國立交通大學

資訊科學與工程研究所

碩 士 論 文

藉由無線感測器網路建立一室內 3D 環境下的緊急逃生系統

Implementation of an Emergency Guiding System in Indoor 3D Environment by Wireless Sensor Networks

研 究 生：蔡佳宏

指導教授：曾煜棋　教授

中 華 民 國 九 十 五 年 七 月

Implementation of an Emergency Guiding System in Indoor 3D Environment by Wireless Sensor Networks

Student: Chia-Hung Tsai
Advisor: Prof. Yu-Chee Tseng

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

July 2006

Hsinchu, Taiwan

藉由無線感測器網路建立一室內3D環境下的緊急導引系統

學生：蔡佳宏　　指導教授：曾煜棋老師

國立交通大學資訊科學與工程學系(研究所)碩士班

摘要

近年來，無線感測器網路已在許多應用中被廣泛討論與使用，而在這篇文當中，我們建立了一套3D環境下的緊急逃生系統，當緊急事件發生時，能導引人們往安全的路逃生。而我們系統在平時有著監控環境的功能，而當偵測到緊急事件發生時，能夠動態地修正網路拓撲，來確保環境監控的可靠性，並能迅速地辨識出危險區域，既快且安全地導引人們避開危險區域，由安全的逃生路徑逃往出口。尤其在我們的設計中，特別把3D大樓的緊急逃生納入考慮。由模擬的結果可以顯示出，我們的演算法在緊急事件發生時，能以交換較少封包的方式，迅速地找到安全的路徑逃生。

關鍵字：住家安全，導引，無所不在的運算，無線通訊，無線感測器網路。

Implementation of an Emergency Guiding System in Indoor 3D Environments by Wireless Sensor Networks

Student: Chia-Hung Tsai Advisor: Prof.Yu-Chee Tseng

Department of Computer Science
National Chiao-Tung University

ABSTRACT

Recently, wireless sensor networks have been widely discussed in many applications. In this paper, we design a novel 3D emergency service that aims to guide people to safe places when emergencies happen. At normal time, the network is responsible for monitoring the environment. When emergency events are detected, the network can adaptively modify its topology to ensure transportation reliability, quickly identify hazardous regions that should be avoided, and find safe navigation paths that can lead people to exits. In particular, the structures of 3D buildings are taken into account in our design. Prototyping and simulation results show that our protocols can react to emergencies quickly at low message cost and can find safe paths to exits.

Keywords: home security, navigation, pervasive computing, wireless communication, wireless sensor network.

誌謝

踏入研究所這兩年，從指導教授及學長姊們身上學到很多寶貴的經驗與知識，也學到作研究的方法與態度，心中對這些提攜我的人充滿了感激，無線網路領域如此廣闊，很榮幸我能是研究者之一，也期盼我的這篇論文能提供些許貢獻。最後，再次感謝所有曾指導、協助與鼓勵我的教授、學長姊、同學及親愛的家人。

Contents

List of Figures

Chapter 1

Introduction

The recent progress of wireless communication and embedded micro-sensing MEMS technologies has made wireless sensor networks (WSN) more attractive. A lot of research works have been dedicated to WSN, including energy-efficient MAC protocols [11, 27], routing and transport protocols [8, 12], self-organizing schemes [17, 24], sensor deployment and coverage issues [13, 20], and localization schemes [4, 9, 21]. In the application side, habitat monitoring is explored in [2], the FireBug project aims to monitor wildfires [1], mobile object tracking is addressed in [5, 19, 25], and navigation applications are explored in [6, 10, 16, 18, 23, 26].

In this paper, we design a wireless sensor network in a 3D indoor environment for emergency guiding and monitoring service. Using wireless sensors to construct such a system has following benefits:

- Wireless sensors can be easily deployed.
- Wireless sensors can support surveillance applications at normal time.
- Wireless sensor can rapidly react to emergency events and report to a control host.
- Wireless sensors can locally exchange information to facilitate computing safer escaping paths.

Fig. 1.1 shows the system architecture. Sensor nodes are deployed floor by floor and are connected together by those at stairs. These sensors form a multi-hop ad hoc network. One node serves as the *sink* of the network, and it is connected to the *control host*, which can issue commands and config the network. To support guiding services, sensors are classified as *normal sensors*, *exit sensors*, and *stair sensors*. We

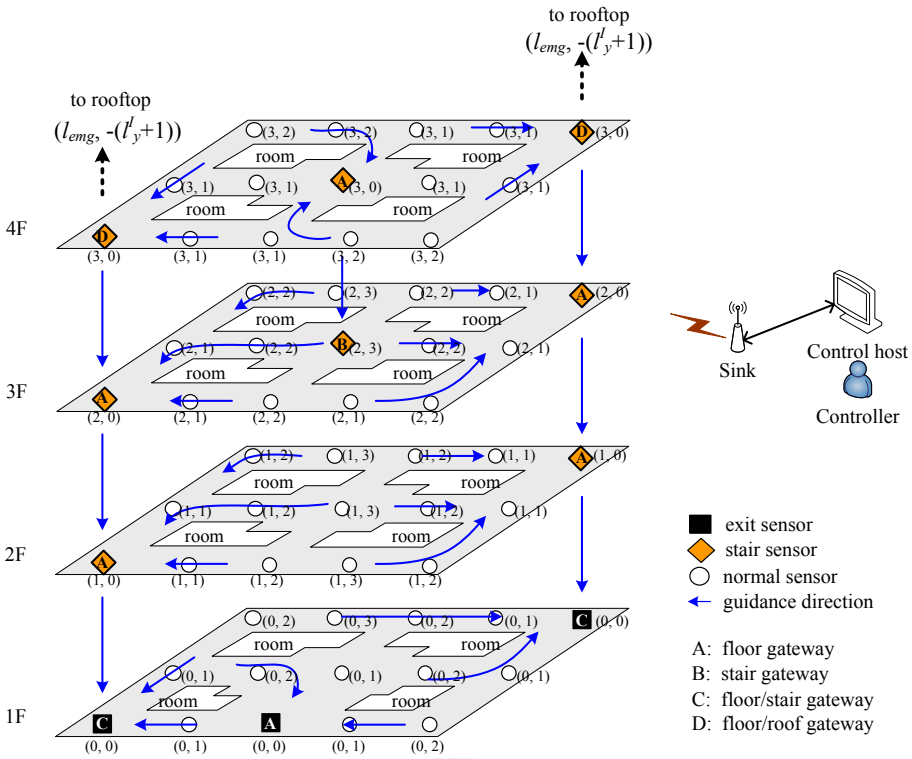


Figure 1.1: System architecture of our indoor 3D sensor network

will discuss how to design emergency guiding and monitoring services. The former addresses how to find escape paths leading to exits, in the event of emergencies, while the later addresses how to quickly report the status in the sensing field to the outside world.

Our approach relies on finding spanning trees rooted at the sink (to report data) and at exits (to guide people in the field). In particular, we will distinguish routing paths (for packets) from escape paths (for human). In this system, emergency events trigger the guiding service. Our protocol is distributed, and allows multiple emergency events and multiple exits coexisting in the sensing field. A concept called *hazardous region*, which people should avoid, is introduced.

The rest of this paper is organized as follows. Chapter 2 gives some related works about 2D navigation algorithms and network topology maintain mechanisms. Chapter 3 presents Guidance initialization procedures. Our emergency guiding algorithm is presented in Chapter 4. Chapter 5 reports our prototyping results. Chapter 6 presents some performance evaluation results. Chapter 7 concludes this paper.

Chapter 2

Related Works

This section reviews existing navigation. For navigation proposes, [6] shows how to guide a robot to a goal using sensors as signposts. Each sensor determines the direction to which the robot should move by computing a probability value for each neighbor. A higher probability means a shorter distance to the target.

Reference [18] has the same goal as our work. It is assumed that there are multiple emergency points (called *obstacles*) and one exit in the environment. The goal is to find a navigation path from each sensor to the exit without passing any obstacle. The concept of *artificial potential* is used. The exit will generate an *attractive potential*, which pulls sensors to the exit, and each obstacle will generate a *repulsive potential*, which pushes sensors away from it. Each sensor will calculate its potential value and tries to find a navigation path with the least total potential value. This result is also applied to robot navigation and distributed search and rescue in [10, 16, 23]. Although the algorithm in [18] is shown to be able to find a shorter and safer path from each sensor to the exit, it has the following drawbacks. First, it may incur high message overheads. Since the construction is rippled from the exit to other sensors, a minor change of potential nearby the exit may cause many sensors to recompute their potentials, thus causing a lot of message exchanges and even delays in making the navigation decision. Second, the algorithm has no concept of hazardous regions. With shortest-path routing, this algorithm may determine a path that is very close to the emergency location. Consider Fig. 2.1(a), where there are two exits, A and B. When an emergency is detected in C, according to [18], some users may be directed to B, which is undesirable because they will pass the hazardous region. Guiding users as in Fig. 2.1(b) will be more desirable because only users inside the hazardous region are directed to exit B.

Although these drawbacks are conquered in [26], both [18] and [26] deal with only

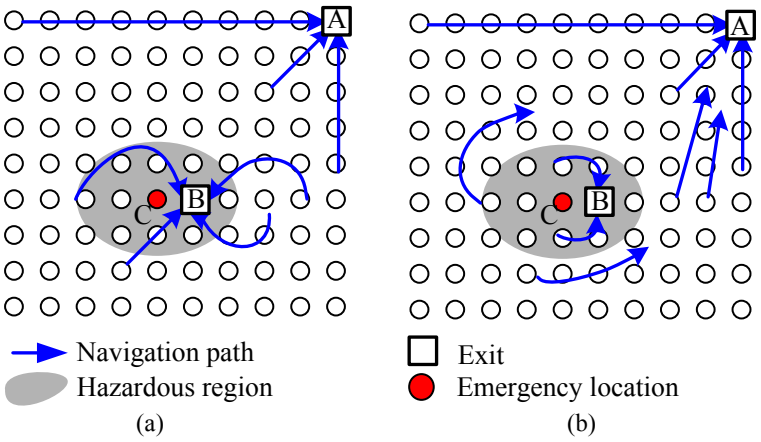


Figure 2.1: Some guidance scenarios when the hazardous region is defined as two hops from the emergency site.

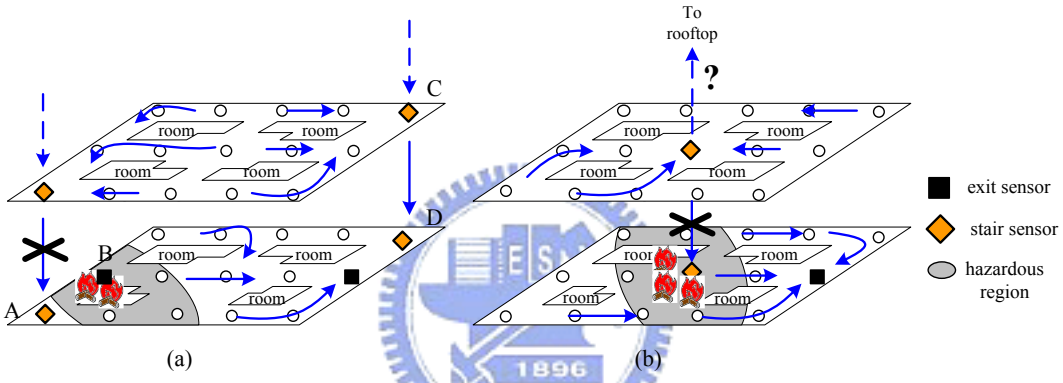


Figure 2.2: Two guiding scenarios that in 3D buildings.

