

國立交通大學

資訊學院 資訊工程學系

博 士 論 文

物件追蹤感測網路中與位置管理相關的通訊協定



Communication Protocols for Location Management in Object
Tracking Sensor Networks

研 究 生：林致宇

指導教授：曾煜棋 教授

共同指導教授：彭文志 教授

中 華 民 國 九 十 六 年 九 月

物件追蹤感測網路中與位置管理相關的通訊協
Communication Protocols for Location Management in Object Tracking Sensor
Networks

研 究 生：林致宇

Student：Chih-Yu Lin


指導教授：曾煜棋

Advisor：Yu-Chee Tseng

共同指導教授：彭文志

Co-advisor：Wen-Chih Peng

國 立 交 通 大 學 資 訊 學 院
資 訊 工 程 學 系
博 士 論 文



A Dissertation
Submitted to Department of Computer Science
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in

Computer Science

September 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年九月

物件追蹤感測網路中與位置管理相關的通訊協定

指導教授：曾煜棋 教授

學生：林致宇

彭文志 教授

國立交通大學資訊工程學系（研究所）博士班

摘 要

無線通訊與感測技術的快速發展使得無線感測網路成為一門新興的科技，無線感測網路的應用也因此被廣泛地討論。在無線感測網路中物件追蹤是一個重要的議題，它包含了許多的應用，例如軍事上入侵者的偵測、野生動物棲息地的監控等。物件追蹤包含了幾個重要的步驟，例如事件的偵測、目標物的辨識、位置的估算等，在無線感測網路中，當物件的位置被估算出來之後，為了能讓使用者去查詢物件的位置，或者為了能讓感測到物件的感測器做回報，一個位置管理機制是需要的。本論文的主題即討論無線感測網路上位置管理相關的通訊協定。我們所提出的位置管理機制利用了無線感測網路的網路內資料處理的能力，以分散式的方式去執行物件位置的更新與查詢。我們更考量了在多資料匯集端的網路下的位置管理機制，在這樣的環境下我們假設使用者可在任一個地方發出位置的查詢。而由於感測資料的不精確是感測網路先天上特有的特性，我們也考慮了在使用者可容忍一些誤差的網路環境下，位置管理機制該如何達成的問題。而不管在哪一種網路環境下，位置管理機制的目的都是為了要降低位置更新與位置查詢所產生的通訊成本。此外，我們觀察到網路底層封包的遺失與碰撞可能導致物件追蹤網路中產生不正確的位置資訊，而物件追蹤網路是一種以事件驅動為主的網路，因此我們也針對以事件驅動為主的感測網路提出了一個鏈結層通訊協定來降低封包碰撞的問題。

物件追蹤通常包含了兩個基本的操作：位置更新與位置查詢。一般而言，位置更新是在當一個物件從一個感測區域移動到另一個感測區域時發生，而位置查詢則是當使用者有需要知道物件的位置而發生查詢。無線感測網路處理位置更新與查詢的方法有很多，一個處理更新與查詢最簡單的做法是將使用者的查詢送給網路上的每一個感測節點，而偵測到物件的感測器在收到查詢後就會回覆物件的位置給資料匯集端，我們可以發現在這個方法中，感測器不需要主動地做位置的更新，然而，很顯然這個方法在網路規模很大或者是查詢頻率很高時會非常地沒有效率，因為使用者的查詢必須被廣播到整個網路上。第二種處理更新與查詢的方法則是要求感測器偵測到物件時就必須主動地將物件的新位置傳送給資料匯集端，如此資料匯集端就隨時有物件的位置資訊，因此資料匯集端本身就可以馬上回覆使用者的查詢，而不再需要將查詢送到感測網路上，然而這方法在物件移動相當頻繁時會產生許多的位置更新訊息，因此我們可以很明顯地看出上述兩個方法是各有利弊。在這篇論文中，我們首先針對單一資料匯集端的網路環境提出一個樹狀的位置管理機制，在建樹的過程中，我們考量了網路實際上的拓撲對通訊代價產生的影響。建樹的程序主要分成兩個階段，在第一個階段我們主要目的是降低位置更新所產生的通訊代價，我們利用了避免偏向(deviation-avoidance)與高移動頻率優先(high-weight-first)這個特性去降低通訊代價。而在第二個階段，我們則是以第一階段所產生的樹再加上位置查詢的

考量而去做調整。

接著，我們則考慮了一個多資料匯集端的感測網路。擁有多重資料匯集端的一個好處是可降低位置查詢的回應時間，除此之外也能降低資料匯集端附近網路流量擁塞的問題，亦即可達到較好的負載平衡。為了要支援多資料匯集端的環境，我們可考慮將單一資料匯集端的方法做延伸，也就是每個資料匯集端都建構一個樹，然而這也就意味著更新多個樹是需要的，因此如何去降低更新多個樹所產生的通訊代價便是我們要去解決的問題。在這篇論文中，為了解決此問題，我們利用了資料匯集的概念提出了有效率的位置更新機制。有了這個機制，我們發現在多重資料匯集端的環境下，位置更新的通訊代價只會增加少許。此外，我們也根據前述的更新機制推導出更新代價的公式，並以此公式設計出兩個分散式的建樹演算法。

在物件移動的環境中要維持物件正確的位置資訊幾乎是不可能的，原因除了定位技術本身就不是很準確，物件的移動與資料傳送的延遲也都使得查詢者得到的位置資訊並不精確。然而幸運的是在物件追蹤的應用中，不精確的位置資訊通常是能夠被容忍的，例如生物科學家為了要追蹤某動物時，生物學家可能只需要知道大概的移動方向即可而不需要知道正確的位置，除此之外，當科學家只是為了要觀察動物的日常作息時，幾個小時前的位置資訊對科學家們仍是有用的資訊。因此我們也提出了一個位置管理機制去支援能夠容忍不精確位置資訊的物件追蹤感測網路。我們認為這個位置管理機制應該要能達成兩個目標，第一，位置查詢的通訊代價應該跟精確程度成正比，第二，多精確程度的支援。我們觀察到樹狀架構的位置管理機制可以很容易地達成這兩個目標，因此我們也提出了一個建樹的演算法。

藉著實驗的模擬我們觀察到封包的碰撞與遺失會導致使用者得到不正確的物件位置資訊，因此我們也提出了一個鏈結層協定來減輕碰撞的問題。無線感測網路一般可分成兩類，一是事件驅動另一則是時間驅動。在事件驅動的感測網路中，感測器會在偵測到事件時做回報的動作，而這樣的回報可能造成事件發生區域的感測器遭受到較高的傳輸競爭。在這篇論文中，我們結合了兩個議題來解決這個問題，一是利用感測資料的空間相關性，另一則是設計一個新的媒介存取協定。我們提出了一個結合 TDMA(Time Division Multiple Access)與 CSMA(Carrier Sense Multiple Access)的鏈結層協定，它與傳統以 TDMA 為基礎的協定有幾個不同的特色。第一，在 TDMA 的部份我們只需要較寬鬆的時間同步且是經由事件的驅動而啟動 TDMA，而 CSMA 則在非事件發生區域被採用以達到低傳送延遲。第二，時槽分配策略考量了感測資料的空間相關性，我們發現藉著允許一個感測器可使用和其鄰居相同的時槽，可使得整個網路所需要的時槽數大幅地降低。第三，藉著拉長一個時槽的長度，我們可在封包離開事件區域後能像水流一樣依序地前進，每個封包會相隔一定的距離以避免干擾。除此之外，利用 TDMA 的特性及感測資料的空間相關性，我們也提出了一個降低回報量的機制。而為了達到省電的目的，我們也討論如何將 LPL (Low Power Listening)的技術結合到我們的方法。

關鍵詞：無線感測網路、物件追蹤、網路內資料處理、資料匯集、行動計算、位置管理、媒介存取控制、空間相關性、TDMA、CSMA

Communication Protocols for Location Management in Object Tracking Sensor Networks

Student : Chih-Yu Lin

Advisors : Dr. Yu-Chee Tseng
Dr. Wen-Chih Peng

Department of Computer Science
National Chiao Tung University

ABSTRACT

The rapid progress of wireless communication and embedded micro-sensing MEMS technologies has made *wireless sensor networks* (WSNs) possible. Applications of WSNs have been studied widely. Object tracking is one of the important issues of WSNs, which has applications in such as military intrusion detection and habitat monitoring. The key steps involved in object tracking include event detection, target classification, and location estimation. In a WSN, when the locations of objects are successfully determined, a location management scheme for reporting objects' locations and disseminating users' queries is required. The main theme of this dissertation is location management. The proposed location management schemes explore the in-network data processing capability of WSNs by executing distributed location updates and queries inside the network. We further consider the multi-sink system, in which a user can issue queries from anywhere in a WSN. Since inaccuracy of sensing data is inherent for WSNs, we also consider the scenarios where users can tolerate a certain degree of imprecision in their query results. The goal of location management schemes is to reduce the communication cost. Besides, we observe that packet collision can lead to incorrect location information. Thus, we also propose a link-layer protocol to relieve the collision problem for event-driven WSNs.

Object tracking typically involves two basic operations: *update* and *query*. In general, updates of an object's location are initiated when the object moves from one sensor to another. A query is invoked each time when there is a need to find the location of an interested object. Location updates and queries may be done in various ways. A naive way for delivering a query is to flood the whole network. The sensor whose sensing range contains the queried object will reply to the query. Clearly, this approach is inefficient because a considerable amount of energy will be consumed when the network scale is large or when the query rate is high. Alternatively, if all location information is

stored at a specific sensor (e.g., the sink), no flooding is needed. But whenever a movement is detected, update messages have to be sent. One drawback is that when objects move frequently, abundant update messages will be generated. The cost is not justified when the query rate is low. Clearly, these are tradeoffs. In this dissertation, we first propose a tree-based location management scheme for single-sink WSNs. We develop several tree structures for in-network object tracking which take the physical topology of the sensor network into consideration. The optimization process has two stages. The first stage tries to reduce the location update cost based on a deviation-avoidance principle and a highest-weight-first principle. The second stage further adjusts the tree obtained in the first stage to reduce the query cost.

We then explore the possibility of having multiple sinks in the network. One advantage of having multiple sinks is to reduce the response time of queries. In addition, using multiple sinks can also relieve the traffic congestion problem associated with a single-sink system (i.e., using multiple sinks can achieve load balance more easily). In order to support location management in a multi-sink WSN, we can extend the tree structure used in the single-sink system by constructing a logical tree for each sink. However, this implies that updating multiple trees is required when a movement event is detected. It is desirable to further reduce the update cost when multiple trees coexist in the network. In this dissertation, by exploring the concept of data aggregation, we propose an algorithm to efficiently update multiple trees. With proper design, we show that the update cost increases slightly when the number of trees (i.e., the number of sinks) increases. Based on the foregoing update algorithm, we formulate the update cost that gives us hints to develop efficient tree-construction algorithms. Two distributed multi-tree construction algorithms are also presented.

In moving object environments, maintaining the exact locations of objects anytime is almost infeasible. Not only the positioning results are error-prone, but also the data transfer delay and object mobility make the locations of objects inaccurate. Fortunately, imprecision is tolerable in many object tracking applications. For example, when life scientists intend to track an animal, it may be sufficient to know its moving direction rather than its exact location. In addition, the location information recorded several hours ago, instead of at the current time, may be still available for the life scientists to understand the animal's daily life. Therefore, we also propose an in-network location management scheme to support imprecision-tolerant queries for object tracking sensor networks. We argue that an imprecision-tolerant location management solution should achieve two desirable goals. First, the query cost should be proportional to the precision level. Second, multiple precision levels should be provided. We observe that the tree-based location management schemes could achieve these two goals inherently. Thus, we also propose a tree construction algorithm for imprecision-tolerant location management model.

By simulation, we observe that packet collision can lead to incorrect location information in object tracking sensor networks. Thus, we also propose a link-layer protocol to relieve the collision

problem for event-driven WSNs. Wireless sensor networks (WSNs) can generally be classified into two categories: time-driven and event-driven. In an event-driven WSN, sensors report their readings only when they detect events. In such behavior, sensors in the event area may suffer from higher contention. In this dissertation, we solve this problem by jointly considering two subissues. One is exploiting the spatial correlation of data reported by sensors in the event area and the other is designing a specific MAC protocol. We propose a novel hybrid TDMA/CSMA protocol with the following interesting features that differentiate itself from conventional TDMA-based protocols. First, the TDMA part is based on very loose time synchronization and is triggered by the appearance of events. On the other hand, the CSMA part is adopted in the non-event area to achieve low latency transmission. Second, the slot assignment strategy associated with the TDMA part takes the spatial correlation of sensor data into consideration and thus allows less strict slot allocation than conventional TDMA schemes. Interestingly, by intentionally allowing one-hop neighbors to share the same time slot, the number of slots required per frame is significantly reduced. Third, by enlarging the slot size on purpose, our scheme enforces packets, after leaving the event area, to form a pipeline in such a way that packets flow like streams, each of which is separated sufficiently in distance to avoid interference. In addition, by exploiting TDMA's features and the spatial correlation of sensor data, we show how to reduce redundant reports. We also discuss how to combine our protocol with the LPL (Low Power Listening) technique to achieve energy efficiency.



Keywords: wireless sensor network, object tracking, in-network processing, data aggregation, mobile computing, location management, imprecision-tolerant, MAC, TDMA, CSMA, spatial correlation.

Acknowledgements

Special thanks goes to my advisor Professor Yu-Chee Tseng for his guidance in my dissertation work. I also thank my co-advisor Professor Wen-Chih Peng for his continuous support in the Ph.D. program. Besides my advisors, I would like to thank Professor Ten Hwang Steve Lai for his kindness and guidance during the period that I visited the Ohio State University last year. I would also like to thank my dissertation committee members: Professor Jang-Ping Sheu, Professor Tsung-Chuan Huang, Professor Yuh-Shyan Chen, Professor Rong-Hong Jan, and Professor Hsi-Lu Chao. They asked me some good questions and gave me useful comments so that I can improve my work in the future.

Let me also say ‘thank you’ to all members of High-Speed Communication & Computing Laboratory for their assistance and kindly helping both in the research and the daily life during these years. Finally, I will dedicate this dissertation to my families for their love and support.



