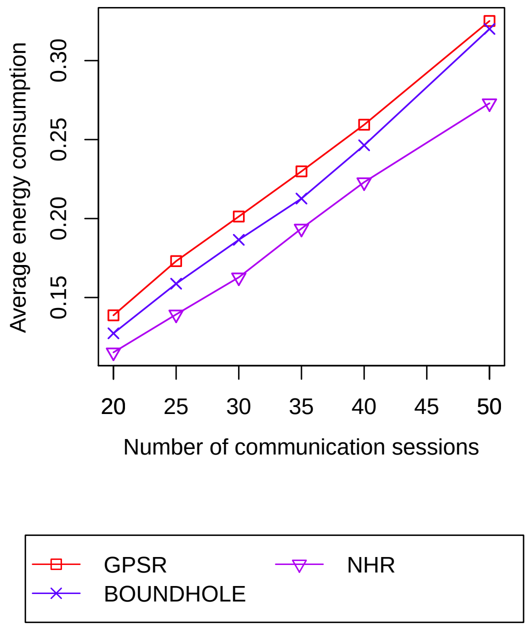


Figure 6.1: Comparison of average energy consumption

6.2 Deviation of energy consumption

The deviation of energy consumption is calculated by the following formula: $\sigma = \sqrt{\frac{1}{K}\sum_{i=1}^{K}\left(e_i - \frac{\sum_{i=1}^{K} e_i}{K}\right)^2}$, where K is number of sensor nodes and e_i is energy consumed by the i^{th} node. The deviation of energy consumption of sensor nodes can give us a view of energy balancing. The more even the distribution of energy consumption is the better load balancing the protocol achieves. The smaller the deviation is, the higher the load balancing of network will be.

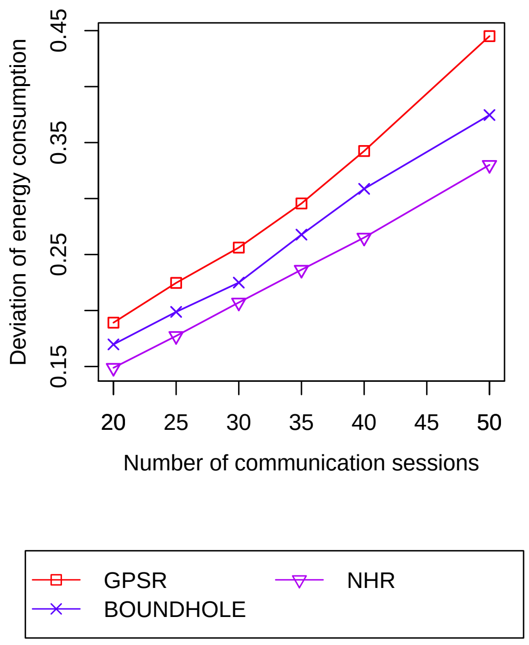


Figure 6.2: Comparison of deviation of energy consumption

6.3 Energy consumption of individual sensor nodes

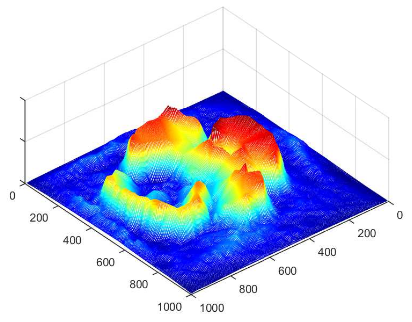
Figure 6.3 scatters the energy consumption of individual sensor nodes in 3-dimension graphs. These figures give us a visual view of load balancing status of network. It is clear that, the per-node energy consumption by using GPSR and BOUNDHOLE are the most imbalanced. This imbalance is illustrated best by a look at the peaks at the center of figure 6.3a, 6.3b. The position of the peaks correspond to the location of the hole.

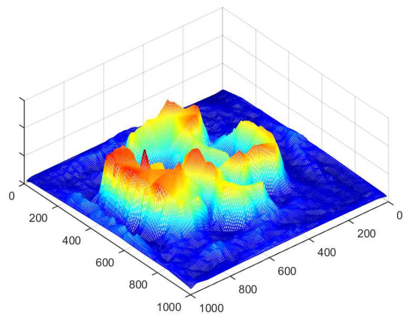
6.4 Average path length

The average path length is calculated as the average hop-count of all successfully transmitted packets. The comparison of the average path length for three protocols is shown in figure 6.4.