2D sensing fields; they can not directly apply to 3D environments. Fig. 2.2 shows two scenarios. In Fig. 2.2(a), an emergency occurs on the ground floor. Sensors on the second floor do not know where the emergency is. Some sensors on the second floor may guide people to go through stair sensor A, which has a shorter escape way to exit B. However, going through stair sensors C and D would be safer. In Fig. 2.2(b), there is a stair connected to the roof. Again, when an emergency occurs near the stair on the ground floor, stair sensors on the second floor can not decide which way (to upstair or downstair) is more appropriate. The goal of this work is to consider 3D sensing fields, especially those inside a building, and address the emergency guiding applications in such environments.

Chapter 3

Guidance Initialization

We are given a set of sensors deployed in a building. Sensors' roles are designated at the deployment stage. Sensors located at the exits of the building are called *exit sensors*, and those located at stairs are called *stair sensors*. Otherwise, they are called *normal sensors*. One sensor is designated as the *sink*, which is connected to the control host.

From the network, we will construct a *communication graph* $G_c = (V, E_c)$ and a *guidance graph* $G_g = (V, E_g)$, where V is the set of sensors. Each edge $(u, v) \in E_c$ represents a communication link between u and $v \in V$, while each edge $(u, v) \in E_g$ represents a walking path between u and v. Note that a walking path is a physical route that human can pass. So E_g has to be constructed manually based on the floor plane of the building. Fig. 3.1 shows this concept. And for the purpose of monitoring, we will construct a minimum spanning tree at the beginning. This scheme will be started by the control host flooding an *INIT_N* packet. A node that receives an *INIT_N* selects a set of neighbors with the smallest hop count to the sink and then chooses a neighbor with the best signal quality as its parent. Then this node will rebroadcast the *INIT_N* if it changes its parent. Furthermore, in our design, sensors also report their neighbor information. As a result, the control host can know the G_c.

In the following, the guidance initialization procedure is presented.

The purpose of guidance initialization is to find escape paths leading to exits at normal times on the graph G_g. Guidance is not purely based on shortest-path routing. Instead, it gives higher precedence to stair sensors. We give three definitions here. On each non-ground floor, a sensor is called a *floor gateway* of that floor if it is a stair sensor and is connected to a downstairs sensor; on the ground floor, a sensor is a *floor gateway* if it is an exit sensor. Stairs are normally extended along

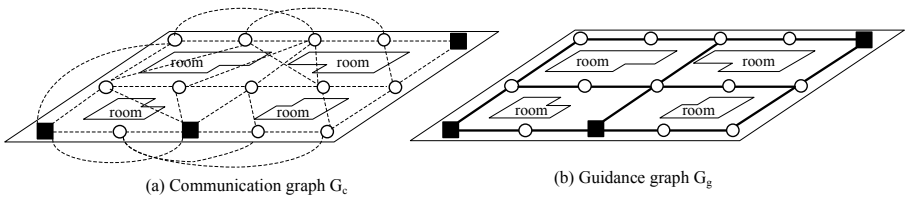


Figure 3.1: (a) Communication graph G_c and (b) guidance graph G_g.

several floors. For such continuous stairs, the stair sensor closest to the ground level is designated as the *stair gateway* and the stair connecting to roof is designated as the *roof gateway*. For example, in Fig. 1.1, type A, B, C, and D sensors are *floor gateways*, *stair gateways*, *floor/stair gateways*,and *floor/roof gateways* respectively. The configuration of sensors' status is done manually through the control host.

After planning G_g, we will compute for each sensor x a weight $w_x = (l_x, alt_x)$, where level l_x is the number of floors from x's current floor to the ground level and altitude alt_x is the hop distance from x to the nearest floor gateway on the same floor. When computing its guiding direction, a normal sensor can ignore the first tuple since it can only direct to neighbors on the same floor. But, the first tuple is important for stair sensors and is used to control stair sensors' guidance directions. Hence, we define the relationship between two weights as: a normal sensor considers $w_x > w_y$ if $alt_x > alt_y$ and a stair sensor considers $w_x > w_y$ if $(l_x > l_y)$ or $(l_x = l_y$ and $alt_x > alt_y)$. Guidance initialization starts by issuing *INIT_G* packets with weight $(0, 0)$ from exit sensors. When a sensor x receives an *INIT_G* packet with a smaller weight $w = (l, alt)$ than its current weight from its guidance neighbor, x executes the following steps:

1. If x is a stair sensor:
   (a) If the sender is a stair sensor or exit sensor:
      i. If x is a floor gateway, x records its weight as $(l + 1, 0)$.
      ii. Otherwise, x records its weight as $(l + 1, alt + 1)$.
   (b) If the sender is a normal sensor:
      i. If x is a floor gateway, x records its weight as $(l, 0)$.
      ii. Otherwise, x records its weight as $(l, alt + 1)$.
2. Otherwise, x records its weight as $(l, alt + 1)$.
3. Then x rebroadcasts the *INIT_G* packet with its ID and weight.

For example, Fig. 1.1 shows the initialization result of the given deployment. After the guidance initialization, each sensor will keep a guidance neighbor table, in which each entry records a neighbor's *ID*, *role*, *weight*, and *location*. One special rule is that for a *roof gateway* y, we will connect it an virtual sensor as its neighbor with weight $(l_{emg}, -(l_y^I + 1))$ if the gateway has a stair leading to the roof. The weight of virtual sensor is static and the details will be discussed in Section 4.

Chapter 4

Emergency Guiding Schemes

In the following, we present our distributed emergency guiding protocol. Our goal is to guide people in a building to escape safely when emergency happens. When a sensor detects an emergency event, this sensor and the sensors surrounding it will form a *hazardous region* by raising their weights. Sensors will locally choose their guiding directions according to their roles. However, when a sensor has a local minimum weight, *partial link reversal* is used to solve this problem. Our design focuses on quick convergence and can avoid guiding people through hazardous regions. After leading people to stairs, floor gateways will direct people to go downstairs if there is no hazards downstairs. Otherwise, floor gateways will force people to go to other stairs. If there is no suitable ways to go downstairs, our protocol will guide people to the rooftop to wait for help. Below, we first introduce some notations:

- D: a constant such that a sensor is considered within a *hazardous region* if its hop count to any emergency location in G_g is less than or equal to this value.
- w_{emg}: the weight (l_{emg}, alt_{emg}) to be assigned to a sensor that detects an emergency event, where l_{emg} and alt_{emg} are large constants.
- w_i^I: the weight (l_i^I, alt_i^I) of sensor i after the guidance initialization.
- $e_{i,j}$: the hop count from an emergency sensor i to a sensor j in G_g.
- *EMG* packet: the emergency notification packet, which has five fields: (1) event sequence number, (2) ID of the sensor which finds the emergency event, (3) sender's ID, (4) weight of the sender, and (5) hop count from the sender to the emergency sensor.

Intuitively, the *altitude* tuple in a weight reflects the dangerous degree of the corresponding sensor, while the *level* tuple is to determine whether the corresponding stairs are proper escaping paths. In this paper, we will assign level $l_{emg} - 1$ to sensors that are located in hazardous regions. So an upstair sensor which finds its downstair sensor with a level larger than $l_{emg} - 1$ will avoid guiding people to the latter if possible. However, if the upstair sensor can not find a proper escaping path, it will keep raising its weight until a neighbor with a smaller weight is found.

Suppose that a sensor x detects an emergency. It will set its weight to w_{emg} and immediately broadcast an *EMG(seq, x, x, w_{emg}, 0)* packet. The packet will be flooded in the network G_g for sensors to re-compute their guiding directions. When a sensor y in G_g receives from a sensor z an *EMG(seq, x, z, w_z, h)* packet originated from x, y first updates z's weight as w_z in its neighbor table. Then y performs the following steps.

1. y judges if this is a new emergency by checking the tuple (seq, x), and updates $e_{x,y}$ as follows.

   (a) If this is a new emergency event to y, y records this event and sets $e_{x,y}$ to $h + 1$.

   (b) Otherwise, y checks if $h + 1 < e_{x,y}$. If so, y changes $e_{x,y}$ to $h + 1$.

   If y is a normal sensor and z is a stair sensor, y will set $l_y = l_z$ to prevent y and z from directing to each other in step 5 when y has a larger altitude but a smaller level than z.

2. This step is for sensors located in hazardous regions, where a sensor y is said to be located in the hazardous region formed by x if $e_{x,y} \leq D$. However, if y a stair sensor, we consider it as in a hazardous region if its downstair sensor is in a hazardous region.

   (a) For a normal sensor y, if $e_{x,y} \leq D$, y sets $l_y = l_{emg} - 1$ and sets

   $$alt_y = \max\{alt_y, alt_{emg} \times \frac{1}{e_{x,y}^2} + alt_y^I\}. \tag{4.1}$$

   In our design, the altitude of a sensor inside a hazardous region is increased by an amount inversely proportionate to the square of its distance to the emergency location. The value alt_y^I is included so as to reflect y's distance to its nearest exit. The max function is to take into

account that y may be located within multiple hazardous regions and thus receive multiple *EMG* packets from different sources, in which case the largest altitude is adopted.

(b) For a stair sensor y, y adjusts its weight by the following rules:

i. If $l_z^I > l_y^I$ and $e_{x,y} \leq D$, y considers itself as in a hazardous region and sets its $l_y = l_{emg} - 1$ and its alt_y according to Eq. (4.1).

ii. If $l_z^I = l_y^I$ and $e_{x,y} \leq D$, y adjusts its altitude according to Eq. (4.1). However, y further resets $l_y = l_{emg}$ to prevent upstair sensors from directing to itself.

iii. If $l_z^I < l_y^I$ and $e_{x,y} - 1 \leq D$, y considers that it is located in a hazardous region because its downstair sensor is located in a hazardous region. Then y sets $e_{x,y} = h$, $l_y = (l_{emg} - 1)$, and $alt_y = alt_z$.

3. Then y checks if it becomes local minimum. A normal sensor y is a local minimum if for each neighbor x of y, $alt_x > alt_y$; a stair sensor y is a local minimum if for each neighbor x of y, $w_x > w_y$.

(a) If a normal sensor y is a local minimum, it adjusts its altitude by

$$alt_y = STA(alt(n_y)) \times \frac{1}{|n_y|} + \min\{alt_{n_y}\} + \delta, \tag{4.2}$$

where n_y is the set of neighbors of y, $STA(alt(n_y))$ is the standard deviation of the altitudes of sensors in n_y, and δ is a small constant. The basic idea of using standard deviation is for quick response to emergencies. When altitudes of sensors in n_y vary significantly, it is likely that y is near a hazardous region. Then it should increase its altitude more quickly to avoid becoming a local minimum again. The constant δ is to guarantee convergency. Its value should be carefully chosen because a large δ may easily guide sensors to cross hazardous regions. On the other hand, a small δ may incur high message cost although it may help find safer paths. The reciprocal of $|n_y|$ is to reflect y's potential choices to select escape directions. A sensor that has less choices will increase its altitude faster to get away from local minimum situation. These designs will speed up the convergence speed.

(b) If a stair sensor y is a local minimum, it adjusts its weight based on its current l_y. Specifically, a stair sensor can guide people to go upstairs,

downstairs, or go through other floor gateways on the same floor. In our design, a stair sensor will set its initial direction to downstairs. If there is no proper way in its downstairs, it will try other floor gateways on the same floor or stair sensors to upstairs. Roof gateways are only used when there are no proper ways to go downstairs. Stair sensors that are not connected to the roof will reverse their guidance directions to downstairs, in which case people will be led to hazardous regions, which is sometimes inevitable. To achieve these goals, the following steps are taken.

i. If $l_y = l_y^I$, y sets $l_y = l_{emg} - 1$ and alt_y by Eq. (4.2). In this case, y can direct to upstair or to the neighbors on the same floor since y's downstair sensor is potentially surrounded by hazards.

ii. If $l_y = l_{emg} - 1$, y sets $l_y = l_{emg}$ and judges if it is a roof gateway or it has an upstair sensor with level smaller than l_{emg}.