Contents

Abstract(in Chinese)	i
Abstract(in English)	iii
Acknowledgements	vi
Contents	vii
List of Tables	xi
List of Figures	xii
1 Introduction	1
1.1 Location Management for Single-Sink WSNs	2
1.2 Location Management for Multi-Sink WSNs	3
1.3 Imprecision-tolerant Location Management Model	4
1.4 A Link-layer Protocol for Event-driven WSNs	5
1.5 Organization of This Dissertation	5
2 Related Works	7
2.1 Object Tracking Using Wireless Sensor Networks	7
2.2 Location Management in Object Tracking Sensor Networks	8



2.3	MAC Protocols for Wireless Sensor Networks	8
3	In-network Location Management for the Single-sink System	10
3.1	Preliminaries	11
3.2	Tree Construction Algorithms	15
3.2.1	Algorithm DAT (Deviation-Avoidance Tree)	15
3.2.2	Algorithm Z-DAT (Zone-based Deviation-Avoidance Tree)	20
3.2.3	Algorithm QCR (Query Cost Reduction)	22
3.3	Simulation Results	29
3.4	Summary	35
4	In-network Location Management for the Multi-sink System	37
4.1	Preliminaries	38
4.1.1	Network Model	38
4.1.2	From Single-sink to Multi-sink WSNs	38
4.2	Update and Query Mechanisms in Multi-Sink WSNs	41
4.2.1	Notations and Data Structures	41
4.2.2	The Location Update Mechanism	42
4.2.3	The Location Query Mechanism	46
4.3	Multi-Tree Construction Algorithms	48
4.3.1	The MT-HW Algorithm	48
4.3.2	The MT-EO Algorithm	49
4.4	Simulation Results	49
4.4.1	Impact of Objects' Speeds	51
4.4.2	Impact of Query Rates	56
4.4.3	Impact of the Number of Sinks	56

4.4.4	Multi-Sink Systems with Partial Storage	59
4.5	Summary	62
5	Imprecision-tolerant Location Management Model	64
5.1	Preliminaries	65
5.1.1	Background and Motivations	65
5.1.2	Network Model	67
5.2	Imprecision-tolerant Location Management Model	68
5.2.1	Imprecision-tolerant Update and Query Mechanisms . . .	68
5.2.2	Tree Optimization	71
5.3	Simulation Results	80
5.4	Summary	85
6	A Link-layer Protocol for Event-driven WSNs	86
6.1	Preliminaries	86
6.1.1	Background and Motivations	86
6.1.2	Some Observations	89
6.2	The Proposed Schedule-based Approach	90
6.2.1	Overview	90
6.2.2	Operations in the ES Mode	92
6.2.3	Operations in the NES Mode	103
6.2.4	Extension for Achieving Energy Efficiency	103
6.3	Simulation Results	104
6.3.1	Evaluation of SC-MAC	106
6.3.2	Evaluation of Parameters in SC-MAC	109
6.4	Summary	110

7	Conclusions and Future Directions	115
	Bibliography	118
	Vita	123



List of Tables

3.1	Summary of notations used in Chapter 3.	14
4.1	Summary of notations used in Chapter 4.	42
4.2	Parameters used in the simulation for multi-sink systems.	51
5.1	Parameters used in the simulation for imprecision-tolerant location management model.	81
6.1	Parameters used in the simulation for our proposed link-layer protocol.	106

List of Figures

3.1	(a) The Voronoi graph of a sensor network. (b) The graph G corresponding to the sensor network in (a).	11
3.2	(a) An object tracking tree T . (b) The events generated as Car1 moves from sensor K to G and Car2 moves from H to C	13
3.3	Four possible location tracking trees for the graph in Fig. 3.1(b).	17
3.4	Snapshots of an execution of DAT.	19
3.5	Possible structures of subtrees with nine sensors.	20
3.6	An example of the Z-DAT algorithm with $\alpha = 4$	22
3.7	Making a non-leaf node v a leaf node.	24
3.8	Connecting a leaf node v_i to $p(p(v_i))$	25
3.9	An execution example of algorithm QCR.	26
3.10	Comparison of update costs.	31
3.11	Snapshots of tree T obtained by DAT and Z-DAT under regular and random deployments.	32
3.12	Comparison of update costs under different (α, δ) for Z-DAT.	33
3.13	Comparison of query costs. ($C = 1.0$)	34
3.14	Comparison of total costs. ($C = 1.0$)	35
4.1	An example of the single-sink system.	39

4.2	(a) The <i>DLs</i> stored in sensors. (b) An example where <i>Car2</i> moves from <i>G</i> to <i>I</i> and <i>Car1</i> moves from <i>H</i> to <i>C</i>	40
4.3	(a) The query graph G'_{Car1} of Fig. 4.2(a) for <i>Car1</i> . (b) Another example of a two-tree system. (c) The query graph of (b) for <i>Car1</i> , which contains a cycle.	47
4.4	Performance study with objects' speeds varied, where sensors are deployed regularly and four sinks are deployed.	52
4.5	Performance study with objects' speeds varied, where sensors are deployed regularly and 256 sinks are deployed.	54
4.6	Performance study with objects' speeds varied, where sensors are deployed randomly and four sinks are deployed.	55
4.7	Performance study with query rates varied, where sensors are deployed regularly and four sinks are deployed.	57
4.8	Performance study with query rates varied, where sensors are deployed regularly and 256 sinks are deployed.	58
4.9	Performance study with the number of sinks varied, where the objects' speed is set to be 0.333 unit/second.	60
4.10	Performance study with the number of sinks varied, where the objects' speed is set to be 0.111 unit/second.	61
4.11	The performance of the partial storage technique.	62
5.1	Examples of spatial imprecision and temporal imprecision.	66
5.2	A tree-based location management scheme.	67
5.3	The values of <i>be_queried</i> of sensors collected by (a) ideal-collection and (b) tree-collection, where the DAT tree is used.	72

5.4	Some observations.	73
5.5	(a) The basic idea of constructing a tree. (b) The problem arising when connecting two subtrees.	74
5.6	The impact of objects' speeds.	82
5.7	The impact of query rates.	83
5.8	Comparison of ratios of update cost to query cost.	83
5.9	The impact of objects' speeds under two different query scenarios	84
5.10	The impact of query rates under two different query scenarios. . .	84
6.1	Two examples of event reporting in an event-driven WSN.	87
6.2	The hidden terminal problem, where R_{tran} denotes the transmis- sion range of sensors.	90
6.3	The redundancy problem.	91
6.4	Comparison of slot assignment strategies.	96
6.5	The impact of slot size beyond event areas.	97
6.6	The advantage of separating flows in time.	98
6.7	The synchronization problem.	111
6.8	Combining our scheme with LPL.	111
6.9	Comparison of different schemes, where $R_{corr} > R_{tran}$	112
6.10	Comparison of different schemes, where $R_{corr} < R_{tran}$	113
6.11	The impact of α under different ratios of R_{corr} to R_{tran}	113
6.12	The impact of ℓ , where R_{corr} is 15 units, R_{tran} is 10 units, and the value of α is 0.3.	114
6.13	The impact of ℓ , where R_{corr} is 5 units, R_{tran} is 10 units, and the value of α is 1.0.	114

Chapter 1

Introduction

The emerging wireless sensor network (WSN) technology may greatly facilitate human life. A WSN may consist of many inexpensive wireless nodes, each capable of collecting, processing, and storing environmental information, and communicating with other nodes. A lot of research efforts have been dedicated to WSNs, including design of physical and medium access layers [26, 35] and routing and transport protocols [10, 13]. Applications of WSNs have been studied in [2, 6, 17].

Object tracking is an important application of WSNs (e.g., military intrusion detection and habitat monitoring). The key steps involved in tracking include event detection, target classification, and location estimation [3, 5, 15, 18]. In a WSN, when the locations of objects are successfully determined, a location management scheme for reporting objects' locations and disseminating users' queries is required [14, 16]. The main theme of this dissertation is location management. In particular, we explore the in-network data processing capability of WSNs by executing distributed location updates and queries inside the network. Updates of an object's location are initiated when the object moves from one sensor to another. A query is invoked each time when there is a need to find the location of an interested object. Location updates and queries may be done in various ways. A naive way for delivering a query is to flood the whole network. The sensor whose sensing range contains the queried object will reply to the query. Clearly,

this approach is inefficient and not scalable because a considerable amount of energy will be consumed when the network scale is large or when the query rate is high. Alternatively, if all location information is stored at a specific sensor (e.g., the sink), no flooding is needed. But whenever a movement is detected, update packets have to be sent to the sink. Thus, when objects move frequently, abundant update packets will be generated. The cost is not justified when the query rate is low. Clearly, these are tradeoffs.

1.1 Location Management for Single-Sink WSNs

In [14], a *Drain-And-Balance* (DAB) tree structure is proposed to address the issue of location management. As far as we know, this is the first in-network object tracking approach in sensor networks where query messages are not required to be flooded and update messages are not always transmitted to the sink. However, [14] has two drawbacks. First, a DAB tree is a logical tree not reflecting the physical structure of the sensor network; hence, an edge may consist of multiple communication hops and a high communication cost may be incurred. Second, the construction of the DAB tree does not take the query cost into consideration. Therefore, the result may not be efficient in some cases.

In this dissertation, we propose a tree-based location management scheme for single-sink WSNs. We develop several tree structures for in-network object tracking which take the physical topology of the sensor network into consideration. The optimization process has two stages. The first stage aims at reducing the update cost, while the second stage aims at further reducing the query cost. For the first stage, several principles, namely deviation-avoidance and highest-weight-first ones, are pointed out to construct an object tracking tree to reduce the communication cost of location update. Two tree construction algorithms are proposed: *Deviation-Avoidance Tree* (DAT) and *Zone-based Deviation-Avoidance Tree* (Z-DAT). The latter approach tries to divide the sensing area into square-like zones, and recursively combine these zones into a tree. Our simulation results indicate

that the Z-DAT approach is very suitable for regularly deployed sensor networks. For the second stage, we develop a *Query Cost Reduction* (QCR) algorithm to adjust the object tracking tree obtained in the first stage to further reduce the total cost. The way we model this problem allows us to analytically formulate the update and query costs of the solution based on several parameters of the given problem, such as rates that objects cross the boundaries between sensors and rates that sensors are queried. We have also conducted extensive simulations to evaluate the proposed solutions. The results do validate our observations.

1.2 Location Management for Multi-Sink WSNs

We further explore the possibility of having multiple sinks in the network. One advantage of having multiple sinks is to reduce the response time of queries. In addition, using multiple sinks can also relieve the traffic congestion problem associated with a single-sink system (i.e., using multiple sinks can achieve load balance more easily). In order to support location management in a multi-sink sensor network, we can extend the tree structure used in the single-sink system by constructing a logical tree for each sink. However, this implies that updating multiple trees is required when a movement event is detected. Assuming that there are m sinks coexisting in the network, if each tree is updated independently, the update cost will become approximately m times. It is desirable to further reduce the update cost when multiple trees coexist in the network. In this dissertation, by exploring the concept of data aggregation, we propose an algorithm to efficiently update multiple trees. With proper design, we show that the update cost increases slightly when the number of trees (i.e., the number of sinks) increases. Based on the foregoing update algorithm, we formulate the update cost that gives us hints to develop efficient tree-construction algorithms. Two distributed multi-tree construction algorithms are presented. Experimental results show that the increased update cost with multiple trees can be compensated by lower query cost and the query cost also depends on m , the number of sinks. This allows us to further

investigate how to choose the value of m under different scenarios.

1.3 Imprecision-tolerant Location Management Model

Since inaccuracy, or even error, of sensing data is inherent for WSNs, applications of WSNs usually have to tolerate some degree of imprecision. This property has been exploited in the design of network protocols for WSNs. For example, precision-constrained data aggregation is considered in [28], and a storage system that supports drill-down queries with different precision levels is proposed in [11]. Similarly, in moving object environments, maintaining the exact locations of objects anytime is almost infeasible. Not only the positioning results are error-prone, but also the data transfer delay and object mobility make the locations of objects inaccurate. Fortunately, imprecision is tolerable in many object tracking applications. For example, when life scientists intend to track an animal, it may be sufficient to know its moving direction rather than its exact location. In addition, the location information recorded several hours ago, instead of at the current time, may be still available for the life scientists to understand the animal's daily life.

In this dissertation, we propose an in-network location management scheme to support imprecision-tolerant queries for object tracking sensor networks. We intend to develop a location management model that can achieve two goals. First, multiple imprecision levels should be provided. Second, the query cost should be proportional to the imprecision level. To achieve these two goals, we propose a tree-based imprecision-tolerant location management model. To begin with, we present the update and query mechanisms that can support imprecision-tolerant queries. We then propose a tree construction algorithm to reduce the query cost while minimize the increment of update cost.

1.4 A Link-layer Protocol for Event-driven WSNs

By simulation, we observe that packet loss may make the location information incorrect in object tracking sensor networks. Thus, we also propose a link-layer protocol to relieve the contention and collision problems for event-driven WSNs. We solve these problems by jointly considering two subissues. One is exploiting the spatial correlation of data reported by sensors in the event area, and the other is designing a specific MAC protocol. We propose a novel hybrid TDMA/CSMA protocol with the following interesting features that differentiate itself from conventional TDMA-based protocols. First, the TDMA part is based on very loose time synchronization and is triggered by the appearance of events. On the other hand, the CSMA part is adopted in the non-event area to achieve low latency transmission. Second, the slot assignment strategy associated with the TDMA part takes the spatial correlation of sensor data into consideration and thus allows less strict slot allocation than conventional TDMA schemes. Interestingly, by intentionally allowing one-hop neighbors to share the same time slot, the number of slots required per frame is significantly reduced. Third, by enlarging the slot size on purpose, our scheme enforces packets, after leaving the event area, to form a pipeline in such a way that packets flow like streams, each of which is separated sufficiently in distance to avoid interference. In addition, by exploiting TDMA's features and the spatial correlation of sensor data, we show how to reduce redundant reports. We also discuss how to combine our protocol with the LPL (Low Power Listening) technique to achieve energy efficiency.