6.5 Routing path stretch

In this section we also evaluate the stretch by hop-count (i.e. the most practical stretch metric). The stretch by hop-count is the ratio between the hop-count of the routing path using routing protocol and the optimal path.


(a) GPSR


(b) BOUNDHOLE

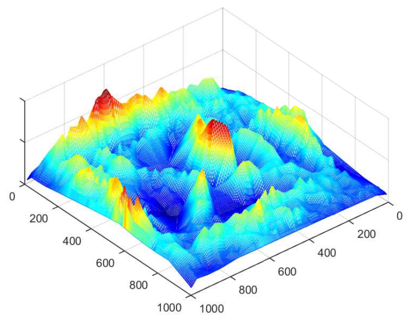

(c) NHR

Figure 6.3: Visual view of energy consumption of individual sensor nodes

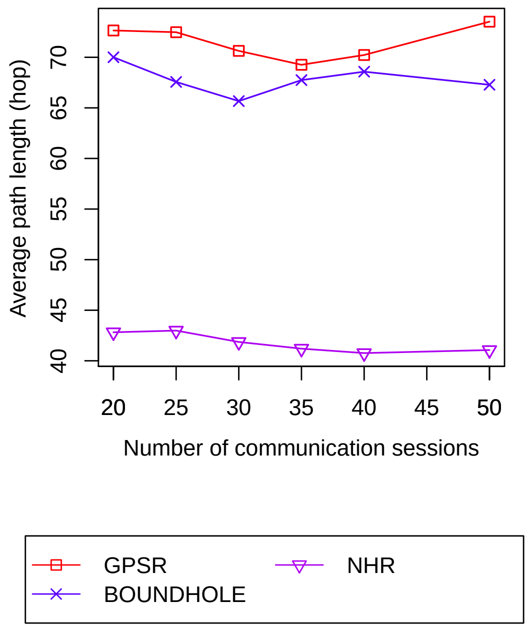
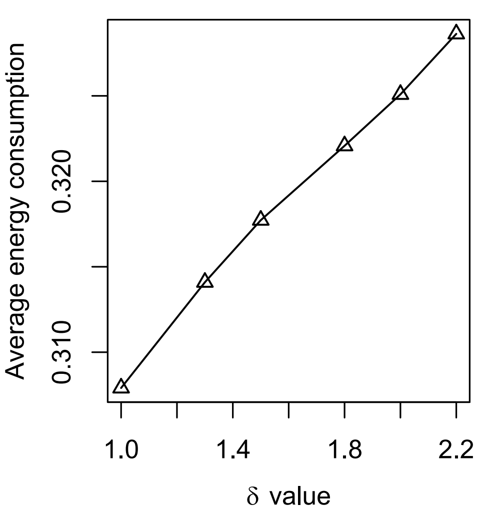


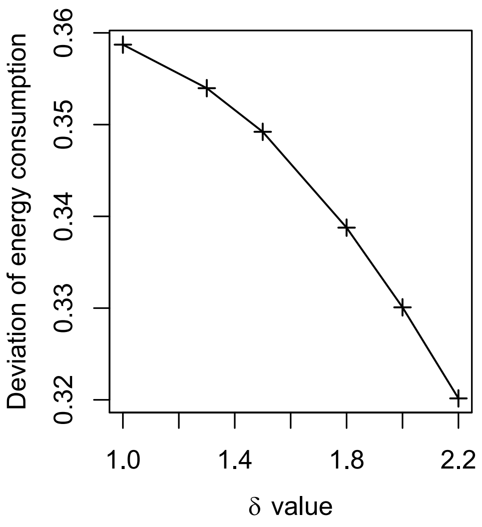
Figure 6.4: Comparison of average path length

6.6 Impact of δ on protocol

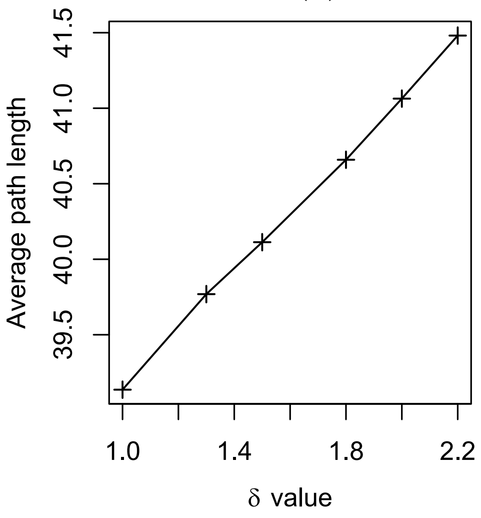
We also conduct another experiment to investigate the impact of δ on the protocol. Three metrics are compared: the average path length, the average energy consumed, the deviation of energy consumption.