   A. If yes, y sets $alt_y = -l_y^I$. Recall that a roof gateway has a virtual sensor with initial weight $(l_{emg}, -(I_y^l + 1))$. Hence, if y is a roof gateway, it will guide people to the roof. After this adjustment, y can direct to upstairs if it has an upstair sensor since the upstair sensor's level is smaller than l_{emg}.

   B. Otherwise, y sets alt_y by Eq. (4.2).

iii. If $l_y = l_{emg}$, y keeps its level unchanged and sets alt_y according to Eq. (4.2) to find possible escaping paths with smaller altitudes, which implies less dangerous.

4. y broadcasts an *EMG(seq, x, y, w_y, $e_{x,y}$)* packet if : (i) this is a new emergency packet heard by y, or (ii) sensor y has changed w_y or $e_{x,y}$ in the previous steps. Also, since broadcast may suffer from loss, y will rebroadcast the same *EMG* packet periodically, until it enters this step again and changes the content of the *EMG*.

5. Finally, sensor y chooses its escape direction as follows. If y is a normal sensor, y directs to the neighbor with lowest altitude. Otherwise, y (as a stair sensor), directs to the neighbor with the lowest weight.

We remark that the above step 3.a adopts the concept of *partial reversal* in [22] to adjust local minimum nodes' altitudes. We do not adopt the *full reversal* approach in our design because it may unnecessarily guide users in a non-hazardous

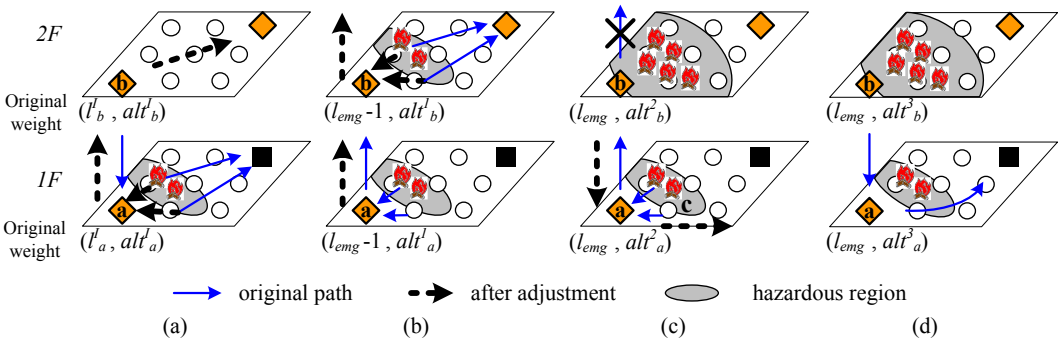


Figure 4.1: The weight adjustments of local minimal stair sensors in step 3(b).

region to pass through a hazardous region. Using partial reversal can help guide users to route around a hazardous region. An example of weight adjustment for local minimal stair sensors is shown in Fig. 4.1. In Fig. 4.1(a), an emergency occurs on the ground floor and a hazardous region blocks the escape directions of sensor a. In this case, stair sensor a will become a local minimal node. Then a executes step 3.b.i to get rid of such situation and point to b, which has the smallest weight among a's neighbor. Sensor b also applies the same procedure as a and finds an escaping path to the other floor gateway on the second floor. Assume that another emergency occurs on the second floor as shown in Fig. 4.1(b). Sensor b will become a local minimal again because its guiding path is blocked by a new hazardous region. It can further direct to upstairs if its satisfies the condition in step 3.b.ii. After b's adjustment, a will become a local minimal and execute step 3.b.ii.B. In our design, altitudes of the sensors on the boundary of a hazardous region is large. After the adjustment, a's altitude can not be higher than the altitudes of those sensors on the boundary of the hazardous region with high probability. As a result, a will direct to b. In case that b can not go to upstairs and the emergency area is enlarged (shown in Fig. 4.1(c)), going through the hazardous regions on the second floor is improper. Then b and a will apply step 3.b.iii to increase their altitudes. A path through c can be found because c is farther from the emergency than other sensors. Finally, Fig. 4.1(d) shows the guidance result, which may not be 100% safe but is inevitable in this case.

We claim that each sensor can find a guidance direction in a finite number of steps if sensors can successfully exchange *EMG* packets. A sensor becomes a local minimum when it has no escape paths. Since δ is a positive constant, the protocol has a progress property in the sense that the number of normal sensors which have no escape paths on a floor will reduce. Local minimum stair sensors adjust weights

according to their levels. Stair sensors normally choose to go downstairs first. If this fails, they will try to find other floor gateways on the same floor or go to upstairs. However, if the roof is not reachable, the stair sensors will reverse their directions to downstairs gradually from the roof by raising their altitudes. This guarantees convergency of our protocol. Moreover, when emergencies occur, each sensor will periodically broadcast *EMG* packets to facilitate the correctness of guidance.

Chapter 5

Prototyping Results

We have implemented our system using MICAz motes and MTS310 sensors on TinyOS. Fig. 5.1 shows the protocol stacks of our system. Our system can be divided into a "user part" and a "sensor part". In the user part, we implement two application-level user interfaces:

- Graphical user interface (GUI): It is to be run in a personal computer connected to a MIB510 programming board. It provides a JAVA interface for users to deploy the sensor network, to initialize the network, and to query the statuses of sensors. Fig. 5.2 shows our user interface. Users can perform the following operations on this GUI.
  - Deployment: The system manager first plans sensors' locations on the "building plan panel". According to the plan, the manager then physically deploys sensors at the corresponding locations. Then the system

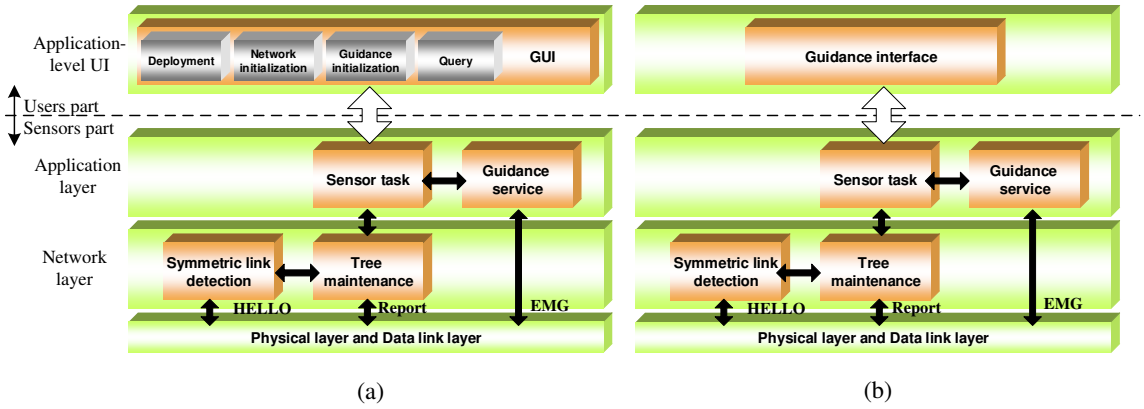


Figure 5.1: Protocol stacks for (a) sink node and (b) sensor node.

manager can design guidance links between sensors and get a guidance graph G_g. The system will also establish a mapping between sensors' IDs and their locations.

  - Network initialization: The connectivity between sensors will be automatically determined by *HELLO* packets received at each sensor. This will generate the communication graph G_c. Network initialization is issued by the "network initialization button". After forming the reporting tree, sensors will report their readings. In our current implementation, sensors will report temperature and light degrees.
  - Guidance initialization: Guidance initialization is issued by the "guidance initialization button". After pressing this button, the control host will disseminate the connectivity information of the guidance graph G_g throughout the network. Then the control host will inform exit sensors to issue *INIT_G* packets to compute the initial guiding direction for each sensor.
  - Query: Users can query each sensor's status by pointing at the sensor on the building plan panel. In response, the monitor panel will show the sensed data, communication neighbors, guidance neighbors, guidance direction, and current weight of the sensor.
  - Commanding: In our design, the controller can also change the sensor's reporting rates to adapt to circumstances needs. In addition, we also provide some other commands such as system reset.

- Guidance interface: We have implemented a LED guidance panel to display the guidance results of sensors. The LED panel is attached to the connector of the MTS310. When a sensor determines its guidance direction, it will send a control signal to the panel. In our current implementation, this panel can display up to six guidance directions as shown in Fig. 5.3.

For the sensor part, the protocol is divided into an application layer and a network layer. The former has three components.

- Guidance service: This component is the core part of our system. The algorithm in Chapter 4 is implemented. The guiding algorithm is triggered when an emergency happens. First, this component will update the information obtained from the *EMG* packet. According to our algorithm, sensors will judge if

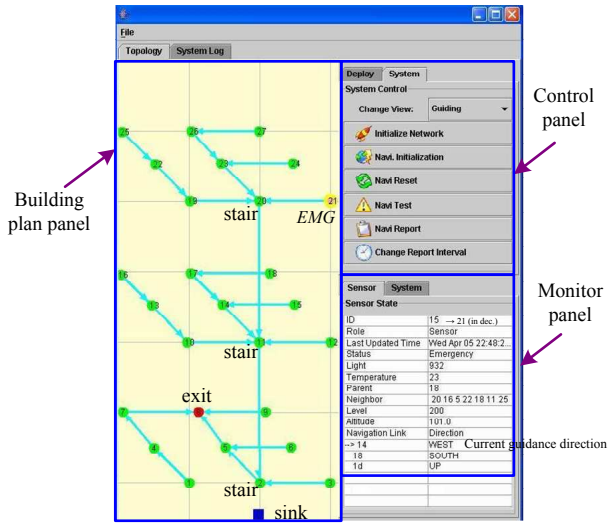


Figure 5.2: Our JAVA graphical user interface.

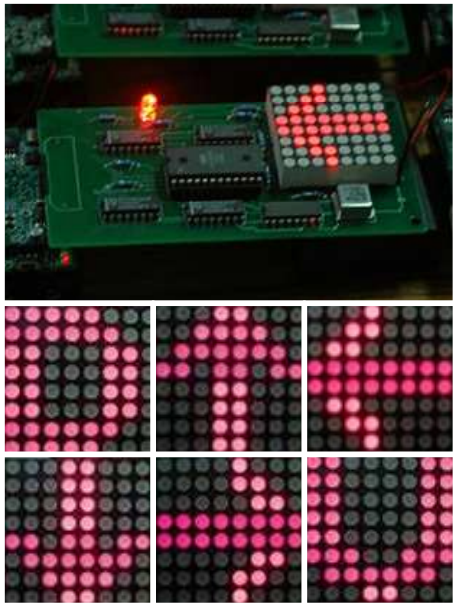


Figure 5.3: Our LED guidance panel (where "D" = downstairs and "U"= upstairs).

Bytes: 0	1	2	3-7	8
Sequence number	EMG Source ID	Sender ID	Weight	Hop count to emergency

Figure 5.4: Format of the EMG packet.

Bytes	Description
3	Level
4-7	Altitude

Figure 5.5: Format of the weight subfield in the EMG packet.

they are located in hazardous regions and if they have became local minimum onces. Finally, sensors determine whether to re-broadcast *EMG* packets, and then find out proper escape directions. The *EMG* packet format is given in Fig. 5.4 and Fig. 5.5. The detail flow chart of this component is shown on Fig. 5.6.

- Sensor task: At normal time, sensors periodically sense environmental data and report to the control host. When a sensor detects an emergency event, this component will trigger the tree reconstruction and guidance service components.