1.5 Organization of This Dissertation

This dissertation is organized as follows. Related works are surveyed in Chapter 2. In Chapter 3, we present the proposed location management scheme for single-sink WSNs. In Chapter 4, we further explore the possibility of having multiple sinks. Based on the tree-based location management schemes, we propose an

imprecision-tolerant location management model in Chapter 5. In Chapter 6, we propose a link layer protocol to solve the packet loss problem that may make location information incorrect. Finally, we conclude our research results and propose some future directions in Chapter 7.



Chapter 2

Related Works

In this chapter, we first review some papers addressing the object tracking issues in wireless sensor networks. Because the main theme of this dissertation is location management. In Sec. 2.2, we discuss some existing location managements schemes proposed for WSNs. As we mentioned above, packet loss may make location information incorrect. Packet loss is usually caused by contention and collision. We propose a link layer protocol to relieve the contention and collision problems. Because MAC (Medium Access Control) protocols are usually used to solve the contention and collision problems, we review some medium access schemes developed for wireless sensor networks in Sec. 2.3.

2.1 Object Tracking Using Wireless Sensor Networks

A significant amount of research effort has been elaborated upon issues of object tracking problems. The authors in [34] explored a localized prediction approach for power efficient object tracking by putting unnecessary sensors in sleep mode. Techniques for cooperative tracking by multiple sensors have been addressed in [3, 7, 18, 37]. In [7], a dynamic clustering architecture that exploits signal strength observed by sensors is proposed to identify the set of sensors to track an object. In [37], a *convoy tree* is proposed for object tracking using data aggregation to reduce energy consumption.

2.2 Location Management in Object Tracking Sensor Networks

In [14], a *Drain-And-Balance* (DAB) tree structure is proposed to address the location management issue. As far as we know, this is the first in-network object tracking approach in sensor networks where query messages are not required to be flooded and update messages are not always transmitted to the sink. However, [14] has two drawbacks. First, a DAB tree is a logical tree not reflecting the physical structure of the sensor network; hence, an edge may consist of multiple communication hops and a high communication cost may be incurred. Second, the construction of the DAB tree does not take the query cost into consideration. Therefore, the result may not be efficient in some cases.

A location management scheme supporting imprecision-tolerant queries for object tracking sensor networks has been studied in [33]. The location information of an object is stored in a centric storage node and a local storage node. When a user intends to know the location of an object, the query will be forwarded from the querying node to the centric storage node of that object. If the precision level is satisfactory, the centric storage node will reply to this query. Otherwise, the query will be forwarded to the local storage node, which has more precise location information of that object. This scheme has two major drawbacks. First, when the querying node is very close to the local storage node of the queried object, the query will still be forwarded to the centric storage node, which may be far from the querying node. Second, only two precision levels are provided.

2.3 MAC Protocols for Wireless Sensor Networks

A significant amount of research effort has been dedicated to the design of MAC protocols for WSNs [1, 20, 22, 24, 27, 29, 30, 35]. The energy efficiency issue has been studied in S-MAC [35] and T-MAC [29] by synchronizing sensors on a common wakeup/sleep schedule. In order to eliminate the synchronization overhead,

B-MAC [20] adopts a preamble sampling technique. Some hybrid TDMA/CSMA MAC protocols have been proposed recently. In Z-MAC [24], some sensors will adopt a CSMA-based MAC protocol and those suffering from high contention will adopt a TDMA-based MAC protocol. By doing so, Z-MAC enjoys the benefits of low latency of CSMA and high channel utilization of TDMA. Funneling-MAC [1] is also a hybrid TDMA/CSMA MAC protocol that aims to solve the funneling effect near the sink. However, these protocols do not address the spatial correlation of sensor data, and thus the contention problem, in event-driven WSNs.

Exploiting the spatial correlation of sensor data on the MAC layer has been discussed in [31], where the relation between the spatial positions of sensors and the event estimation reliability is investigated. Specifically, a distortion function is derived and a term *correlation radius* (R_{corr}) is introduced. Then, CC-MAC (spatial Correlation-based Collaborative Medium Access Control) is proposed. CC-MAC consists of two components: E-MAC (Event MAC) and N-MAC (Network MAC). E-MAC aims to filter out correlated reporting (i.e., determine which sensors can report). On the other hand, N-MAC is mainly used for sensors not in the event area to forward reporting packets. However, CC-MAC has the following drawbacks: (i) E-MAC is a pure contention-based protocol. Although some sensors may withdraw from reporting, those sensors that decide to report will still cause a lot of contention, because they will report simultaneously. (ii) The RTS/CTS mechanism is adopted, which causes high overheads when packet sizes are small. (iii) In CC-MAC, when a sensor x overhears a packet reported by another sensor y , x will judge whether the distance between itself and y is smaller than R_{corr} . If so, x will suspend its report. As to be shown later, this simple condition cannot completely avoid redundant reporting. (iv) The report reduction technique proposed in CC-MAC highly depends on overhearing; thus, redundancy may still exist when one misses overhearing.

Chapter 3

In-network Location Management for the Single-sink System

In this chapter, we present our proposed location management scheme designed for the single-sink sensor networks. We propose a tree structure for in-network object tracking in a sensor network. The location update part of our solution can be viewed as an extension of [14]. In particular, we take the physical topology of the sensor network into consideration. We take a two-stage approach. The first stage aims at reducing the update cost, while the second stage aims at further reducing the query cost. For the first stage, several principles, namely deviation-avoidance and highest-weight-first ones, are pointed out to construct an object tracking tree to reduce the communication cost of location update. Two solutions are proposed: *Deviation-Avoidance Tree* (DAT) and *Zone-based Deviation-Avoidance Tree* (Z-DAT). The latter approach tries to divide the sensing area into square-like zones, and recursively combine these zones into a tree. Our simulation results indicate that the Z-DAT approach is very suitable for regularly deployed sensor networks. For the second stage, we develop a *Query Cost Reduction* (QCR) algorithm to adjust the object tracking tree obtained in the first stage to further reduce the total cost. The way we model this problem allows us to analytically formulate the update and query costs of the solution based on several parameters of the given problem, such as rates that objects cross the boundaries between sensors and rates that sensors are queried. We have also conducted extensive simulations to evaluate

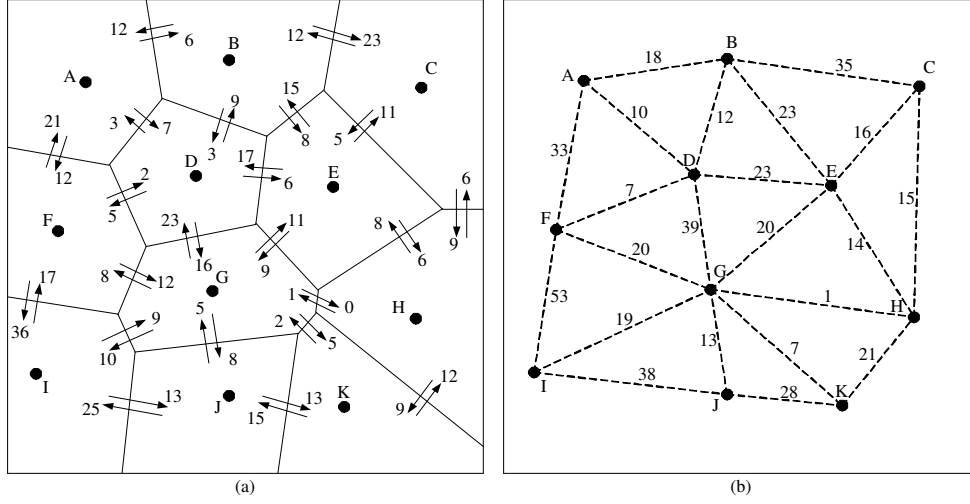


Figure 3.1: (a) The Voronoi graph of a sensor network. (b) The graph G corresponding to the sensor network in (a).

the proposed solutions. The results do validate our observations.

3.1 Preliminaries

We consider a wireless sensor network deployed in a field for the purpose of object tracking. Sensors' locations are already known at a special node, called *sink*, which serves as the gateway of the sensor network to the outside world. We adopt a simple *nearest-sensor* model, which only requires the sensor that receives the strongest signal from the object to report to the sink (this can be achieved by [7]). Therefore, the sensing field can be partitioned into a Voronoi graph [4], as depicted in Fig. 3.1(a), such that every point in a polygon is closer to its corresponding sensor in that polygon than to any other. In practice, a sensor under our model may represent the clusterhead of a cluster of reduced-function sensors. In this work, however, we are only interested in the reporting behavior of these clusterheads.

Our goal is to propose a data aggregation model for object tracking. We assume that whenever an object arrives at or departs from the sensing range (polygon) of a sensor, a *detection event* will be reported (note that this update message

are not always forwarded to the sink, as will be elaborated later). Two sensors are called *neighbors* if their sensing ranges share a common boundary on the Voronoi graph; otherwise, they are *non-neighbors*. Multiple objects may be tracked concurrently in the network, and we assume that from mobility statistics, it is possible to collect the event rate between each pair of neighboring sensors to represent the frequency of objects travelling from one sensor to another. For example, in Fig. 3.1(a), the arrival and departure rates between sensors are shown on the edges of the Voronoi graph. Note that before the statistics is done, the initial weights can be the same value for all edges. In addition, the communication range of sensors is assumed to be large enough so that neighboring sensors (in terms of their sensing ranges) can communicate with each other directly. Thus, the network topology can be regarded as an undirected weighted graph $G = (V_G, E_G)$ with V_G representing sensors and E_G representing links between neighboring sensors. The weight of each link $(a, b) \in E_G$, denoted by $w_G(a, b)$, is the sum of event rates from a to b and b to a . This is because both arrival and departure events will be reported in our scheme. In fact, G is a Delaunay triangulation of the network [4]. Fig. 3.1(b) shows the corresponding Delaunay triangulation of the sensor network in Fig. 3.1(a). Note that the number labelled on each edge represents its weight.

In light of the storage in sensors, the sensor network is able to be viewed as a distributed database. We will exploit the possibility of conducting in-network data aggregation for object tracking in a sensor network. Similar to the approach in [14], a logical weighted tree T will be constructed from G . Note that T may not be a spanning tree in which each node's parent is its neighbor. For example, Fig. 3.2(a) shows an object tracking tree T constructed from the network G in Fig. 3.1(b), where the dotted lines are the forwarding path of a query for Car1. Movement events of objects are reported based on the following rules. Each node a in T will maintain a *detected list* $\mathbf{DL}_a = (L_0, L_1, \dots, L_k)$ such that L_0 is the set of objects currently inside the coverage of sensor a itself, and $L_i, i = 1, \dots, k$, is the set of objects currently inside the coverage of any sensor who is in the subtree rooted at the i -th child of sensor a , where k is the number of children of a .

Table 3.1: Summary of notations used in Chapter 3.

$dist_G(u, v)$	The minimum hop count between u and v in G .
$dist_T(u, v)$	The sum of w_T s of edges on the path connecting u and v in T .
$w_G(u, v)$	The event rate between u and v .
$w_T(u, v)$	The weight of edge (u, v) in T . ($= dist_G(u, v)$).
$lca(u, v)$	The lowest common ancestor of u and v .
$p(v)$	The parent of v in T .
$Subtree(v)$	Members of the subtree rooted at v .
$root(v)$	The root of the temporary subtree containing v during the construction of T .
$q(v)$	The query rate of v .
$neighbors(v)$	Neighbors of v .
$children(v)$	Children of v .

any descendant of sensor i in T . Therefore, searching the location of an object can be done efficiently in T ; a query is only required to be forwarded to a proper subtree and no flooding is needed. For example, Fig. 3.2(a) shows the forwarding path of a query for Car1 in T . Fig. 3.2(b) shows the reporting events as Car1 and Car2 move and the forwarding path of a query for the new location of Car1.

Our goal is to construct an object tracking tree $T = (V_T, E_T)$ that incurs the lowest communication cost given a sensor network $G = (V_G, E_G)$ and the corresponding event rates and query rates, where $V_T = V_G$ and E_T consists of $|V_T| - 1$ edges with the sink as the root. Intuitively, T is a logical tree constructed from G , in which each edge $(u, v) \in T$ is one of the shortest paths connecting sensors u and v in G . Therefore, the weight of each edge (u, v) in T , denoted by $w_T(u, v)$, is modelled by the minimum hop count between u and v in G . The cost function can be formulated as $C(T) = U(T) + Q(T)$, where $U(T)$ denotes the update cost and $Q(T)$ is the query cost.

Table 4.1 summaries the notations used in this chapter.

3.2 Tree Construction Algorithms

This section presents our algorithms to construct efficient object tracking trees. In Section 3.2.1, we develop algorithm DAT targeted at reducing the update cost. Then, in Section 3.2.2, based on the concept of divide-and-conquer, we devise algorithm Z-DAT to further reduce the update cost. In Section 3.2.3, algorithm QCR is developed to adjust the tree obtained by algorithm DAT/Z-DAT to further reduce the total cost.

3.2.1 Algorithm DAT (Deviation-Avoidance Tree)

Object tracking typically involves two basic operations: update and query. Based on the aggregation model in Section 3.1, updates will be initiated when an object o moves from sensor a to sensor b . It can be seen that both the departure event $dep(o, a, b)$ and the arrival event $arr(o, b, a)$ will be forwarded to the root of the minimum subtree containing both a and b . Therefore, the update cost $U(T)$ of a tree T can be formulated by counting the average number of messages transmitted in the network per unit time:

$$U(T) = \sum_{(u,v) \in E_G} w_G(u, v) \times (dist_T(u, lca(u, v)) + dist_T(v, lca(u, v))), \quad (3.1)$$

where $lca(u, v)$ denotes the root of the minimum subtree in T that includes both u and v (from now on, we will call $lca(u, v)$ the lowest common ancestor of u and v), and $dist_T(x, y)$ is the sum of weights of the edges on the path connecting x and y in T . For example in Fig. 3.2(a), $dist_T(F, K) = w_T(F, I) + w_T(I, J) + w_T(J, K) = 3$. In order to identify which factors affecting the value of $U(T)$, we show that $U(T)$ also can be formulated in a different way as follows.

Theorem 1. *Given any logical tree T of the sensor network G , we have*

$$U(T) = \sum_{(p(v), v) \in E_T} \left(w_T(p(v), v) \times \sum_{\substack{(x,y) \in E_G \wedge x \in Subtree(v) \\ \wedge y \notin Subtree(v)}} w_G(x, y) \right), \quad (3.2)$$

where $Subtree(v)$ is the subtree of T rooted at node v and $p(v)$ is the parent of v .