(a) Average energy consumption



(b) Deviation of energy consumption



(c) Average path length

Figure 6.5: Impact of δ on: average energy consumption, deviation of energy consumption and average path length

Chapter 7

Conclusion

Routing protocol designing is classical problem in network communication. With the wireless sensor network, especially the wireless sensor network with harsh condition, the routing protocol designing becomes more difficult due to the existence of routing holes. Although a lot of protocols have been proposed, none of them targets and solves the near hole routing problem. In this thesis, we have thorough studied the routing in wireless sensor network with holes problem. Our research proposed a novel algorithm that targets and solves the near hole routing problem. Through theoretical analysis and simulations, we proved that our proposed ensures both two main requirements of a routing protocol: energy consumption and load balancing. As future work, we will extend our protocol for the multiple holes problem, support the network model with multiples sinks or propose an adaptive protocol for the dynamic routing hole.

References

[1] https://www.eol.ucar.edu/isf/facilities/isa/internal/crossbow/datasheets/mica2.pdf. 2

[2] sedic.soict.hust.edu.vn/. 29

[3] sedic.soict.hust.edu.vn/wissim/. 5

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, **38**[4]:393–422, March 2002. 1

[5] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008. 6, 20

[6] Dan Chen, Zhixin Liu, Lizhe Wang, Minggang Dou, Jingying Chen, and Hui Li. Natural disaster monitoring with wireless sensor networks: A case study of data-intensive applications upon low-cost scalable systems. *Mobile Networks and Applications*, **18**[5]:651–663, October 2013. 1

[7] Hyunseung Choo, Mooshik Choi, Minhan Shon, and Doogsoo Stephen Kim. Efficient hole bypass routing scheme using observer packets for geographic routing in wireless sensor networks. *SIGAPP Appl. Comput. Rev.*, **11**[4]:7–16, December 2011. viii, 4, 10, 11, 12, 18

[8] Hyunseung Choo, Mooshik Choi, Minhan Shon, and Doogsoo Stephen Kim. Efficient hole bypass routing scheme using observer packets for geographic routing in wireless sensor networks. *SIGAPP Appl. Comput. Rev.*, **11**[4]:7–16, December 2011. 10, 11

[9] Qing Fang, Jie Gao, and Leonidas J. Guibas. Locating and bypassing holes in sensor networks. *Mob. Netw. Appl.*, **11**[2]:187–200, April 2006. 4, 16, 20, 38

[10] Ting-Chao Hou and Victor Li. Transmission range control in multihop packet radio networks. *IEEE Transactions on Communications*, **34**:38–44, January 1986. 2

[11] Brad Karp and H. T. Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, MobiCom '00, pages 243–254, New York, NY, USA, 2000. ACM. 4, 15, 38

[12] Nguyen Phi Le, Nguyen Trung Hieu, and Nguyen Khanh Van. Elbar: Efficient load balanced routing scheme for wireless sensor networks with holes. In *Proceedings of the Third Symposium on Information and Communication Technology*, SoICT '12, pages 190–199, New York, NY, USA, 2012. ACM. 4, 10, 11, 18

[13] Nguyen Phi Le, Bui Tien Quan, Nguyen Trung Hieu, and Nguyen Khanh Van. Efficient approximation of routing holes in wireless sensor networks. In *Proceedings of the Second Symposium on Information and Communication Technology*, SoICT '11, pages 72–79, New York, NY, USA, 2011. ACM. 13

[14] Phi-Le Nguyen, Duc-Trong Nguyen, and Khanh-Van Nguyen. Load balanced routing with constant stretch for wireless sensor network with holes. In *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore, April 21-24, 2014*, pages 1–7, 2014. 4, 10, 11, 18

[15] Victor Shnayder, Mark Hempstead, Bor-rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 188–200, New York, NY, USA, 2004. ACM. 38

[16] Ye Tian, Fucai Yu, Younghwan Choi, Soochang Park, Euisin Lee, Minsook Jin, and Sang-Ha Kim. Energy-efficient data dissemination protocol for detouring routing holes in wireless sensor networks. 4, 10, 11