In our implementation, the network layer has two components:

- Symmetric link detection: When booting up, this component will be executed first. The main goal of this component is to maintain sensors' neighbor tables. The neighbor table format is shown in Fig. 5.7. In our system, sensors regularly broadcast *HELLO* packets. After sending out a *HELLO* packets, a sensor will trigger an *adjustLink* procedure to maintain its communication links. Inside the procedure, the sensor will check if it has not received any *ACK* packet from any neighbor for a period of time *expire_t*. If so, this neighbor will be pruned. On the other hand, on receiving a *HELLO* packet, a sensor will check the link quality. If the RSSI strength is over a threshold, it will reply an *ACK* packet. When the *HELLO* sender receives an *ACK*, it will trigger the *maintainLink* procedure. If the sender of the *ACK* is not in its neighbor table, the procedure will add this node into its neighbor table. Otherwise, this sender is an existing neighbor, and its timer *expire_t* will be reset. Fig. 5.8 outlines the above procedure. Packet formats of *HELLO* and *ACK* are in Fig. 5.9.

- Tree maintenance: This component maintains sensors' parents information. Each sensor dynamically chooses a node that has the least hop count to the

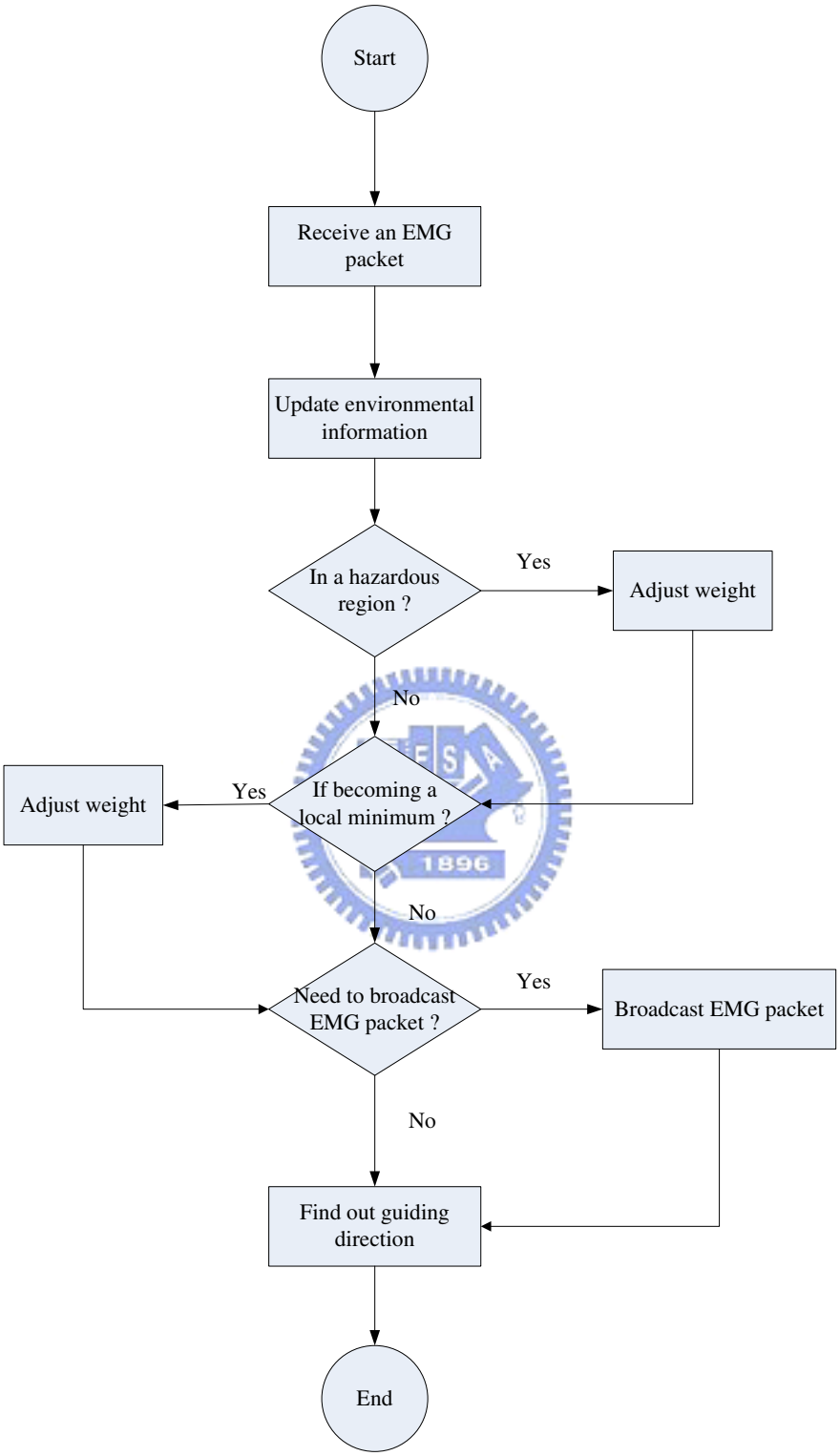


Figure 5.6: The flow chart of our guidance service.

Neighbor Table	Description
Neighbor ID	The sensor id of the communication neighbor
Expired_T	Expiration time of the link if no HELLO is received

Figure 5.7: The subfield of the neighbor table.

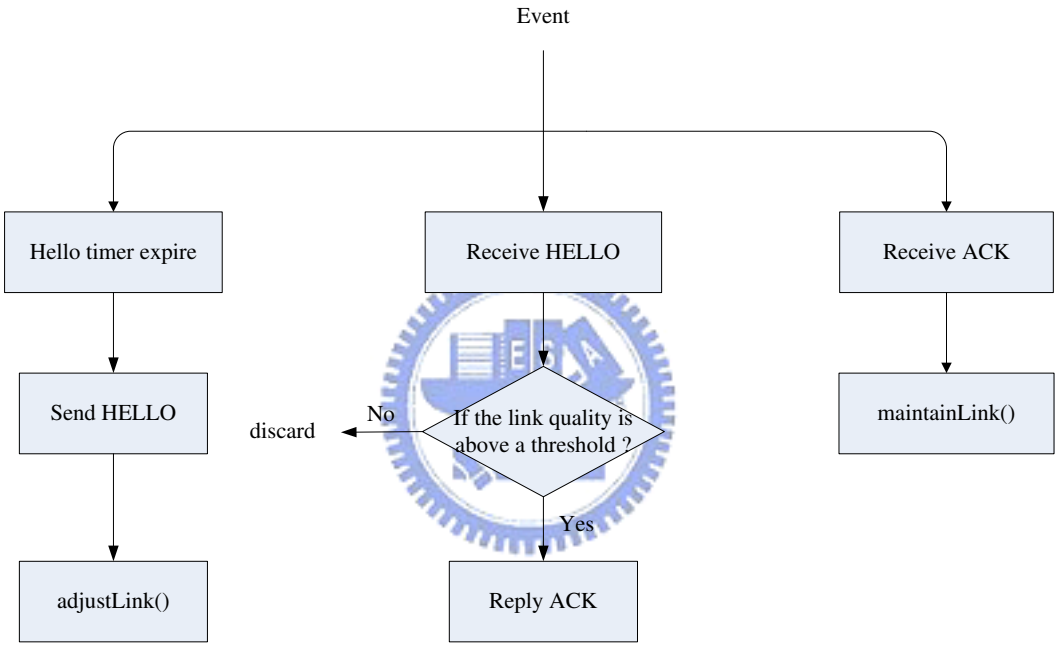


Figure 5.8: The flow chart of symmetric link detection.

Bytes: 0	1	2
Type	Sender ID	Hop count to sink

(a)

Type value	Description
0	HELLO
1	ACK

(b)

Figure 5.9: (a) The format of HELLO/Ack packets. (b) Definitions of the type subfield.

sink as its parent. If more than one neighbor satisfies the condition, the one with the best signal strength is chosen. The reporting tree is used to control packets transmission and packets reception. The implementation details are in Appendix A.

Chapter 6

Performance Evaluation

To investigate the performance of the proposed algorithm, we have conducted real experimental tests and simulation evaluation. The results are presented in this section.

6.1 Experimented Results

We have implemented our guiding protocol in TinyOS using B-MAC. We use the Micaz wireless sensor nodes with MTS310 sensor board to conduct our experiments. A virtual 2-store building as shown in Fig. 6.1 is built. Each floor is a 4x3 grid network. Fig. 6.2 shows some guiding results. Each normal sensor has guidance neighbors on its east, west, north, and south sides and each stair sensor has two addition neighbors, upstairs and downstairs. The default values of alt_{emg}, l_{emg}, δ, and D are set to 200, 100, 0.3 and 1, respectively. To emulate multi-hop communication, we also reduce the transmission power of sensors. This may result in unstable communication links and loss of packets. However, we find that our algorithms can handle these situations well.

In Fig. 6.2(a), an emergency occurs at the exit sensor on the ground floor. The stair sensor in the upper-right corner on the second floor thus guides people to the lower-left stair sensor and then to the ground floor. Fig. 6.2(b) shows a case with two emergency events. Sensors on the right-hand side will guide people to route around the boundaries of hazardous regions to the exit. In this case, since there is only one exit, sensors on the second floor can only guide people to the only stair sensor on the corner. Fig. 6.2(c) shows a similar case except that there is roof stair. In this case, people on the second floor will be guided to the roof sensor.

We have also built a 27-node prototype in a virtual 3-store home. Each floor is

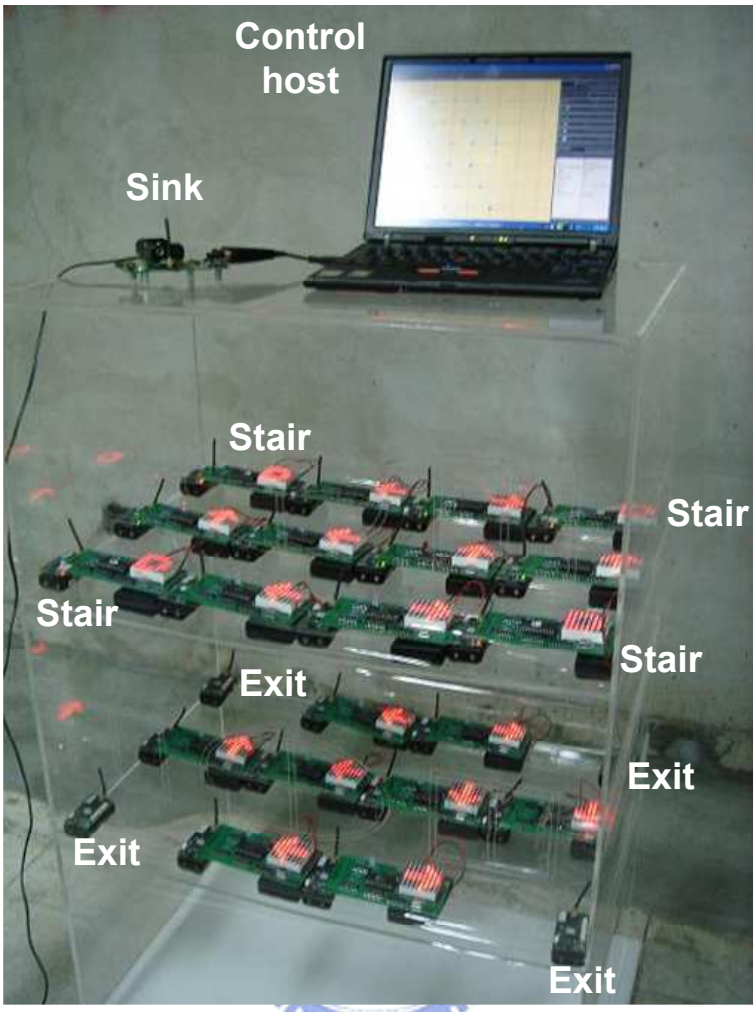


Figure 6.1: A virtual 2-store building.