Proof. This can be proved by observing which events will be reported along an edge in T . Given $(p(v), v) \in E_T$, for any $(x, y) \in E_G$ where $x \in Subtree(v)$ and $y \notin Subtree(v)$, since the lowest common ancestor of x and y must not be in $Subtree(v)$, any event generated on (x, y) will be transmitted from v to $p(v)$. Otherwise, no message will be transmitted from v to $p(v)$. This leads to the theorem. \square

From Eq. 3.1 and Eq. 3.2, we make three observations about $U(T)$:

- Eq. 3.1 contains the factor $dist_T(u, lca(u, v))$. Its minimal value is $dist_G(u, lca(u, v))$, which denotes the minimum hop count between sensor u and sensor $lca(u, v)$ in G . Therefore, we would expect that $dist_T(u, sink) = dist_G(u, sink)$ for each $u \in V_G$; otherwise, we say that u deviates from its shortest path to the sink. If $dist_T(u, sink) = dist_G(u, sink)$ for each $u \in V_G$, we say that tree T is a *deviation-avoidance tree*. Fig. 3.3 shows four possible object tracking trees for the graph G in Fig. 3.1(b). The one in Fig. 3.3(b) is not a deviation-avoidance tree since $dist_T(E, A) = 3 > dist_G(E, A) = 2$. The other three are deviation-avoidance trees.
- Eq. 3.2 contains the factor $w_T(u, v)$. Its minimal value is 1 when $u \neq v$. Consequently, it is desirable that each sensor's parent is one of its neighbors. Only the tree in Fig. 3.3(d) satisfies this criterion. By selecting neighboring sensors as parents, the average value of $dist_T(u, lca(u, v)) + dist_T(v, lca(u, v))$ in Eq. 3.1 can be minimized. For example, the average values of $dist_T(u, lca(u, v)) + dist_T(v, lca(u, v))$ are 3.591, 2.864, and 2.227 for the trees in Fig. 3.3(a), 3.3(c), and 3.3(d), respectively.
- In Eq. 3.1, the weight $w_G(u, v)$ will be multiplied by $dist_T(u, lca(u, v)) + dist_T(v, lca(u, v))$. For two edges (u, v) and $(u', v') \in E_G$ such that $w_G(u, v)$

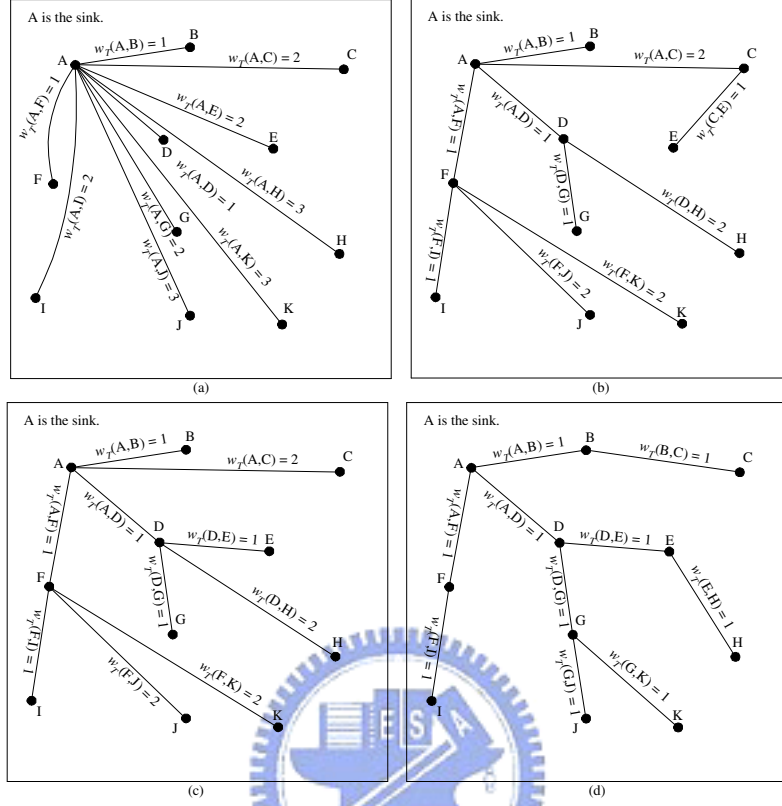


Figure 3.3: Four possible location tracking trees for the graph in Fig. 3.1(b).

$> w_G(u', v')$, it is desirable that $\text{dist}_T(u, \text{lca}(u, v)) + \text{dist}_T(v, \text{lca}(u, v)) < \text{dist}_T(u', \text{lca}(u', v')) + \text{dist}_T(v', \text{lca}(u', v'))$. Combining this observation with the second observation, an edge (u, v) with a higher $w_G(u, v)$ should be included into T as early as possible and $p(v)$ should be set to u if $\text{dist}_G(u, \text{sink}) < \text{dist}_G(v, \text{sink})$, and vice versa. We call this the *highest-weight-first* principle.

Based on above observations, we develop our algorithm DAT. Initially, DAT treats each node as a singleton subtree. Then we will gradually include more links to connect these subtrees together. In the end, all subtrees will be connected into one tree T . The detailed algorithm is shown in Algorithm 1, where notation $\text{root}(x)$ represents the root of the temporary subtree that contains x . To begin with, E_G is sorted into a list L in a decreasing order of links' weights. Based

on the third observation, algorithm DAT will examine edges in L one by one for possibly being included into tree T . For each edge (u, v) in L being examined by algorithm DAT, (u, v) will be included into T only if u and v are currently located in different subtrees. Also, (u, v) will be included into T only if at least one of u and v is currently the root of its temporary subtree and the other is on a shortest path in G from the former node to the sink (these conditions are reflected by the **if** statements in lines 5 and 7). An edge in G passing these checks will then be included into T . Note that without these conditions, deviations may occur. It can be seen that T is always a subgraph of G and $w_T(u, v) = 1$ for all $(u, v) \in E_T$. For example, Fig. 3.4(a) is a snapshot of an execution of DAT. The solid lines are those edges that have been included into T . When (F, G) is examined by DAT, it will not be included into T , because neither F nor G is the root of its temporary subtree. Another snapshot is shown in Fig. 3.4(b). When (B, D) is examined, it will not be included into T . Although D is the root of its temporary subtree, B is not on the shortest path from D to A , i.e., $\text{dist}_G(D, A) \neq \text{dist}_G(B, A) + 1$. (A, D) will be then examined after (B, D) . (A, D) can be included into T , because D is the root of its temporary subtree and A is on the shortest path from D to A .

Algorithm 1 DAT(G)

```

1: Let  $T = (V_T, E_T)$  such that  $V_T = V_G$  and  $E_T = \phi$ 
2: Sort  $E_G$  into a list  $L$  in a decreasing order of their event rates.
3: for each  $(u, v) \in E_G$  in  $L$  do
4:   if ( $\text{root}(u) \neq \text{root}(v)$ ) then
5:     if ( $u = \text{root}(u) \wedge (\text{dist}_G(u, \text{sink}) = \text{dist}_G(v, \text{sink}) + 1)$ ) then
6:       Let  $E_T = E_T \cup (u, v)$  and let the root of the new subtree be  $\text{root}(v)$ .
7:     else if ( $v = \text{root}(v) \wedge (\text{dist}_G(v, \text{sink}) = \text{dist}_G(u, \text{sink}) + 1)$ ) then
8:       Let  $E_T = E_T \cup (u, v)$  and let the root of the new subtree be  $\text{root}(u)$ .
9:     end if
10:   end if
11: end for

```

Theorem 2. *If G is connected, the tree T constructed by algorithm DAT is a connected deviation-avoidance tree rooted at the sink.*

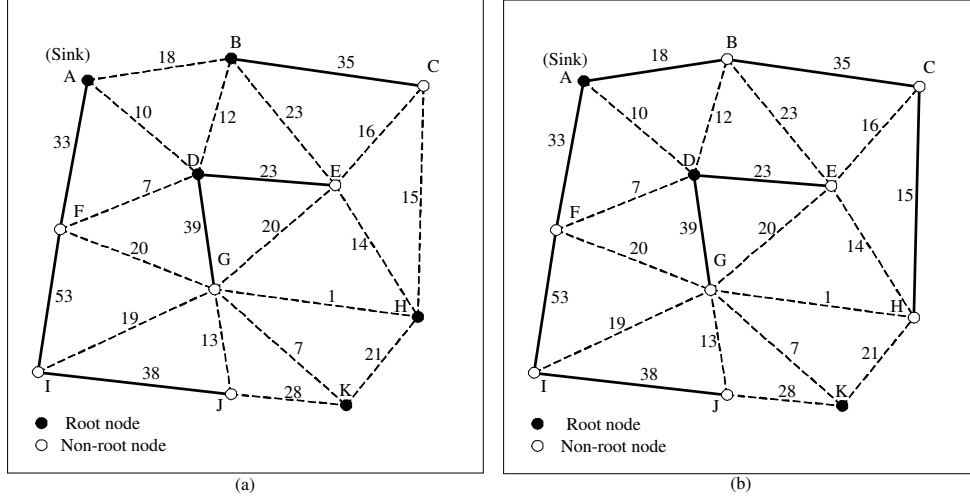


Figure 3.4: Snapshots of an execution of DAT.

Proof. First, we show that T is connected. Each sensor is the root of a singleton subtree in the beginning and we will prove that only one sensor will be the root in the ending. Since G is connected, when a sensor $x \neq \text{sink}$ is the root of a subtree (i.e., $x = \text{root}(x)$), it always can find a neighboring sensor y such that $\text{dist}_G(x, \text{sink}) = \text{dist}_G(y, \text{sink}) + 1$. It is clear that $\text{root}(y) \neq x$, because $\text{dist}_G(\text{root}(y), \text{sink}) \leq \text{dist}_G(y, \text{sink})$. Hence, edge (x, y) can be included into T , and x will not be the root anymore. By repeating such arguments, T must be connected and rooted at the sink. Second, we show that T is a deviation-avoidance tree. This can be derived from two observations. First, when an edge (u, v) is included into T , DAT will choose v as the child of u if $\text{dist}_G(v, \text{sink})$ is larger than $\text{dist}_G(u, \text{sink})$, and vice versa. Therefore, if the path from the sink to sensor u is one of the shortest paths, the path from the sink to sensor v is also one of the shortest paths. Second, assuming $\text{dist}_G(v, \text{sink}) = \text{dist}_G(u, \text{sink}) + 1$, DAT will include (v, u) only when v itself is the root of a subtree. This guarantees that all descendant nodes in $\text{Subtree}(v)$ will not deviate from their shortest paths to the sink. Hence, the theorem follows. \square

3.2.2 Algorithm Z-DAT (Zone-based Deviation-Avoidance Tree)

The Z-DAT is derived based on the following locality concept. Assume that u is v 's parent in T . According to Eq. 3.2, for any edge $(x, y) \in E_G$ such that $x \in Subtree(v)$ and $y \notin Subtree(v)$, arrival/departure events between x and y will cause a message to be transmitted on $(p(v), v)$, thus increasing the value of $\sum_{(x,y) \in E_G \wedge x \in Subtree(v) \wedge y \notin Subtree(v)} w_G(x, y)$. Therefore, the perimeter that bounds the sensing area of sensors in each $Subtree(v)$ will impact the update cost $U(T)$. A longer perimeter would imply more events crossing the boundary. For example, in the three subtrees in Fig. 3.5, although all subtrees have the same number of sensors, the perimeter of the subtree in Fig. 3.5(a) is smaller than that in 3.5(b), which is in turn less than that in 3.5(c). In geometry, it is clear that a circle has the shortest perimeter to cover the same area as compared with other shapes. Circle-like shapes, however, are difficult to be used in an iterative tree construction. As a result, Z-DAT will be developed based on square-like zones.

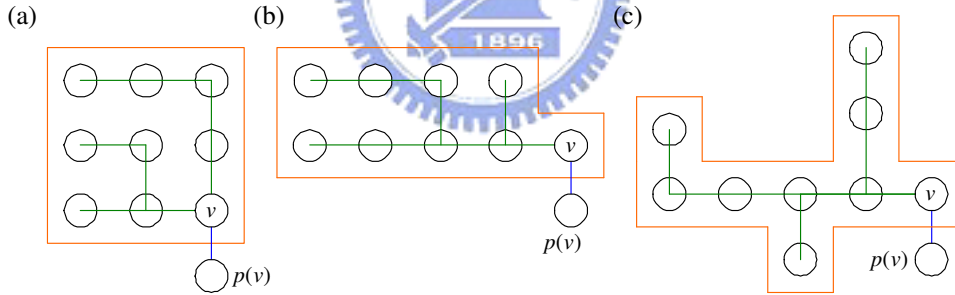


Figure 3.5: Possible structures of subtrees with nine sensors.

Z-DAT is derived based on the deviation-avoidance principle and the above locality concept. The algorithm builds T in an iterative manner based on two parameters, α and δ , where α is a power of 2 and δ is a positive integer. To begin with, Z-DAT first uses $(\alpha - 1)$ horizontal lines to divide the sensing field into α strips. For each horizontal line between two strips, we are allowed to further move it up and down within a distance no more than δ units. This gives $2\delta + 1$ possible locations of each horizontal line. For each location of the horizontal line, we can

calculate the total event rate that objects may move across the line. Then we pick the line with the lowest total event rate as its final location. After all horizontal lines are determined, we then further partition the sensing field into α^2 regions by using $(\alpha - 1)$ vertical lines. Following the adjustment as above, each vertical line is also allowed to move left and right within a distance no more than δ units and the one with the lowest total event rate is selected as its final location.