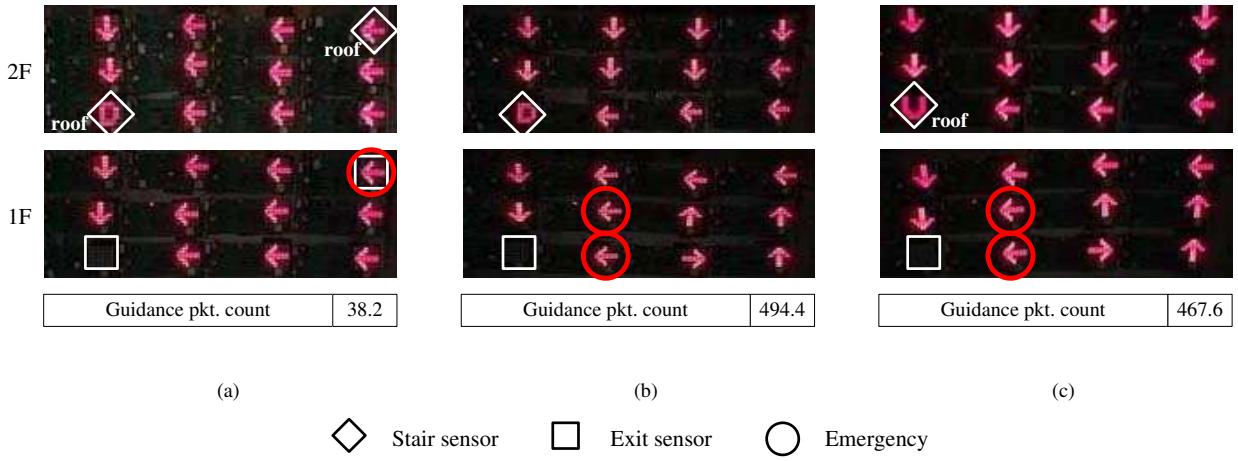


Figure 6.2: Some guiding results in a 2-store building.

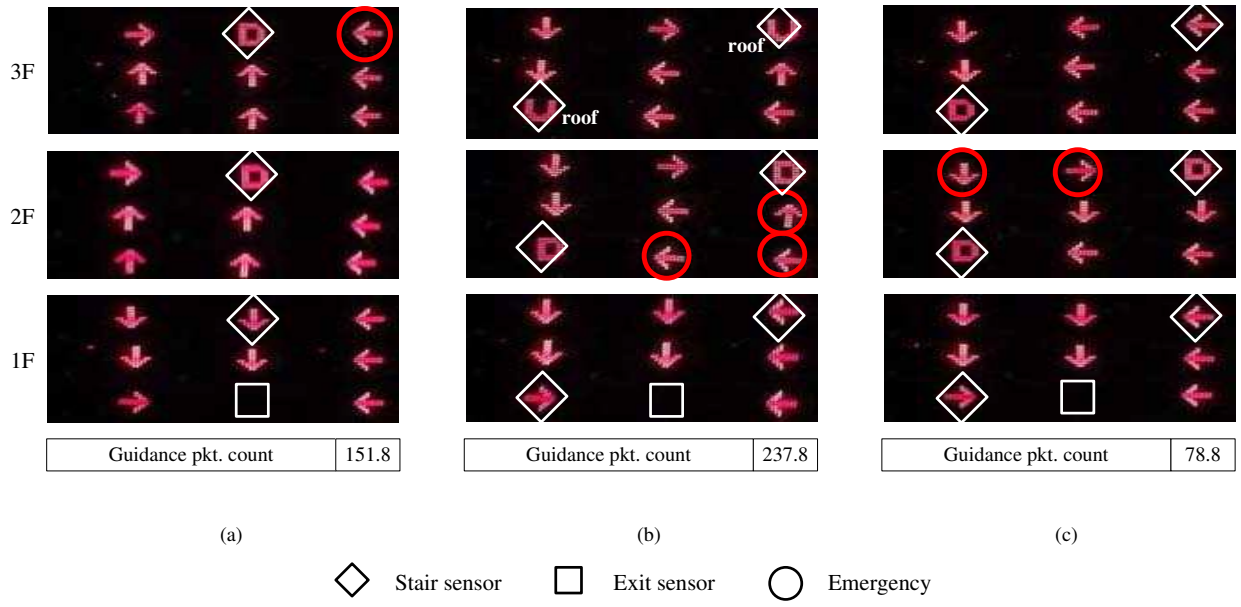


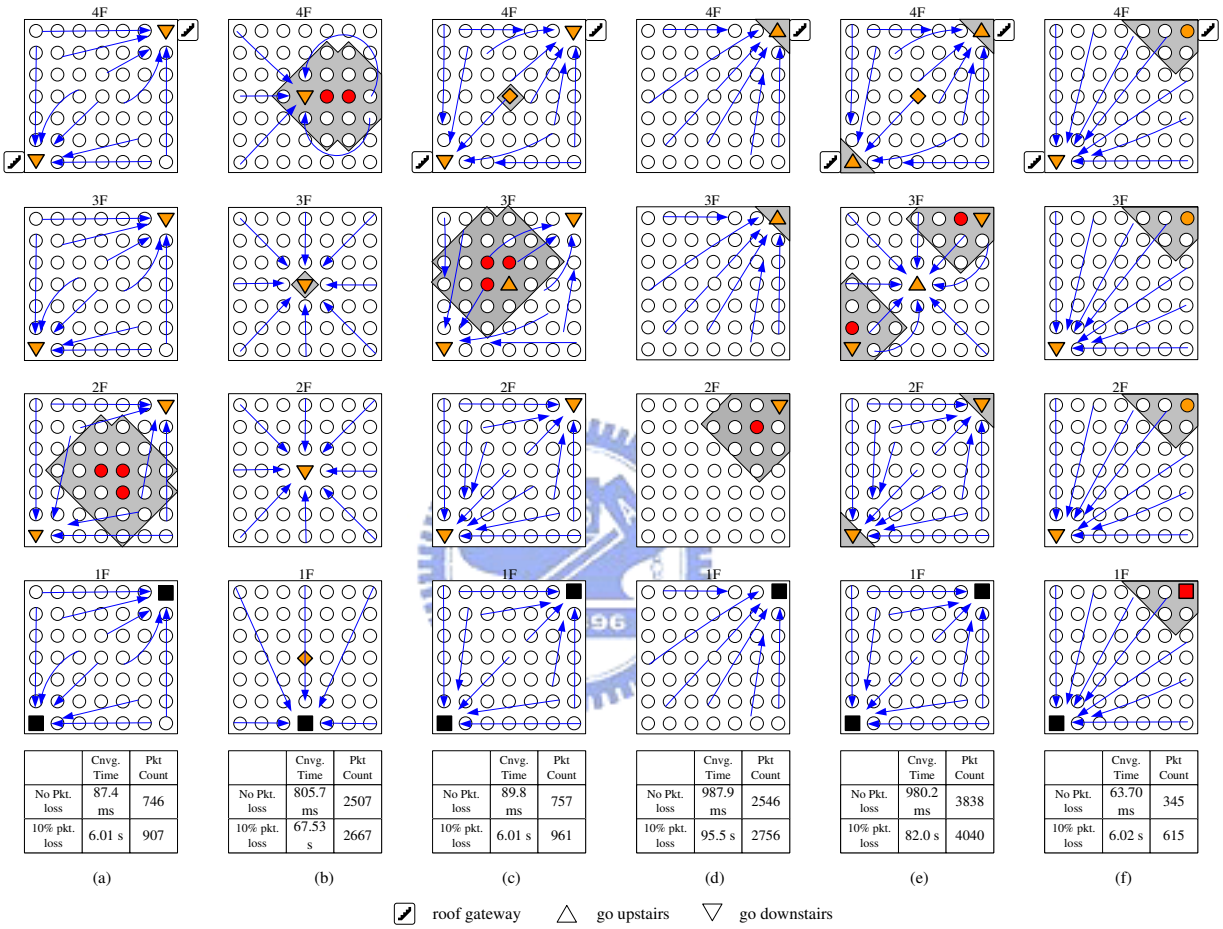
Figure 6.3: Some guiding results in a 3-store building.

a 3x3 grid network. Fig. 6.3(a) shows a case that a sensor near the stair sensor on the third floor detects an emergency event. Since there is no way to roof, all sensors will guide people to the exit on the ground floor. In Fig. 6.3(b), three emergencies are detected. Since the second floor is almost all covered by emergencies, people on the third floor will be guided to the roof. In Fig. 6.3(c), the stair sensor in the upper-right corner on the third floor will not guide people to downstairs because its downstairs sensor is located in a hazardous region. This stair sensor will lead people to the other stair sensor on the third floor.

In Fig. 6.2 and Fig. 6.3 we show the packet counts of all scenarios. These counts do not include the packets for periodical reports. Note that among all cases, the scenarios which need to force people to the roof, such as those in Fig. 6.2(b) and Fig. 6.2(c), have the highest costs.

6.2 Simulation Results

Different structures of buildings should be taken into consideration in measurements. However, this is difficult to achieve by real experiments. Furthermore, large-scale experiments are infeasible. Below, we present our simulation results to test our algorithm under more complex scenarios. Unslotted CSMA/CA protocol following



	Cnvg. Time	Pkt Count
No Pkt. loss	87.4 ms	746
10% pkt. loss	6.01 s	907

(a)

	Cnvg. Time	Pkt Count
No Pkt. loss	805.7 ms	2507
10% pkt. loss	67.53 s	2667

(b)

	Cnvg. Time	Pkt Count
No Pkt. loss	89.8 ms	757
10% pkt. loss	6.01 s	961

(c)

	Cnvg. Time	Pkt Count
No Pkt. loss	987.9 ms	2546
10% pkt. loss	95.5 s	2756

(d)

	Cnvg. Time	Pkt Count
No Pkt. loss	980.2 ms	3838
10% pkt. loss	82.0 s	4040

(e)

	Cnvg. Time	Pkt Count
No Pkt. loss	63.70 ms	345
10% pkt. loss	6.02 s	615

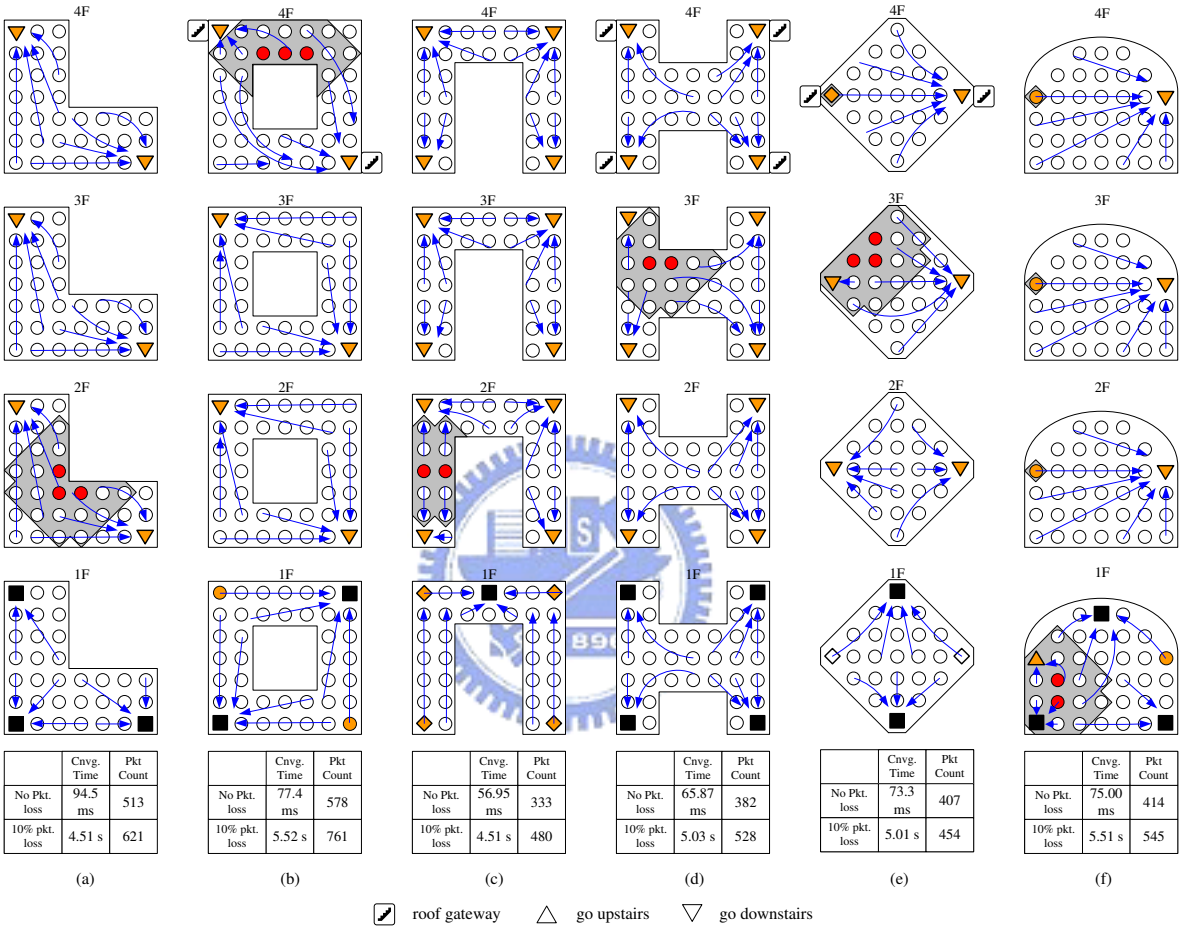
(f)

Figure 6.4: Some guidance results in 4-store buildings.

the IEEE 802.15.4 [14] is used in the simulation. The PHY rate of 250 *kbps* is assumed. The values alt_{emg}, l_{emg}, and δ are the same as above experiments except that D is set to 2.

We consider a 4-store building and each floor is a 7x7 grid network. Fig. 6.4(a) shows a case where some emergencies occur near the center of the second floor and how sensors on the second floor guide people to avoid hazardous regions. Fig. 6.4(b) simulates a building with no roof stair and only one stairs. Emergency events are detected nearby the stair on fourth floor. Since the stair sensor on fourth floor realizes that there is no rooftop, it guides people to downstairs. Fig. 6.4(c) shows a case where some emergencies occur near the stair sensor on the third floor. Sensors on the forth floor will guide people to the ground floor instead of to the roof. Note that the stair sensor on the center of the third floor will direct people to upstairs. This is reasonable because people currently in the staircase between the third and the forth floor should be guided to upstairs. Fig. 6.4(d) is a case with only one stairs and has one roof top. As there are emergencies nearby the floor gateway on the second floor, the stair sensor on the fourth floor will adjust its weight until its weight becomes larger than the virtual sensor. As a result, the sensors on the fourth floor will direct people to the roof top and then sensors on the third floor will also guide people to upstairs. Fig. 6.4(e) is a case with roof stairs. As there are emergencies nearby the two stair sensors on the third floor, stair sensors on fourth floor will adjust their weights until their altitudes become larger than the virtual sensor. After this adjustment, these stair sensors will direct people to roof stairs since there is no safer path to the ground. Since the floor gateways on the third floor are all in the hazardous regions, they will direct people to upstairs through the middle stair gateway. Fig. 6.4(f) is a case that an exit senses an emergency occurred. All the upstairs sensors will direct people downstairs through the floor gateways that are not in hazardous regions.

In Fig. 6.5, we simulate our algorithm in 4-store buildings of various kinds of shapes. Fig. 6.5(a)-(d) are similar cases but are for different shapes of buildings. These results indicate that our guidance protocol is suitable for various kinds of building architectures. Fig. 6.5(e) shows a case where some emergencies occur nearby a floor gateway on the third floor. Since the left-hand side floor gateways on the third and the fourth floors are all in hazardous regions, the sensor on these two floors will direct people downstairs through the other floor gateways. The sensors on the first and second floors are in the dangerous regions, so they will guide people to exits using the shortest paths. Fig. 6.5(f) is a case similar with Fig. 6.4(f), where



	Cnvg. Time	Pkt Count
No Pkt. loss	94.5 ms	513
10% pkt. loss	4.51 s	621

(a)

	Cnvg. Time	Pkt Count
No Pkt. loss	77.4 ms	578
10% pkt. loss	5.52 s	761

(b)

	Cnvg. Time	Pkt Count
No Pkt. loss	56.95 ms	333
10% pkt. loss	4.51 s	480

(c)

	Cnvg. Time	Pkt Count
No Pkt. loss	65.87 ms	382
10% pkt. loss	5.03 s	528

(d)

	Cnvg. Time	Pkt Count
No Pkt. loss	73.3 ms	407
10% pkt. loss	5.01 s	454

(e)

	Cnvg. Time	Pkt Count
No Pkt. loss	75.00 ms	414
10% pkt. loss	5.51 s	545

(f)

Figure 6.5: Some guidance results in 4-store buildings of various shapes of architecture.

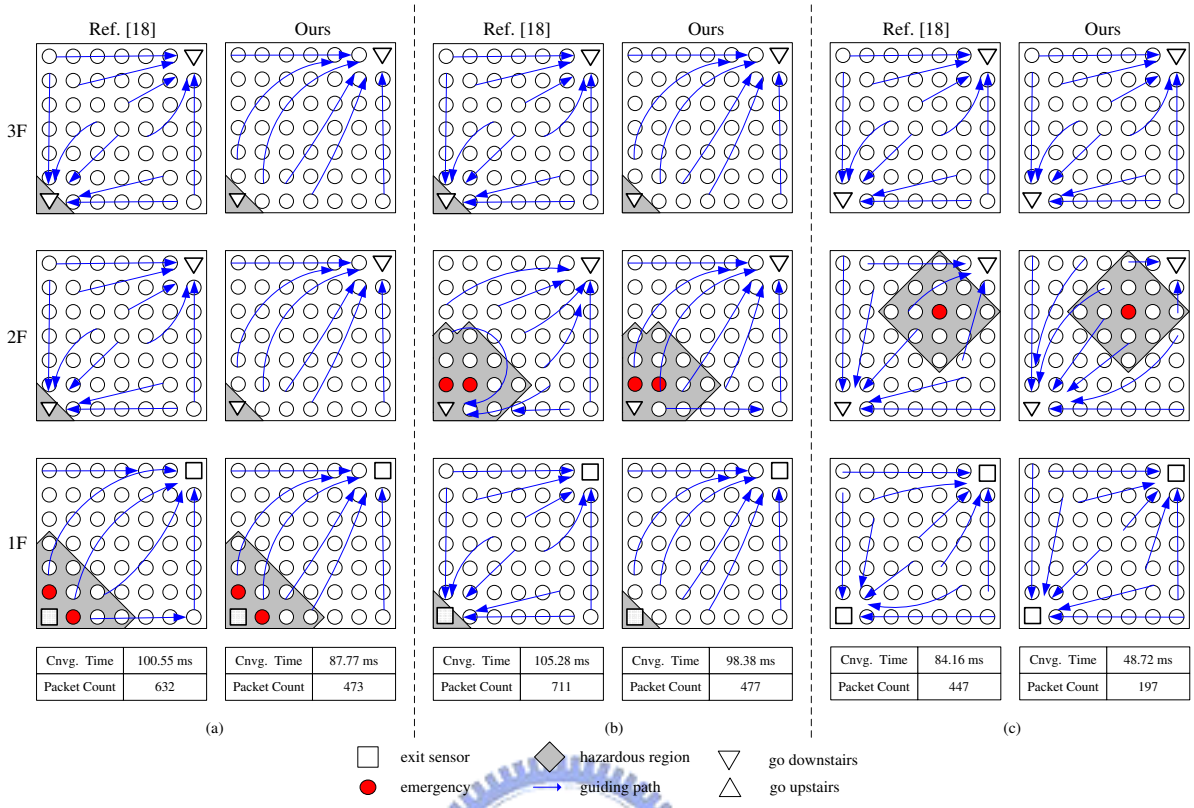


Figure 6.6: Comparison of escaping paths, convergence times, and message overheads against [18].

there is an emergency on one of the exits. We should pay attention to the floor gateway in the dangerous region on the first floor. Since sensors around this stair gateway are all in the dangerous region, this stair gateway can only guide people to upstairs.

In Fig. 6.4 and Fig. 6.5, the convergence time and packet counts are obtained assuming two cases: no packet loss and a 10% packet loss rate. The loss of *EMG* packets may lead to unstable guidance results, which will be connected by our periodical reporting scheme. Sensors will broadcast *EMG* packets every 0.5 seconds when emergencies occur. The convergence time, as well as packet count, is measured by the time the last sensor updating its guidance direction.

In Fig. 6.6, we compare our escaping paths, convergence times, and message overheads against those obtained by [18]. In the comparison, we only simulate the case of no packet loss. In Fig. 6.6(a), sensors near the left corner exit on the ground floor detect an emergency. The scheme in [18] will pull some sensors on the second and the third floors to go downstairs, which is more dangerous. This is

because the scheme in [18] does not implement the concept of dangerous regions. On the contrary, ours will lead people away from such dangerous regions. Fig. 6.6(b) illustrates another case where sensors nearby the stairs on the second floor detect an emergency. Again, because shorter paths are preferred, the algorithm in [18] will guide some people to pass the hazardous region. Fig. 6.6(c) shows a case where all floor gateways are not in dangerous regions, therefore sensors on the ground and third floors will guide people using the shortest path to the floor gateway. Nevertheless, on the second floor, the algorithm in [18] will still guide some people to across the hazardous region. Besides providing safer escape paths, our scheme also outperforms [18] in packet counts and convergence time.

Chapter 7

Conclusions and Future Work

In this paper, we have presented an emergency guiding for indoor 3D environments on wireless sensor networks. The proposed emergency guidance scheme can quickly converge and find safe guidance paths to exits when emergencies occur. We also implement our results on Micaz motes in a virtual building. Simulation results have also been obtained on buildings of various shapes with different combinations of hazardous regions. Comparisons have been made against [18] have demonstrated the advantages of our scheme. In our current design, the hazardous region is defined by the numbers of hops in G_g and the altitudes of nodes in hazardous regions are adjusted by a static function. In fact, the definition of hazardous regions and altitude adjustments can be application- or scenario-dependent. For example, sensors detecting temperatures of $100°C$ and $70°$ can both claim detecting emergencies. But altitude adjustment functions can be designed by taking the sensed temperatures into account.