After the above steps are completed, the sensing field is divided into α^2 square-like zones. First, we run DAT on the sensors in each zone. This will result in one or multiple subtrees in each zone. Next, we will merge subtrees in the above α^2 zones recursively as follows. First, we combine these zones together into $\frac{\alpha}{2} \times \frac{\alpha}{2}$ larger zones, such that each larger zone contains 2×2 neighboring zones. Then we merge subtrees in these 2×2 zones by sorting all inter-zone edges (i.e., edges connecting these 2×2 zones) according to their event rates into a list L and feeding L to steps 3 ~ 11 of the original DAT algorithm. Second, we further combine the above larger zones together into $\frac{\alpha}{4} \times \frac{\alpha}{4}$ even larger zones, such that each even larger zone contains 2×2 neighboring larger zones. This process is repeated until one single tree is obtained. The algorithm is summarized in Algorithm 2. An illustrated example is shown in Fig. 3.6. In the first iteration, we divide the field into $\alpha \times \alpha$ zones and adjust their boundaries according to δ as shown in Fig. 3.6(a). In the second iteration, each 2×2 neighboring zones is combined into a larger zone as shown in Fig. 3.6(b).

Algorithm 2 Z-DAT(G, α, δ)

- 1: Divide the network into $\alpha \times \alpha$ zones based on parameters α and δ .
 - 2: Run DAT on the sensors in each zone.
 - 3: $i \leftarrow 1$
 - 4: **while** $\frac{\alpha}{2^i} \neq 0$ **do**
 - 5: The network is divided into $\frac{\alpha}{2^i} \times \frac{\alpha}{2^i}$ zones.
 - 6: Run DAT on each zone to merge its subtrees.
 - 7: $i \leftarrow i + 1$
 - 8: **end while**
-

To summarize, Z-DAT is similar to DAT except that it examines links of E_G

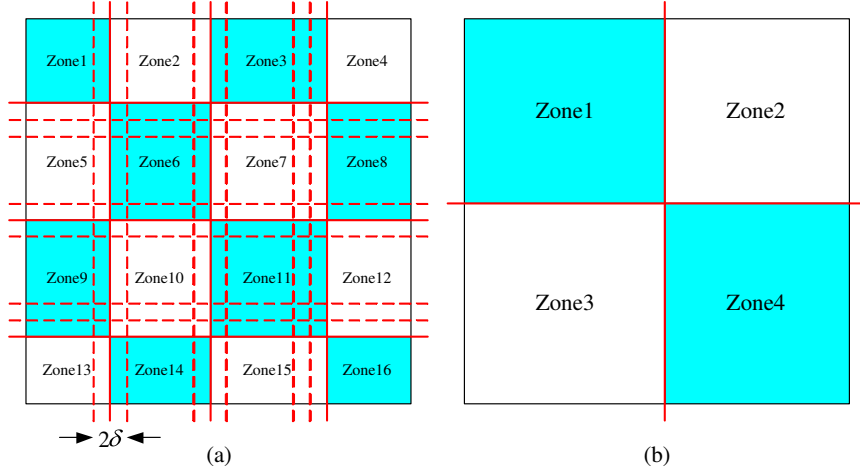


Figure 3.6: An example of the Z-DAT algorithm with $\alpha = 4$.

in a different order. By partitioning the sensing field into zones, each subtree in T is likely to cover a square-like region, thus avoiding the problem pointed out in Fig. 3.5. Also, by using the parameter δ to fine-tune the lowest-level zones, Z-DAT tends to avoid high-weight links becoming inter-zone edges. In fact, this is a consequence of the the highest-weight-first design principle.

Theorem 3. *If G is connected, the tree T constructed by algorithm Z-DAT is a connected deviation-avoidance tree rooted at the sink.*

Proof. Z-DAT will examine all links of G , but in a different order from DAT. However, the proof of Theorem 5 is independent of the order of the links being examined for being included into T . Therefore, the same proof is still applicable here. \square

3.2.3 Algorithm QCR (Query Cost Reduction)

The above DAT and Z-DAT only try to reduce the update cost. The query cost is not taken into account. QCR is designed to reduce the total update and query cost by adjusting the object tracking tree obtained by DAT/Z-DAT. To begin with, we define the query rate $q(v)$ of each sensor v as the average number of queries that refer to objects within the sensing range of v per unit time in statistics.

Given a tree T , we first derive its query cost $Q(T)$. Suppose that an object x is within the sensing range of v . When x is queried, if v is a non-leaf node, the query message is required to be forwarded to v since $p(v)$ only indicates that x is in the subtree rooted at v . On the other hand, if v is a leaf node, the query message only has to be forwarded to $p(v)$, because sensor $p(v)$ knows that the object is currently monitored by v . The following equation gives $Q(T)$ by taking into account the number of hops that query requests and query replies have to travel on T .

$$Q(T) = 2 \times \left(\sum_{\substack{v \in V_T \wedge \\ v \notin \text{leaf node}}} q(v) \times \text{dist}_T(v, \text{sink}) + \sum_{\substack{v \in V_T \wedge \\ v \in \text{leaf node}}} q(v) \times \text{dist}_T(p(v), \text{sink}) \right) \quad (3.3)$$

We make two observations on $Q(T)$. First, because $\text{dist}_T(p(v), \text{sink})$ is always smaller than $\text{dist}_T(v, \text{sink})$, Eq. 3.3 indicates that placing a node as a leaf can save the query cost instead of placing it as a non-leaf. For example, when query rates are extremely high, it is desirable that every node will become a leaf node and T will become a star-like graph. Second, the second term in Eq. 3.3 implies that the value of $\text{dist}_T(p(v), \text{sink})$ should be made as small as possible. Thus, we should choose a node closer to the sink as v 's parent (however, this is at the expense of the update cost).

Based on the above observations, QCR tries to adjust the tree T obtained by DAT or Z-DAT. In QCR, we examine T in a bottom-up manner and try to adjust the location of each node in T by the following operations.

1. If a node v is not a leaf node, we can make it a leaf by cutting the links to its children and connecting each of its children to $p(v)$. (Note that we can do so because T is regarded as a logical tree.) Let T' be the new tree after

modification. We derive that

$$\begin{aligned}
C(T) - C(T') &= Q(T) - Q(T') + U(T) - U(T') = \\
&= 2 \times \left(q(v) + \sum_{\substack{i \in \text{children}(v) \wedge \\ i \in \text{leaf node}}} q(i) \right) \\
&- \sum_{\substack{i \in \text{neighbors}(v) \\ \wedge i \in \text{Subtree}(v)}} w_G(v, i) - \sum_{i \in \text{children}(v)} \left(\sum_{\substack{(x,y) \in E_G \wedge y \notin \text{Subtree}(i) \\ \wedge x \in \text{Subtree}(i)}} w_G(x, y) \right) \\
&+ \sum_{\substack{(x,y) \in E_G \wedge y \notin \text{Subtree}(v) \\ \wedge x \in \text{Subtree}(v) \wedge x \neq v}} w_G(x, y). \quad (3.4)
\end{aligned}$$

If the amount of reduction is positive, we replace T by T' . Otherwise, we keep T unchanged. Fig. 3.7 illustrates this operation.

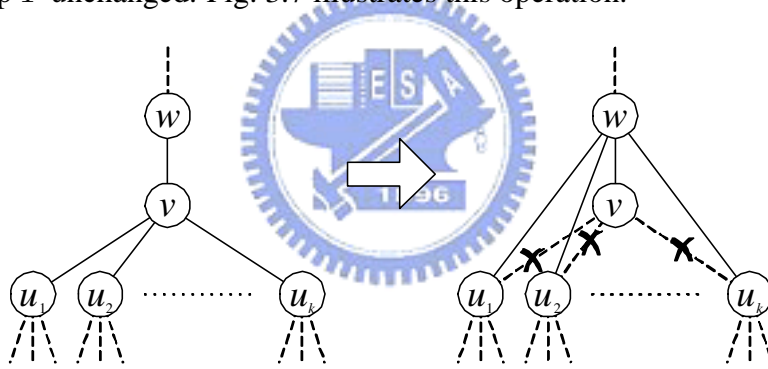


Figure 3.7: Making a non-leaf node v a leaf node.

2. If a node v is a leaf node, we can make $p(v)$ closer to the sink by cutting v 's link to its current parent $p(v)$ and connect v to its grandparent $p(p(v))$. Let T' be the new tree. We derive that

$$\begin{aligned}
C(T) - C(T') &= Q(T) - Q(T') + U(T) - U(T') = \\
&= 2 \times (q(v) + q'(v)) - \left(2 \times \sum_{\substack{(x,y) \in E_G \wedge y \notin \text{Subtree}(v) \wedge \\ x \in \text{Subtree}(v) \wedge y \in \text{Subtree}(p(v))}} w_G(x, y) \right), \quad (3.5)
\end{aligned}$$

where

$$q'(v) = \begin{cases} 0 & \text{if } p(v) \text{ has more than one child in } T \\ q(p(v)) & \text{otherwise} \end{cases}.$$

If the amount of reduction is positive, we replace T by T' . Otherwise, T remains unchanged. Fig. 3.8 illustrates this operation.

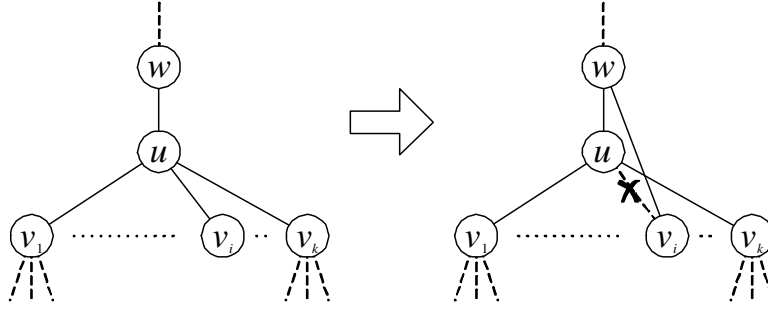


Figure 3.8: Connecting a leaf node v_i to $p(p(v_i))$.

Note that Eq. 3.4 and Eq. 3.5 allow us to compute the reduction of cost without computing $U(T')$ and $Q(T')$. This saves computational overhead. Also note that T is examined in a bottom-up manner in a layer-by-layer manner. Nodes that are moved to an upper layer will have a chance to be reexamined. However, to avoid going back and forth, nodes that are not moved will not be reexamined.

For example, suppose that we are given a DAT tree in Fig. 3.9(a) (which is constructed from Fig. 3.1(b)), where the number labelled on each node is its query rate. When examining the bottom layer, we will apply step 2 to sensors H , J , and K and obtain reductions of 1974, -62 , and -6 , respectively. Hence, only H is moved upward as shown in Fig. 3.9(b). When examining the second layer, we will apply step 1 to sensor G and I and apply step 2 to sensors C , E , and H . Only when applying to sensor H , it will result in a positive reduction of 1970. This updates the tree to Fig. 3.9(c). Finally, sensors B , D , and F are examined. Only D has a positive reduction of 1842. Thus, D will become a leaf and all its children are connected to D 's parent as shown in Fig. 3.9(d). Overall, the cost is reduced from 7124 to 5150, 3180, and then 1338 after each step respectively.

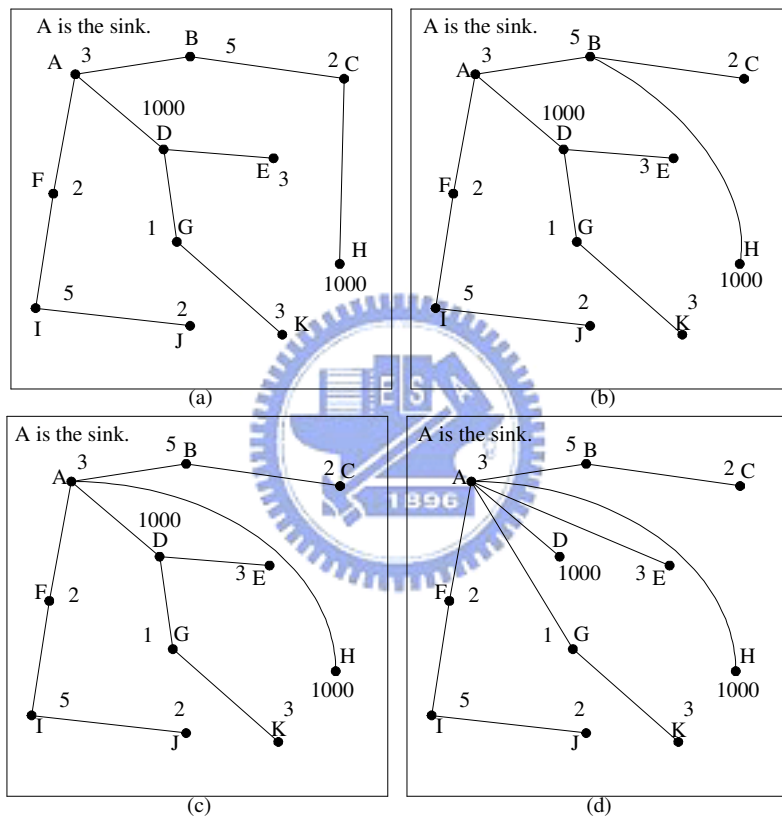


Figure 3.9: An execution example of algorithm QCR.

Finally, we show how to derive Eq. 3.4 and Eq. 3.5. To begin with, we present two implicit facts used in the following derivations. First, according to Theorem 1, we can conclude that if the members of $Subtree(v)$ are not changed, the number of messages transmitted on edge $(v, p(v)) \in T$ will be unchanged. Second, when a node v is being examined by QCR, $w_T(p(v), p(p(v)))$ must be 1. This fact holds because the input of QCR algorithm is a DAT/Z-DAT tree and the tree is examined in a bottom-up manner.

First, we derive the $Q(T) - Q(T')$ in Eq. 3.4. When v becomes a leaf and the queried object locates at the sensing field of v , the query only has to be sent to $p(v)$. In addition, when one of v 's children, say i , is connected to $p(v)$ and i is a leaf, $p(v)$ also can reply the query if the queried object locates at the sensing field of i . Thus, we have

$$Q(T) - Q(T') = 2 \times \left(q(v) + \sum_{\substack{i \in children(v) \\ \wedge i \in leafnode}} q(i) \right).$$

Now we derive the $U(T) - U(T')$ in Eq. 3.4. The operation of QCR ensures that when one of v 's children, say i , changes its parent to $p(v)$, the update cost will be increased by

$$\sum_{i \in children(v)} \left(\sum_{\substack{(x,y) \in E_G \wedge y \notin Subtree(i) \\ x \in Subtree(i)}} w_G(x, y) \right).$$

In addition, the events between v and i , where $i \in neighbors(v)$ and $i \in Subtree(v)$, will be reported to $p(v)$ rather than v when v becomes a leaf. Thus, v must forward an additional message to $p(v)$. The increased cost is

$$\sum_{\substack{i \in neighbors(v) \wedge \\ i \in Subtree(v)}} w_G(v, i).$$

However, when v becomes a leaf, the event across an edge $(x, y) \in E_G$ such that $y \notin Subtree(v)$, $x \in Subtree(v)$, and $x \neq v$ will not be transmitted on $(v, p(v))$.