Appendix A

Tree Maintenance Procedure

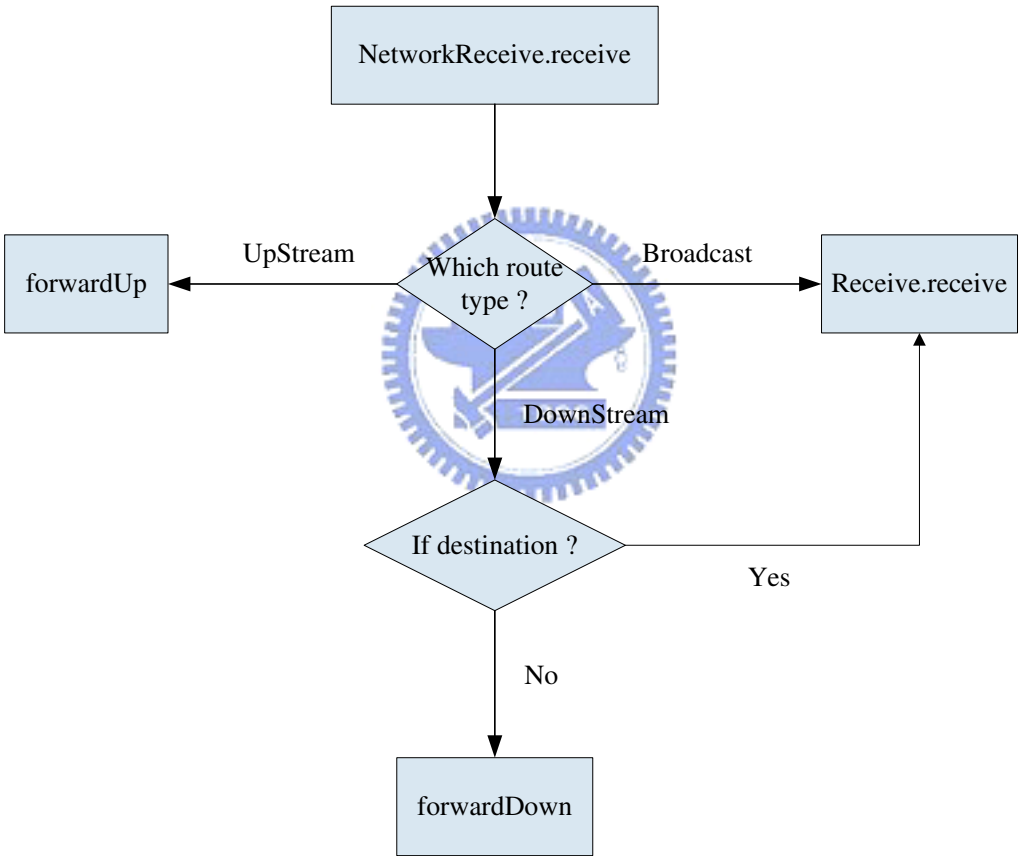


Figure A.1: The flow chart of tree maintenance.

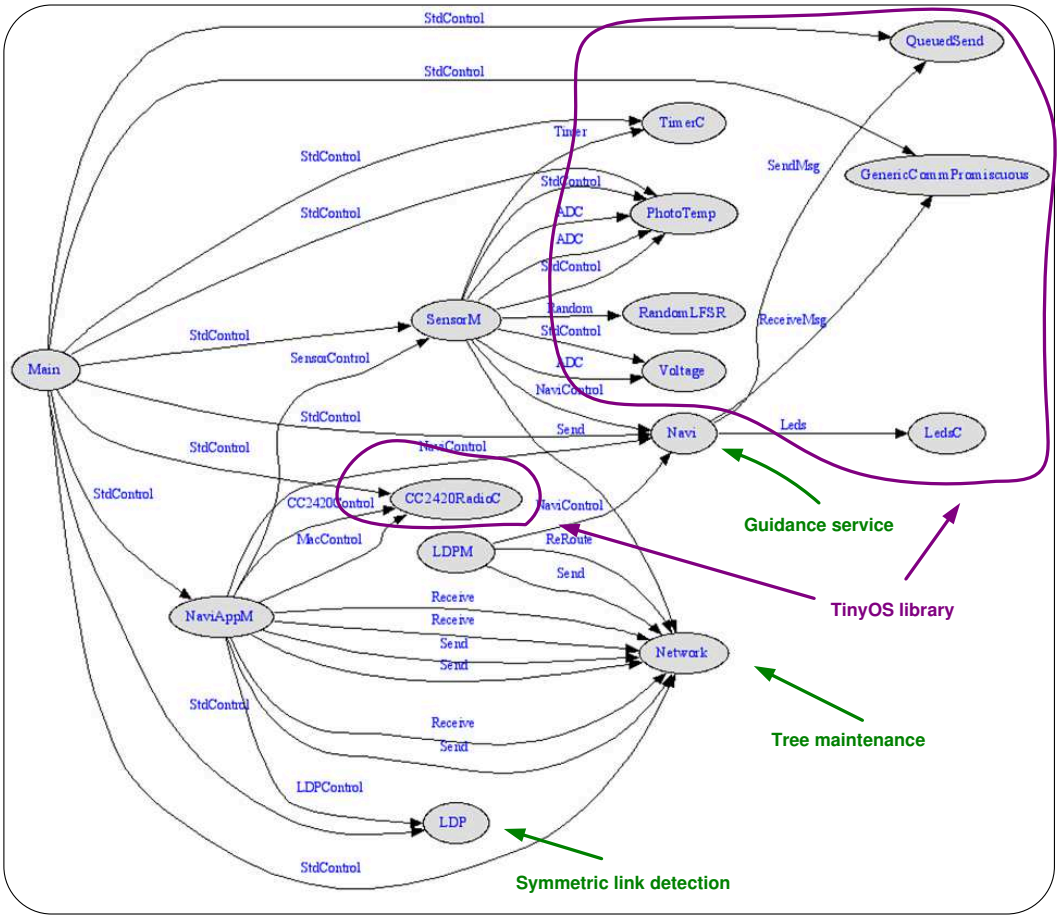
Fig. A.1 shows the flow chart of packet transmission. When a sensor receives a packet, it will check the route type of this packet. We implement three functions, *forwardUp()*, *forwardDown()*, and *broadcast()*, to deal with upstream, downstream, and broadcast packets, respectively. The function *forwardUp()* will directly forward

packets to its parent. For the downstream type, the receiving sensor checks if it is the destination, so the function *Receive.receive* will be triggered. The *Receive.recieve* function is defined by *GenericCommPromiscuous* interface provided by *TinyOS* library. When dealing with the broadcast type packet, sensors will will directly trigger the *Receive.receive* function, and then rebroadcast. The detail implementation about network receive module can be available in Fig. B.2.

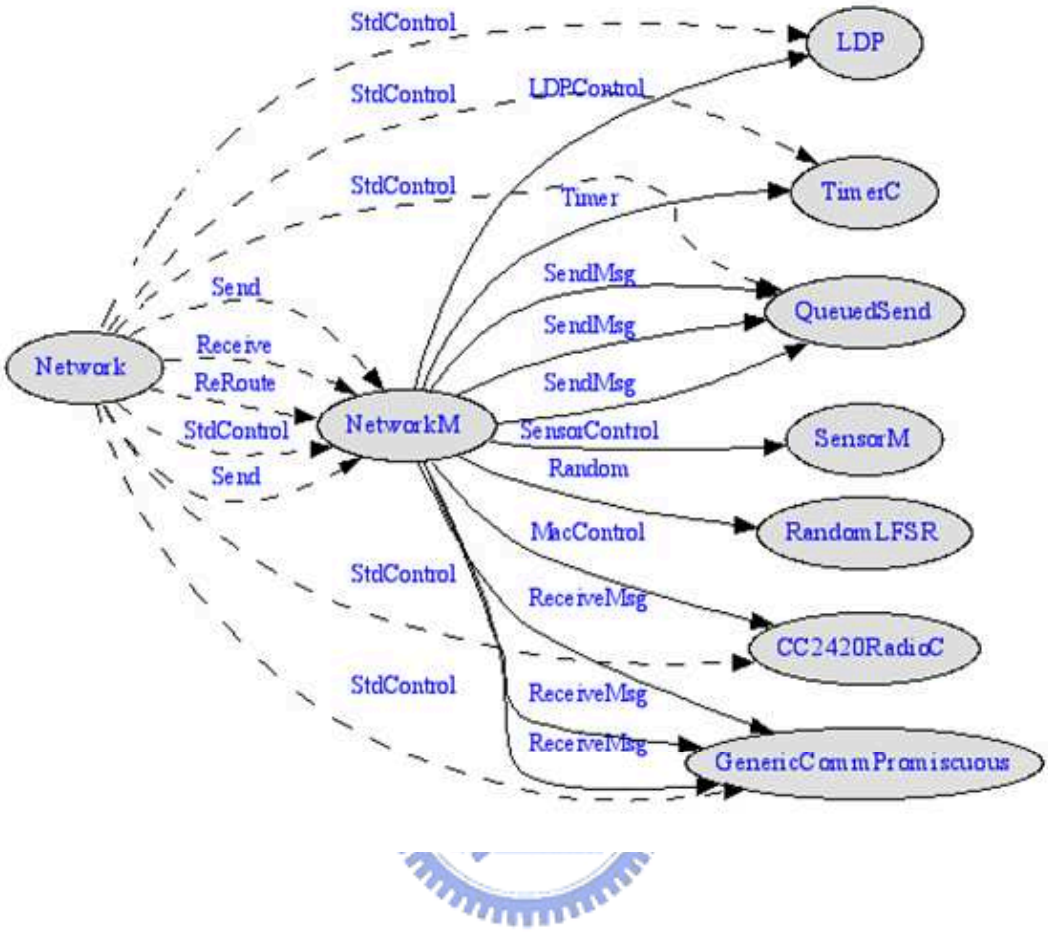
Appendix B

Component Graph

B.1 System Overview



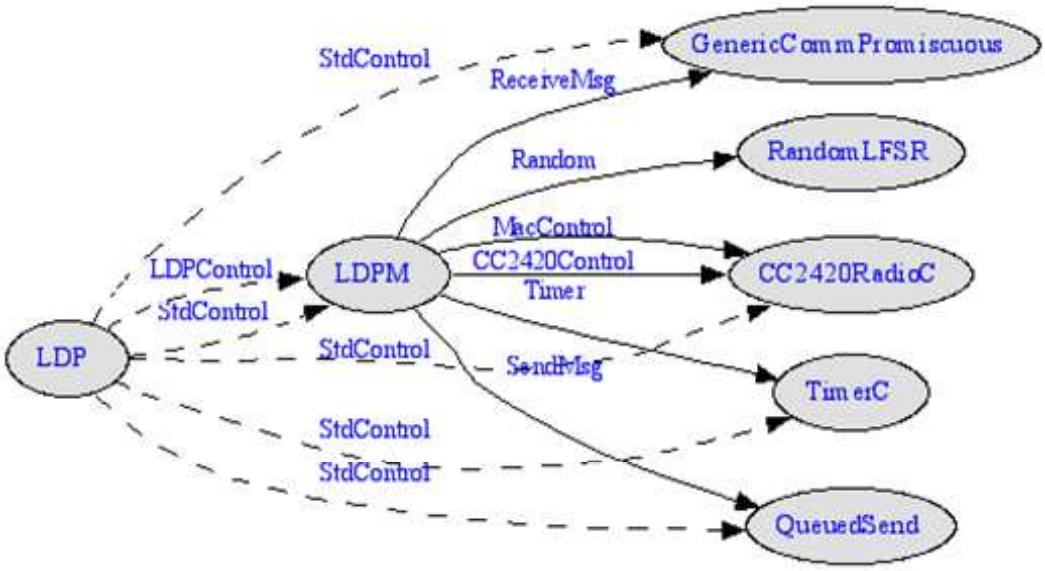
B.2 Tree Maintenance



Codes of NetworkM

```
1.  event TOS_MsgPtr NetworkReceive.receive[uint8_t id](TOS_MsgPtr pMsg) {
2.      TOSPacket       *packet = (TOSPacket*)pMsg->data;
3.      DownStreamPacket    *dsPacket = (DownStreamPacket*)packet->data;
4.      if(id != AM_NAVI_ENV && id != AM_NAVI_CMD && id != AM_NAVI_REPORT &&
          id != AM_READING && id !=AM_NAVI_WEIGHT && id != AM_NAVI_ACK &&
          id != AM_DEBUG) {
5.          return pMsg;
6.      }
7.      if( packet->UpStream ){
8.          pMsg = forwardUp(pMsg,id);
9.      } else if( packet->Broadcast ){
10.         signal Receive.receive[id](pMsg,packet->data,packet->len);
11.     } else if( packet->DownStream ){
12.         if( packet->dst == (uint8_t)TOS_LOCAL_ADDRESS ){
13.             signal Receive.receive[id](pMsg,dsPacket->data,packet->len-MAX_PATH_LENGTH);
14.         } else {
15.             pMsg = forwardDown(pMsg,id);
16.         }
17.     }
18.     return pMsg;
19. }
```

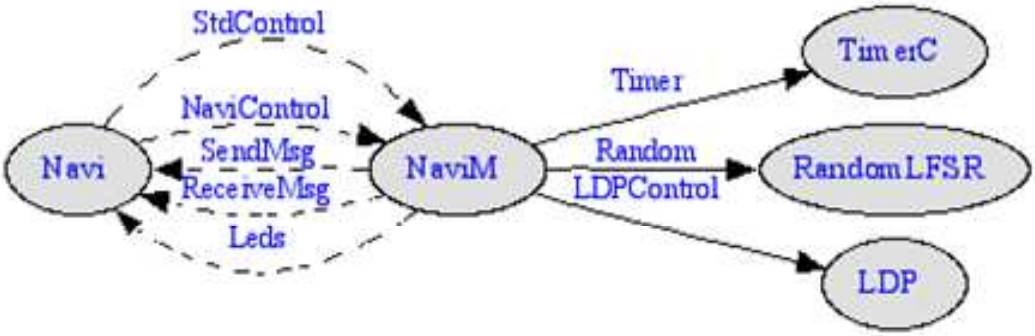
B.3 Symmetric Link Detection



Codes of the LDPM

```
1.  event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr pMsg) {
2.      LDProtocol     *ldp = (LDProtocol*)pMsg->data;
3.      if (254-pMsg->strength + 45 > RSSI_HIGHMARK) return pMsg;
4.      if( ldp->type == HELLO ){
5.          Ack(ldp->src);
6.      } else if (ldp->type == ECHO)
7.          maintainLink(ldp, 254-pMsg->strength + 45);
8.      return pMsg;
9.  }
10.
11. event result_t Timer.fired() {
12.     if( timer_state == TIMER_INIT_BACKOFF ){
13.         post Hello();
14.         adjustLink();
15.         timer_state = TIMER_COLLECT_INFO;
16.         call Timer.start(TIMER_ONE_SHOT, COLLECT_PERIOD);
17.     } else if( timer_state == TIMER_COLLECT_INFO ){
18.         timer_state = TIMER_INIT_BACKOFF;
19.         if(startup_search_count < 3){
20.             startup_search_count++;
21.             call Timer.start(TIMER_ONE_SHOT, STARTUP_INIT_BACKOFF);
22.         } else    call Timer.start(TIMER_ONE_SHOT, INIT_BACKOFF);  //5
23.     }
24.     return SUCCESS;
25. }
```

B.4 Guidance Service



Codes of NaviM

```
1.  event TOS_MsgPtr NaviReceive.receive(TOS_MsgPtr pMsg) {
2.      NaviInfo    *navi = (NaviInfo*)pMsg->data;
3.      int         navi_index = getLinkIndex(navi->src);
4.      int         emg_index;
5.      bool    new_emg = FALSE, hopChanged = FALSE, changed = FALSE;
6.      uint8_t org_level = level;
7.      float org_altitude = altitude;
8.
9.      if(Role & ROLE_BASE)return pMsg;
10.     if( navi_index == -1 ) return pMsg;
11.     if( navi->type == EMG ){
12.         if ((altitude - BaseWeight) > MAX_WEIGHT * 2) return pMsg;
13. /* case 1 */
14.         LinkTable[navi_index].level = navi->level;
15.         LinkTable[navi_index].altitude = navi->altitude;
16.         emg_index = getEmgIndex(navi->emg);
17.         /* case 1.(a) */
18.         if (emg_index == -1) {
19.             new_emg = TRUE;
20.             hopChanged = TRUE;
21.             emg_index = newEmgEntry(navi->emg, (navi->hop + 1));
22.             call LDPControl.parentIsEmg(navi->emg);
23.         /* case 1.(b) */
24.         } else {
25.             if (navi->hop + 1 < EmgPool[emg_index].hop) {
26.                 EmgPool[emg_index].hop = navi->hop + 1;
27.                 hopChanged = TRUE;
28.             }
29.         }
30.         if (Role == ROLE_SENSOR && (LinkTable[navi_index].role & ROLE_STAIR_F ||
31.             LinkTable[navi_index].role & ROLE_STAIR_S)) {
32.             level = navi->level;
33.         }
34. /* case 2 */
35.         if (hopChanged && navi->hop + 1 <= SafetyFactor) {
36.             if (level < MAX_LEVEL - 1) {
37.                 level = MAX_LEVEL - 1;
```

```
38.                         }
39.                         altitude = dangerZone(EmgPool[emg_index].hop, 0);
40.                         if ((Role & ROLE_STAIR_F || Role & ROLE_STAIR_S) &&
                                LinkTable[navi_index].dir != DIR_UP &&
41.                             LinkTable[navi_index].dir != DIR_DOWN) {
42.                             level = MAX_LEVEL;
43.                         }
44.                 }
45.                 if (hopChanged && navi->hop <= SafetyFactor && LinkTable[navi_index].dir ==
                        DIR_DOWN) {
46.                         EmgPool[emg_index].hop = navi->hop;
47.                         if (level < MAX_LEVEL - 1) {
48.                             level = MAX_LEVEL - 1;
49.                         }
50.                         altitude = dangerZone(navi->hop, 1);
51.                 }
52.         /* case 3: local minimum */
53.                 if (!(Role & ROLE_EXIT) && isLocalMinimum()) {
54.                         /* case 3.(a) */
55.                         if (Role == ROLE_SENSOR) {
56.                             altitude = localMinimum();
57.                         /* case 3.(b) */
58.                         } else if (Role & ROLE_STAIR_S || Role & ROLE_STAIR_F) {
59.                             switch(level) {
60.                                 /* case 3.(b).Ⅱ */
61.                                 case MAX_LEVEL - 1:
62.                                     level = MAX_LEVEL;
63.                                     altitude = -1 * initLevel;
64.                                     if (!canUpstairs()) {
65.                                         altitude = localMinimum();
66.                                     }
67.                                     break;
68.                                 /* case 3.(b).Ⅲ */
69.                                 case MAX_LEVEL:
70.                                     altitude = localMinimum();
71.                                     break;
72.                                 /* case 3.(b).Ⅰ */
73.                                 default:
```

```
74.                                                     altitude = localMinimum();
75.                                                     if (level < MAX_LEVEL - 1) {
76.                                                             level = MAX_LEVEL - 1;
77.                                                     }
78.                                                     break;
79.                                             }
80.                                     }
81.                             }
82.
83.                             if (org_level != level || org_altitude != altitude) {
84.                                     changed = TRUE;
85.                             }
86.     /* case 4 */
87.                             if (new_emg || hopChanged || changed) {
88.                                     navi->level = level;
89.                                     navi->altitude = altitude;
90.                                     navi->hop = EmgPool[emg_index].hop;
91.                                     emg_packet_count++;
92.                                     pMsg = forward(pMsg);
93.                                     gfNaviTimerCounter = 0;
94.                             }
95.     /* case 5 */
96.                             guide_index = findGuideIndex();
97.                             showDirection(LinkTable[guide_index].dir);
98.                             if (!hasEmgNode) {
99.                                     regularReport();
100.                            }
101.                    }
102.                    return pMsg;
103.    }
```

Bibliography

[1] Design and construction of a wildfire instrumentation system using networked sensors. http://firebug.sourceforge.net/.

[2] Habitat monitoring on great duck island. http://www.greatduckisland.net/technology.php.

[3] Zigbee alliance. http://www.zigbee.org/.

[4] J. Bachrach, R. Nagpal, M. Salib, and H. Shrobe. Experimental results and theoretical analysis of a self-organizing global coordinate system for ad hoc sensor networks. *Telecommunications Systems Journal Special Issue on Wireless System Networks*, 26(2-4):213–234, 2004.

[5] P. Bahl and V. N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *Proc. of IEEE INFOCOM*, pages 775–784, 2000.

[6] M. A. Batalin, G. S. Sukhatme, and M. Hattig. Mobile robot navigation using a sensor network. In *Proc. of IEEE Int'l Conf. on Robotics and Automation*, 2004.

[7] A. Boukerche, R. W. N. Pazzi, and R. B. Araujo. A fast and reliable protocol for wireless sensor networks in critical conditions monitoring applications. In *Proc. of ACM Int'l Symp. on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, 2004.

[8] D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. In *Proc. of ACM Int'l Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002.

[9] S. Capkun, M. Hamdi, and J.-P. Hubaux. GPS-free positioning in mobile ad-hoc networks. In *Proc. of Hawaii Int'l Conf. on Systems Science (HICSS)*, 2001.

[10] P. Corke, R. Peterson, and D. Rus. Networked robots: Flying robot navigation using a sensor net. In *Proc. of Int'l Symp. Robotics Research (ISRR)*, 2003.

[11] T. Dam and K. Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *Proc. of ACM Int'l Conf. on Embedded Networked Sensor Systems (SenSys)*, 2003.

[12] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocols for wireless microsensor networks. In *Proc. of Hawaii Int'l Conf. on Systems Science (HICSS)*, 2000.

[13] C.-F. Huang, Y.-C. Tseng, and L.-C. Lo. The coverage problem in three-dimensional wireless sensor networks. In *Proc. of IEEE Global Telecommunications Conference (Globecom)*, 2004.

[14] IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks specific requirements part 15.4: wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs), 2003.

[15] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Networking*, 11(1):2–16, 2003.

[16] G. Kantor, S. Singh, R. Peterson, D. Rus, A. Das, V. Kumar, G. Pereira, and J. Spletzer. Distributed search and rescue with robot and sensor teams. In *Proc. of Int'l Conf. on Field and Service Robotics*, 2003.

[17] M. Kochhal, L. Schwiebert, and S. Gupta. Role-based hierarchical self organization for wireless ad hoc sensor networks. In *Proc. of ACM Int'l Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 98–107, 2003.

[18] Q. Li, M. DeRosa, and D. Rus. Distributed algorithm for guiding navigation across a sensor network. In *Proc. of ACM Int'l Symp. on Mobile Ad Hoc Networking and Computing (MobiHOC)*, 2003.

[19] C.-Y. Lin and Y.-C. Tseng. Structures for in-network moving object tracking in wireless sensor networks. In *Proc. of Broadband Wireless Networking Symp.(BroadNet)*, 2004.

[20] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *Proc. of IEEE INFOCOM*, pages 1380–1387, 2001.

[21] D. Niculescu and B. Nath. DV based positioning in ad hoc networks. *Telecommunications Systems Journal*, 22(1-4):267–280, 2003.

[22] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proc. of IEEE INFOCOM*, 1997.

[23] R. Peterson and D. Rus. Interacting with a sensor network. In *Proc. of Australasian Conf. on Robotics and Automation*, 2002.

[24] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Commun.*, 7(5):16–27, October 2000.

[25] Y.-C. Tseng, S.-P. Kuo, H.-W. Lee, and C.-F. Huang. Location tracking in a wireless sensor network by mobile agents and its data fusion strategies. In *Proc. of Int'l Symp. on Information Processing in Sensor Networks (IPSN)*, 2003.

[26] Y.-C. Tseng, M.-S. Pan, and Y.-Y. Tsai. Wireless sensor networks for emergency navigation. In *IEEE Comput.*, (to appear).

[27] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *Proc. of IEEE INFOCOM*, pages 1567–1576, 2002.