The cost is reduced by

$$\sum_{\substack{(x,y) \in E_G \wedge y \notin \text{Subtree}(v) \\ \wedge x \in \text{Subtree}(v) \wedge x \neq v}} w_G(x, y).$$

Combining above three factors, we have

$$U(T) - U(T') = - \sum_{\substack{i \in \text{neighbors}(v) \\ \wedge i \in \text{Subtree}(v)}} w_G(x, i) - \sum_{i \in \text{children}(v)} \left(\sum_{\substack{(x,y) \in E_G \wedge y \notin \text{Subtree}(i) \\ \wedge x \in \text{Subtree}(i)}} w_G(x, y) \right) + \sum_{\substack{(x,y) \in E_G \wedge y \notin \text{Subtree}(v) \\ \wedge x \in \text{Subtree}(v) \wedge x \neq v}} w_G(x, y).$$

Next, we derive Eq. 3.5. To see $Q(T) - Q(T')$, observe that when v changes its parent from $p(v)$ to $p(p(v))$, the saved query cost is $q(v)$. Furthermore, when $p(v)$ has only one child v , the adjustment of v will make $p(v)$ a leaf. This saves a query cost of $q(p(v))$. Therefore, we have

$$Q(T) - Q(T') = 2 \times (q(v) + q'(v)).$$

The value of $U(T) - U(T')$ is affected by three factors, when v changes its parent from $p(v)$ to $p(p(v))$. The update cost will be increased by

$$\sum_{\substack{(x,y) \in E_G \wedge y \notin \text{Subtree}(v) \\ \wedge x \in \text{Subtree}(v)}} w_G(x, y).$$

For edges that have one incident vertex in $\text{Subtree}(v)$ and one incident vertex is in $\text{Subtree}(p(v))$ but not in $\text{Subtree}(v)$, the events across these edges cannot be absorbed by $p(v)$ after v changes its parent from $p(v)$ to $p(p(v))$. The increased update cost will be:

$$\sum_{\substack{(x,y) \in E_G \wedge y \notin \text{Subtree}(v) \wedge \\ x \in \text{Subtree}(v) \wedge y \in \text{Subtree}(p(v))}} w_G(x, y).$$

However, for edges that have one incident vertex in $\text{Subtree}(v)$ and one incident vertex is not in $\text{Subtree}(p(v))$, the events across these edges will be transmitted

on $(v, p(p(v)))$ rather than $(v, p(v))$ when we connects v to $p(p(v))$. The update cost will be decreased by

$$\sum_{\substack{(x,y) \in E_G \wedge x \in \text{Subtree}(v) \\ \wedge y \notin \text{Subtree}(p(v))}} w_G(x, y).$$

Combing these terms leads to the following equation

$$\begin{aligned} U(T) - U(T') = & - \sum_{\substack{(x,y) \in E_G \wedge y \notin \text{Subtree}(v) \\ \wedge x \in \text{Subtree}(v)}} w_G(x, y) - \sum_{\substack{(x,y) \in E_G \wedge y \notin \text{Subtree}(v) \wedge \\ x \in \text{Subtree}(v) \wedge y \in \text{Subtree}(p(v))}} w_G(x, y) \\ & + \sum_{\substack{(x,y) \in E_G \wedge x \in \text{Subtree}(v) \\ \wedge y \notin \text{Subtree}(p(v))}} w_G(x, y) = - \left(2 \times \sum_{\substack{(x,y) \in E_G \wedge y \notin \text{Subtree}(v) \wedge \\ x \in \text{Subtree}(v) \wedge y \in \text{Subtree}(p(v))}} w_G(x, y) \right). \end{aligned}$$

3.3 Simulation Results

We have simulated a sensing field of size 256×256 . Unless otherwise stated, 4096 sensors are deployed in the sensing field. Two deployment models are considered. In the first one, sensors are regularly deployed as a 64×64 grid-like network. In the second model, sensors are randomly deployed. In both models, the sink may be located near the center of the network or one corner of the network.

Event rates are generated based on a model similar to the *city mobility model* in [14]. Assuming the sensing field as a square of size $r \times r$, the model divides the field into 2×2 sub-squares called *level-1* subregions. Each level-1 subregion is further divided into 2×2 sub-squares called *level-2* subregions. This process is repeated recursively. Given an object located in any position in the sensing field, it has a probability p_1 to leave its current level-1 subregion, and a probability $1 - p_1$ to stay. In the former case, the object will move either horizontally or vertically with a distance of $r/2$. In the latter case, the object has a probability p_2 to leave its current level-2 subregion, and a probability $1 - p_2$ to stay. Again, in the former case, the object will move either horizontally or vertically with a distance of $r/2^2$, and in the latter case it may cross level-3 subregions. The process

repeats recursively. The probability p_i is determined by an exponential probability $p_i = e^{-C \cdot 2^{d-i}}$, where C is a positive constant and d is the total number of levels. In fact, the above behavior only formulates how objects move in the sensing field. After sensors are deployed in the network (no matter the sensors are deployed in a regular or random way), the movement patterns of these objects will generate event rates between neighboring sensors. Also, objects are queried by the sink with the same probability. Since objects may be located at different sensors with different probabilities, the query rates may vary in different sensors.

We compare our schemes with a naive scheme and the DAB scheme [14]. In the naive scheme, any update is sent to the sink (i.e., there is no in-network processing capability.) In this case, the query cost is always zero, so it is preferable when the query rates are relatively high. For the DAB scheme, all sensors are considered leaf nodes, and a logical structure is used to connect these leaf nodes. When two subtrees are merged into one, the root of the subtree which is closer to the sink will become the root of the merged tree (note that this may still cause deviation).

First, we observe the advantage of using in-network processing to reduce update cost. Fig. 3.10 shows the result under different values of C for regular and random sensor deployment. Note that (α, δ) is set to $(8, 0)$ for the Z-DAT. As can be seen, a larger C implies a higher moving locality, thus leading to a lower update cost. The naive scheme has the highest update cost, which is reasonable. By exploiting the concept of deviation avoidance and taking the physical topology into account, DAT and Z-DAT further outperform DAB.

Next, we investigate the effect of deployment models. By comparing, the graphs in Fig. 3.10, we see that Z-DAT outperforms DAT under regular deployment, but the advantage is almost negligible under random deployment. This is because maintaining the shapes of subtrees in Z-DAT is difficult. For example, Fig. 3.11 shows snapshots of DAT trees and Z-DAT trees under regular and random deployments. Note that in this experiment, we assume that there are only 1024 sensors with the sink at the lower-left corner and (α, δ) is set to $(8, 0)$ for the

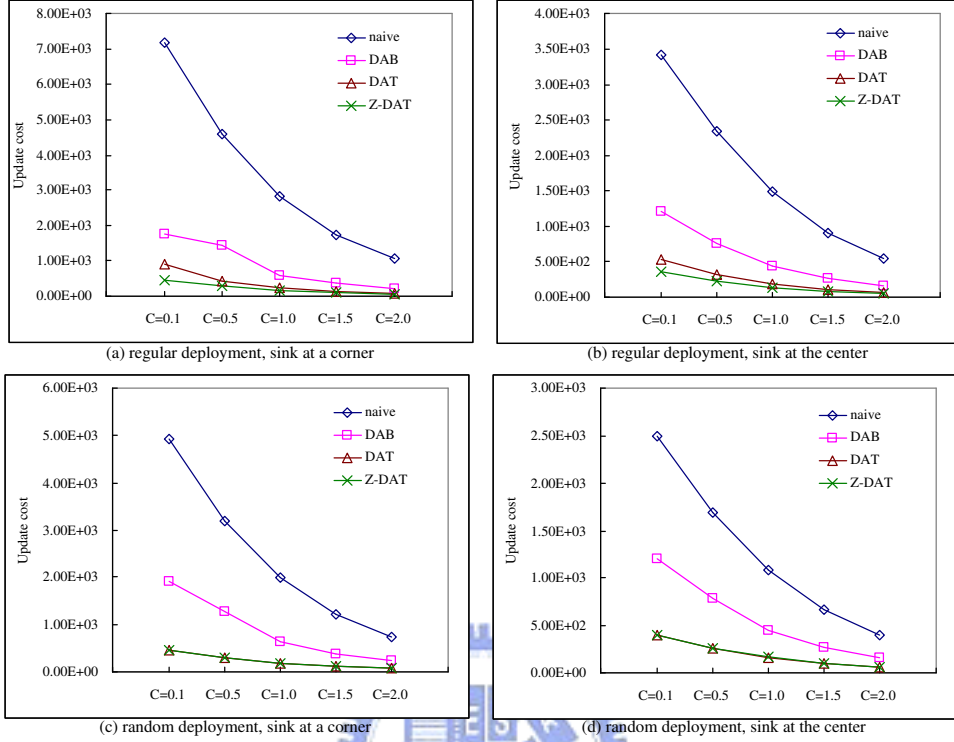


Figure 3.10: Comparison of update costs.

Z-DAT. As can be seen, Z-DAT does exploit the locality of sensors by partitioning sensors into zones under regular deployment. However, this is not true for the random case.

To get further insight into the performance of Z-DAT, we vary α and δ , and show the results in Fig. 3.12, where a 4096- and a 2500-node sensor networks are simulated and sinks are located at the center of the network. Note that when $\alpha = 1$ and $\delta = 0$, Z-DAT is equivalent to DAT. For regular deployment, Z-DAT performs well when α is larger than 4. However, for random deployment, the Z-DAT does not perform well, because maintaining the shapes of subtrees in Z-DAT is difficult. Furthermore, it can be seen that when $\delta = 0$, Z-DAT has better performance. This means that a square-like zone is better than a rectangle-like zone. Also, note that the trend in both 4096- and 2500-node sensors networks (the latter has a non-power-of-2 number of nodes) are quite similar.

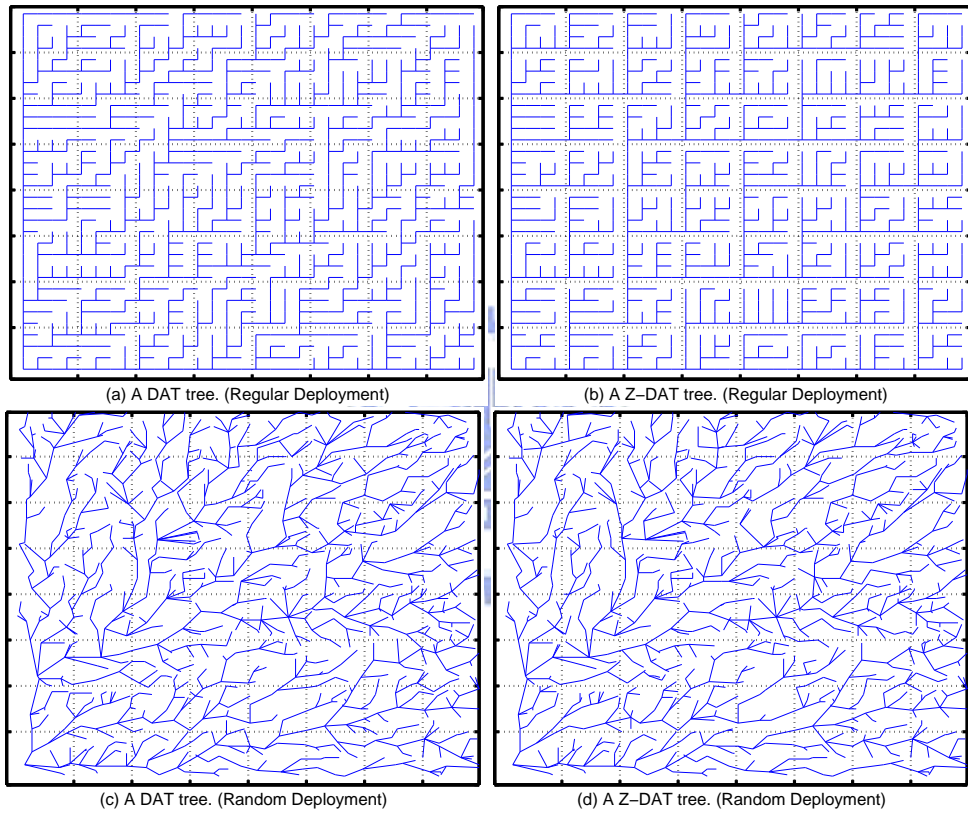


Figure 3.11: Snapshots of tree T obtained by DAT and Z-DAT under regular and random deployments.

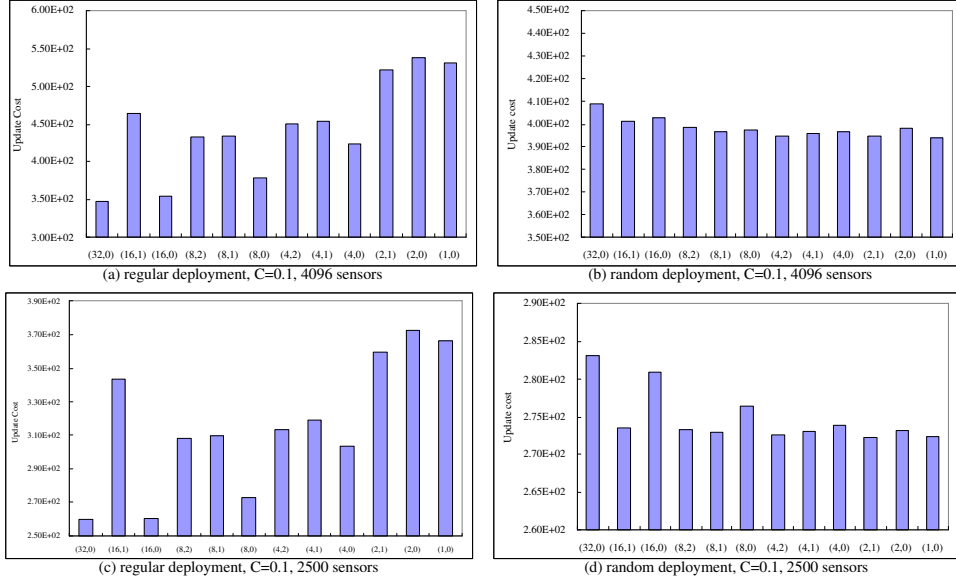


Figure 3.12: Comparison of update costs under different (α, δ) for Z-DAT.

Next, we examine the query cost. The result is shown in Fig. 3.13. In general, the query cost increases linearly with the aggregate query rate. As mentioned earlier, the query cost of the naive scheme is always zero. Both query costs for DAT and Z-DAT are lower than that of DAB. This is attributed to the fact that query messages are always transmitted along the shortest paths between the sink and sensors in DAT and Z-DAT. Also due to the similar reason, the query cost is independent of the shape of T ; thus, DAT and Z-DAT perform similarly despite the deployment models.

Finally, we examine the effectiveness of algorithm QCR by showing the total update and query costs of different schemes in Fig. 3.14. (For visual clarity, the cost of DAT are not shown.) The naive scheme has a constant cost because it is not affected by the query rate. The costs of DAB and Z-DAT increase linearly with respect to the query rate. As a result, they are outperformed by the naive scheme after the query rate reaches a certain level. Our Z-DAT with QCR scheme performs the best at all query rates. When the query rate is low, it performs close to Z-DAT. On the other hand, when the query rate increases, it works similar to

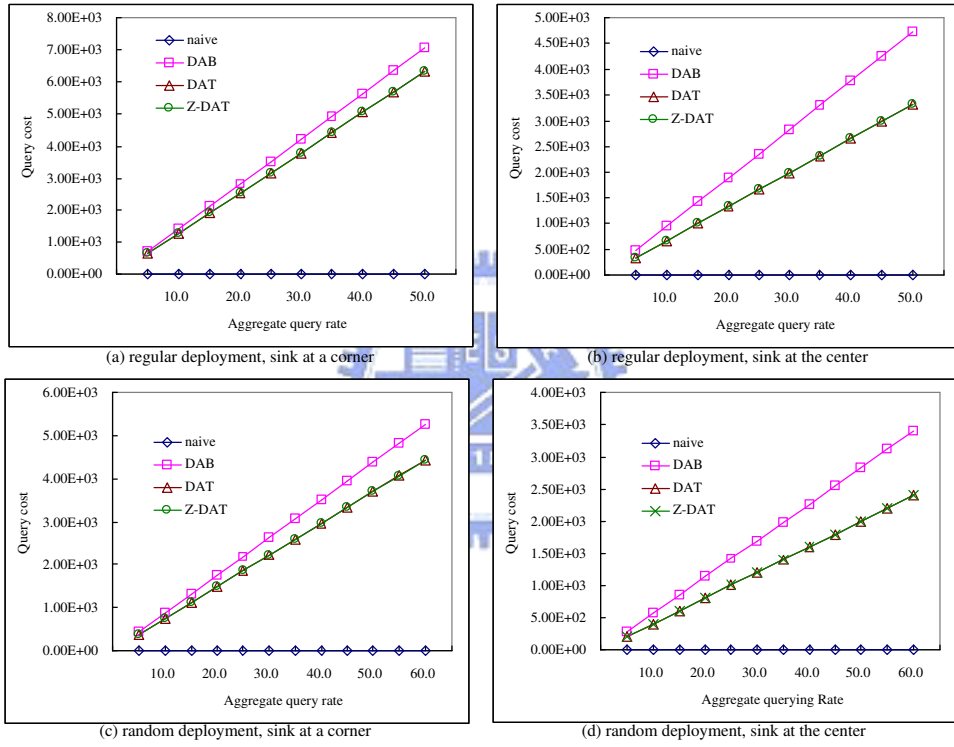


Figure 3.13: Comparison of query costs. ($C = 1.0$)

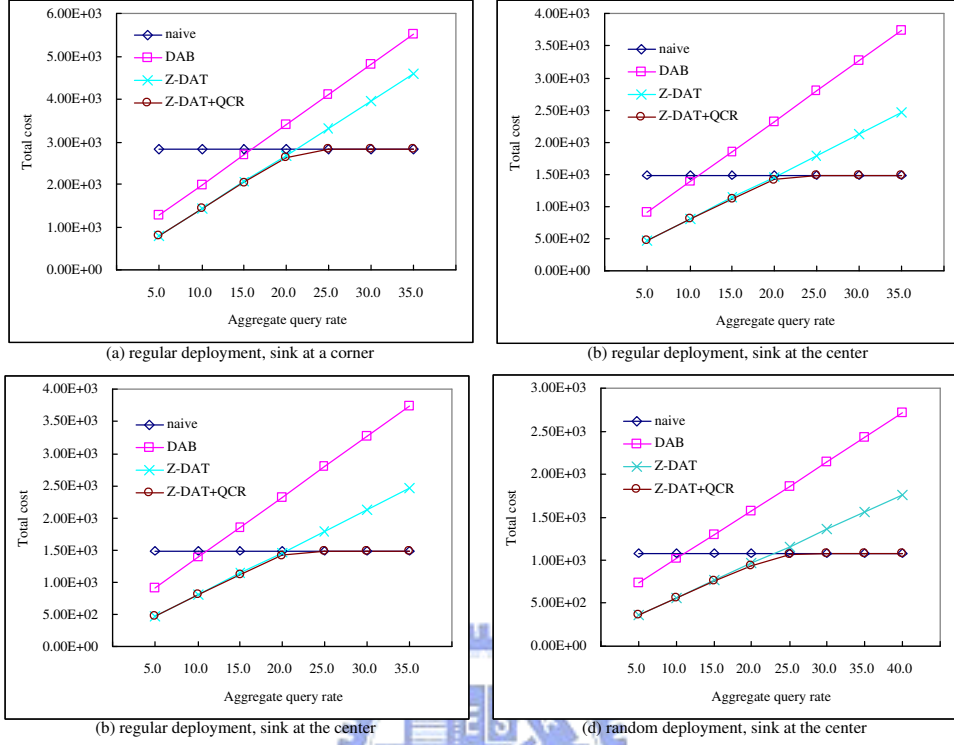


Figure 3.14: Comparison of total costs. ($C = 1.0$)

the naive schemes. This verifies the advantage of the proposed DAT/Z-DAT with QCR schemes.

3.4 Summary

In this chapter, we have developed several efficient ways to construct a logical object tracking tree in a single-sink sensor network. We have shown how to organize sensor nodes as a logical tree so as to facilitate in-network data processing and to reduce the total communication cost incurred by object tracking. For the location update part, our work can be viewed as the extension of the work in [14], and we enhance the work by exploiting the physical structure of the sensor network and the concept of deviation avoidance. In addition, we also consider the query operation and formulate the query cost of an object tracking tree given the query rates

of sensors. In particular, our approach tries to strike a balance between the update cost and query cost. Performance analyses are presented with respect to factors such as moving rates and query rates. Simulation results show that by exploiting the deviation-avoidance trees, algorithms DAT and Z-DAT are able to reduce the update cost. By adjusting the deviation-avoidance trees, algorithm QCR is able to significantly reduce the total cost when the aggregate query rates is high, thus leading to efficient object tracking solutions.



Chapter 4

In-network Location Management for the Multi-sink System

In the previous chapter, it is assumed that there is only one sink in the network. In this chapter, we explore the possibility of having multiple sinks in the network. One advantage of having multiple sinks is to reduce the response time of queries. In addition, using multiple sinks can also relieve the traffic congestion problem associated with a single-sink system (i.e., using multiple sinks can achieve load balance more easily). In order to support location management in a multi-sink wireless sensor network, we can extend the tree structure used in the single-sink system by constructing a logical tree for each sink. However, this implies that updating multiple trees is required when a movement event is detected. Assuming that there are m sinks coexisting in the network, if each tree is updated independently, the update cost will become approximately m times. It is desirable to further reduce the update cost when multiple trees coexist in the network. In this chapter, by exploring the concept of data aggregation, we propose an algorithm to efficiently update multiple trees. With proper design, we show that the update cost increases slightly when the number of trees (i.e., the number of sinks) increases. Based on the foregoing update algorithm, we formulate the update cost that gives us hints to develop efficient tree-construction algorithms. Two distributed multi-tree construction algorithms are presented in this chapter. Experimental results show that the increased update cost with multiple trees can be compensated by

lower query cost and the query cost also depends on m , the number of sinks. This allows us to further investigate how to choose the value of m under different scenarios.

4.1 Preliminaries

4.1.1 Network Model

The network model used in this chapter is the same with that used in the single-sink system. We consider a WSN to be used for object tracking. We adopt a simple *nearest-sensor tracking* model, in which the sensor that receives the strongest signal from an object is responsible for tracking the object (this can be achieved by [7] and we omit the details). Therefore, the sensing field can be modelled by a *Voronoi graph* [4], where each sensor's responsible area is the polygon containing itself. Two sensors are called *neighbors* if their sensing ranges share a common boundary on the Voronoi graph. Multiple objects may be tracked concurrently by the network, and we assume that from mobility statistics, it is possible to collect the frequency that objects move between each pair of neighboring sensors, called the *event rate*.

4.1.2 From Single-sink to Multi-sink WSNs

In the previous chapter, an in-network location management scheme for a single-sink sensor network is proposed. First, a tree T rooted at the sink is constructed. If an object moves from one sensor to another, update messages will be forwarded to the lowest common ancestor of these two nodes in T . For example, in Fig. 4.1, a tree rooted at sensor A is constructed from the G shown in Fig. 3.1(b). When *Car1* moves from H to C , update messages will be forwarded from H to B and from C to B respectively. This allows each node x to always keep a fresh list of objects that are currently tracked by each of the subtrees rooted at x 's children. When a user in F inquires *Car1*'s location, the query will be sent to the sink first

and then forwarded along a path of the tree according to the lists maintained by sensors, as shown in Fig. 4.1.

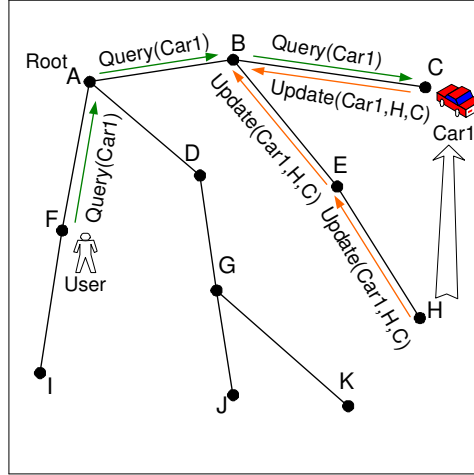


Figure 4.1: An example of the single-sink system.

In this chapter, we assume that multiple sinks coexist in G . Our goal is to reduce the number of messages transmitted for update and query. A naive way to extend a single-sink system to a multi-sink system is to construct a virtual tree $T_x = (V_G, E_{T_x})$ for each sink x , where $E_{T_x} \subseteq E_G$. For example, Fig. 4.2(a) extends the network in Fig. 4.1 such that both sensors A and B are sinks. Three issues should be addressed when multiple trees coexist.

1. **Update and query mechanisms:** When an object moves, updating multiple trees is required in a multi-sink system. If we apply the same update mechanism used in a single-tree system to each tree independently, the update cost will increase approximately m times, where m is the number of trees. This is apparently inefficient. Therefore, update aggregation should be done to reduce the update cost in a multi-sink system. Further, the query mechanism should be designed carefully. We will show later that the query paths from sinks to the target sensor may cause a cycle. The cycle problem should be avoided.

2. **Multi-tree construction:** Our proposed update and query mechanisms can be applied to any multi-tree system. However, different multi-tree construction algorithms will cause different update costs. We will formulate the update cost and point out the factors that affect the update cost. Then, we propose two efficient distributed multi-tree construction algorithms.
3. **The number of trees used:** Obviously, using multiple trees will increase update cost; however, the increase can be compensated by lower query cost (this will be verified further through simulation). Because both the update cost and the query cost are affected by the number of trees used, we will investigate the proper value of m under various scenarios.

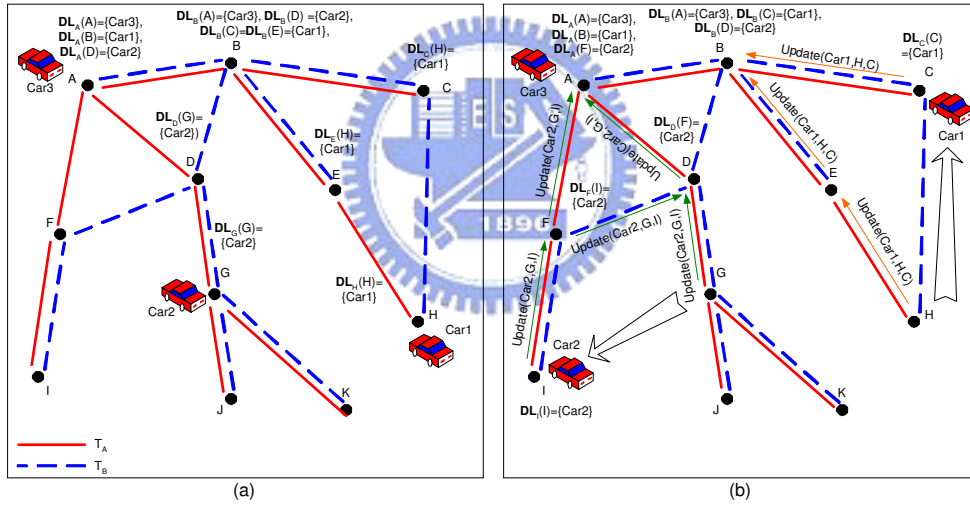


Figure 4.2: (a) The DL s stored in sensors. (b) An example where $Car2$ moves from G to I and $Car1$ moves from H to C .

4.2 Update and Query Mechanisms in Multi-Sink WSNs

4.2.1 Notations and Data Structures

We consider a WSN with n sensors, m of which (denoted by $\sigma_i, i = 1, \dots, m$) are designated as sinks. For each sink σ_i , we assume that a tree T_{σ_i} rooted at σ_i has been constructed from G . Table 4.1 summaries the notations used in this chapter. Then, we introduce the data structures used in this chapter. Moreover, each sensor x will keep two tables in order to process updates and queries:

- *Subtree_Member* S_x : It is an $m \times n$ table to indicate whether another sensor is a descendant of x in a certain tree. Specifically, $S_x(T_{\sigma_i}, j) = 1$ if sensor j is a descendant of x in tree T_{σ_i} ; otherwise, $S_x(T_{\sigma_i}, j) = 0$. For example, in Fig. 4.2(a), $S_D(T_B, F) = 1$ and $S_D(T_A, F) = 0$. All values in this table will not change after all trees are through with construction.
- *Detected_List* DL_x : It is a table with $k + 1$ entries, where k is the number of neighbors of x . Each entry maintains a set of objects. For sensor x itself, $DL_x(x)$ contains the objects currently being tracked by x . For each neighbor y of x , $DL_x(y)$ contains all objects that are currently being tracked by the subtrees of some $T_{\sigma_i}, i = 1, \dots, m$, rooted at y , i.e., $DL_x(y) = \{o | \exists z, i \text{ s.t. } o \in DL_z(z), S_y(T_{\sigma_i}, z) = 1, \text{ and } x = p_i(y)\}$. This implies that if o is tracked by sensor z currently and y is an ancestor of z in a certain tree, then x can know how to find o by asking y . For example, in Fig. 4.2(a), D is a neighbor of A . Because $S_D(T_A, G) = 1$ and $Car2$ is tracked by G , $Car2 \in DL_A(D)$. (Note that in Fig. 4.2(a) entries with empty set are not shown.) *Detected_List* is a dynamic table. When an object moves from one sensor to another, some sensors' *Detected_Lists* have to be modified accordingly.

Table 4.1: Summary of notations used in Chapter 4.

$dist_G(u, v)$	The minimum hop count between u and v in G .
$nei(v)$	The neighbors of v in G .
$dist_{T_{\sigma_i}}(u, v)$	The hop count of the path connecting u and v in T_{σ_i} .
$w_G(u, v)$	The event rate between u and v .
$lca_i(u, v)$	The lowest common ancestor of u and v in T_{σ_i} .
$p_i(v)$	The parent of v in T_{σ_i} .
σ_i	The root of T_{σ_i} .

4.2.2 The Location Update Mechanism

The goal of location update is to ensure that the *Detected_Lists* of sensors are fresh. The main idea here is that when an object o moves from sensor a 's responsible polygon to sensor b 's responsible polygon, for each sink σ_i , the update messages should be sent from a and b along T_{σ_i} to $lca_i(a, b)$, the lowest common ancestor of a and b in T_{σ_i} . The reason for doing so is that the *Detected_Lists* of the ancestors of $lca_i(a, b)$ will not be affected by this movement. Furthermore, instead of allowing all trees to update independently, we will update trees simultaneously with some data aggregation techniques. We make the following observation. In a system with m trees, a sensor x needs to maintain $p_i(x)$ for each T_{σ_i} , $i = 1, \dots, m$. Because the number of neighbors of x may be smaller than m , some of the $p_i(x)$ s may be duplicate and thus can be updated together. This also implies that when a node y receives an update message, node y should update its *Detected_List* by considering several trees rather than one tree. Thus, the update mechanism comprises two parts: (1) the forwarding rule of the update message, and (2) the updating rule of the *Detected_List*. Furthermore, the update message sent for the event that an object o moves from sensor a to sensor b is denoted by $Update(o, a, b, eventid)$, where $eventid$ is to uniquely represent this event.

Forwarding Rule: When an object o moves from sensor a to sensor b , for each tree T_{σ_i} , every node on the tree paths from a to $lca_i(a, b)$ and from b to $lca_i(a, b)$ should receive the update message at least once. Note that if a node x is on the path from a to $lca_i(a, b)$ in T_{σ_i} and $x \neq lca_i(a, b)$, then $S_x(T_{\sigma_i}, a) = 1$

and $S_x(T_{\sigma_i}, b) = 0$. Similarly, if x is on the path from b to $lca_i(a, b)$ in T_{σ_i} and $x \neq lca_i(a, b)$, then $S_x(T_{\sigma_i}, a) = 0$ and $S_x(T_{\sigma_i}, b) = 1$. If x is $lca_i(a, b)$, then $S_x(T_{\sigma_i}, a) = 1$ and $S_x(T_{\sigma_i}, b) = 1$. Thus, when any node x receives a new $Update(o, a, b, eventid)$ message, node x can use the following statement to determine whether it is on the tree paths from a to $lca_i(a, b)$ or from b to $lca_i(a, b)$:

$$\begin{aligned} \exists i((S_x(T_{\sigma_i}, a) = 0 \wedge S_x(T_{\sigma_i}, b) = 1) \vee \\ (S_x(T_{\sigma_i}, a) = 1 \wedge S_x(T_{\sigma_i}, b) = 0)) \end{aligned} \quad (4.1)$$

(Note that Eq. 4.1 includes the special cases of $x = a$ and $x = b$, in which the movement of o rather than receiving an update message will make x checking Eq. 4.1.) If x receives the update message for the first time and there is a tree T_{σ_i} making Eq. 4.1 true, then an update message should be sent to $p_i(x)$. However, if two trees T_{σ_i} and T_{σ_j} both satisfy Eq. 4.1 and $p_i(x) = p_j(x)$, then only one update message needs to be sent (the same applies if multiple trees satisfy Eq. 4.1). This is what we mean by update aggregation.

Updating Rule: When a node is notified that an object o moves from sensor a to sensor b , it will update its *Detected List* as follows.

- For sensor a , it will remove o from $DL_a(a)$ and check whether there exist a tree T_{σ_i} and a neighbor y such that $S_a(T_{\sigma_i}, b) = 1$ and $a = p_i(y)$. If the answer is affirmative, this implies that a can find o by asking y . Thus, it adds o into $DL_a(y)$.
- For sensor b , it will add o into $DL_b(b)$ and remove o from other entries of DL_b if o appears in other entries.
- For any other sensor x that receives the update message from y , if $\exists i(S_x(T_{\sigma_i}, b) = 1 \wedge x = p_i(y))$ is true, this implies that x can find o by asking y ; thus o will be added to $DL_x(y)$. Otherwise, o will be removed from $DL_x(y)$ if o appears in $DL_x(y)$.

Fig. 4.2(b) shows an example where *Car2* moves from *G* to *I* and *Car1* moves from *H* to *C*. The modified *DLs* and the reported messages are also shown in Fig. 4.2(b). Our update scheme ensures that when an object *o* moves from one sensor to another, if no packet loss happens and the update procedure can be completed before *o* moves to another sensor, then the freshness of *Detected Lists* of sensors can be guaranteed.

Next, we derive the number of messages required to be sent per unit time for location update as follows.

$$U = \left(\sum_{i=1}^m U(T_{\sigma_i}) \right) - \left(\sum_{v \in V_G} SC(v) \right), \quad (4.2)$$

where $U(T_{\sigma_i})$ is the update cost for tree T_{σ_i} if T_{σ_i} is the only tree in the network and $SC(v)$ is the saved cost for sensor v due to the overlap of tree edges among m trees. $U(T_{\sigma_i})$ can be formulated as

$$U(T_{\sigma_i}) = \sum_{\substack{(u,v) \in E_G \wedge \\ (u,v) \notin E_{T_{\sigma_i}}}} (w_G(u,v) \times (dist_{T_{\sigma_i}}(u, lca_i(u,v)) + dist_{T_{\sigma_i}}(v, lca_i(u,v)))), \quad (4.3)$$

where $dist_{T_{\sigma_i}}(x, y)$ is the hop count of the path connecting x and y in T_{σ_i} . To explain the meaning of Eq. 4.3, we assume that T_{σ_i} is the only tree in the network. When an event occurs on (u, v) , the update messages will be forwarded to $lca_i(u, v)$ according to the forwarding rule. Eq. 4.3 is similar to the cost function for a single tree in [16], except that when $(u, v) \in E_{T_{\sigma_i}}$ there is no cost because either u or v is $lca_i(u, v)$ and thus no update message has to be sent. This leads to Eq. 4.3. The formulation of $SC(v)$ depends on the forwarding schemes. Two forwarding schemes are considered: the broadcast scheme and the unicast scheme. Due to the broadcast nature of wireless radio, when a sensor sends an update message, we assume all its neighbors will receive the update message in the broadcast

scheme. In this case, $SC(v)$ can be formulated as

$$SC(v) = \sum_{i=2}^m \left((i-1) \times \sum_{\substack{(s,t) \in E_G \wedge \\ f(s,t,v)=i}} w_G(s,t) \right), \quad (4.4)$$

where $f(s,t,v)$ represents the number of trees, each of which, say T_{σ_j} , makes the following statement true $((s,t) \neq (v, p_j(v))) \wedge ((S_v(T_{\sigma_j}, s) \wedge \neg S_v(T_{\sigma_j}, t)) \vee (S_v(T_{\sigma_j}, t) \wedge \neg S_v(T_{\sigma_j}, s)))$. Intuitively, this means that when an object moves from s to t or from t to s , v will broadcast an update message to its neighbors for updating tree T_{σ_j} and this broadcast message can update these $i (=f(s,t,v))$ trees simultaneously; therefore, $(i-1)$ messages are saved. This leads to Eq. 4.4.

However, the packet transmission is unreliable in the broadcast scheme. Once the update messages are lost during the transmission, *Detected Lists* may not contain up-to-date information, resulting in the failures of queries. Thus, one also can adopt the unicast scheme to forward update messages in which each update message has a designated destination. In this case, $SC(v)$ can be formulated as

$$SC(v) = \sum_{u \in nei(v)} \left(\sum_{i=2}^m \left((i-1) \times \sum_{\substack{(s,t) \in E_G \wedge \\ g(s,t,v,u)=i}} w_G(s,t) \right) \right), \quad (4.5)$$

where $nei(v)$ denotes the neighbors of v in G and $g(s,t,v,u)$ represents the number of trees, each of which, say T_{σ_j} , makes the following statement true $(u = p_j(v)) \wedge ((s,t) \neq (v,u)) \wedge ((S_v(T_{\sigma_j}, s) \wedge \neg S_v(T_{\sigma_j}, t)) \vee (S_v(T_{\sigma_j}, t) \wedge \neg S_v(T_{\sigma_j}, s)))$. Eq. 4.5 is similar to Eq. 4.4 except that each of v 's neighbors is considered separately. Though the unicast scheme can provide reliable transmission using acknowledgement mechanisms, the number of saved packets is smaller than that in the broadcast scheme. We will compare the performances of the broadcast scheme and the unicast scheme through simulation in which packet loss will be simulated. Eq. 4.3, Eq. 4.4 and Eq. 4.5 will give us hints for constructing message-efficient multiple virtual trees.

4.2.3 The Location Query Mechanism

Now, we describe our location query mechanism. We assume that a user can issue a query from any sensor. When a sensor x receives a query for object o , there are two scenarios: (1) o does not appear in any of the entries of DL_x , and (2) o appears at least in one of the entries of DL_x .

In the first scenario, x will forward the query to the closest sink, say σ_j , in order to inquire o 's location. The reason for doing so is that, for each sink σ_i , it can be easily shown that all objects tracked by the network will be contained in DL_{σ_i} . However, on the query's way to sink σ_j , if an intermediate node y finds that o appears in DL_y , then the second scenario will be initiated immediately.

In the second scenario, we will show how x can forward the query to locate o . We can model the WSN responsible for tracking object o as a directed *query graph* $G'_o = (V_G, E_{G'_o})$, where a directed edge $(u, v) \in E_{G'_o}$ if and only if $o \in DL_u(v)$. Our location update mechanism guarantees that if x forwards the query along the query graph G'_o , then o is always reachable. For example, Fig. 4.3(a) shows the query graph G'_{Car1} of Fig. 4.2(a) for *Car1*, where A and B are sinks. It means that x can simply forward the query to any y such that $o \in DL_x(y)$. This is repeated until a sensor z such that $o \in DL_z(z)$ is reached. However, the fact that o is reachable via y from x in G'_o does not necessarily imply that G'_o is cycle-free when multiple trees coexist in the network. For example, Fig. 4.3(b) shows two trees T_A and T_B and Fig. 4.3(c) shows the query graph for *Car1*, which have a cycle containing D , F , and G . A query forwarded as above may loop infinitely.

A simple way to solve the infinite loop problem is to force a query to always travel along a designated tree. In order to achieve this, we can add a field *tree_index* to the query request. Once the *tree_index* is set by a certain sensor, the following sensors can follow the tree designated by *tree_index*. Here, we propose an alternative solution which imposes that all trees be shortest-path trees. If so, not only the query and update paths can be shortest, but also the corresponding G'_o for each object o is always cycle-free. Thus, our query mechanism will work

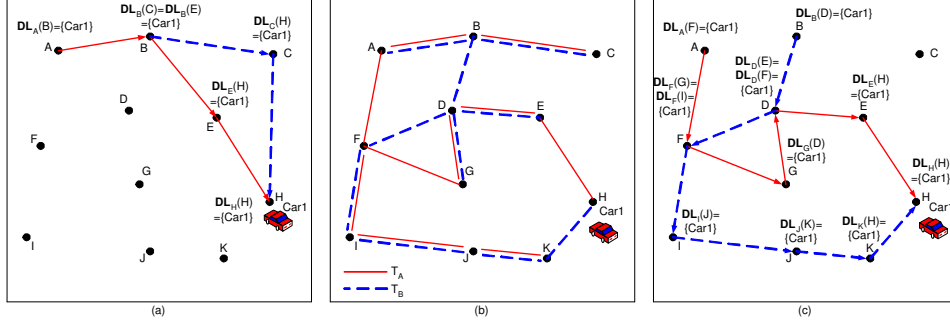


Figure 4.3: (a) The query graph G'_{Car1} of Fig. 4.2(a) for $Car1$. (b) Another example of a two-tree system. (c) The query graph of (b) for $Car1$, which contains a cycle.

correctly.

Theorem 4. *If all trees are shortest-path trees, the query graph G'_o for each object o tracked by the network must be cycle-free.*

Proof. Without loss of generality, we assume o is tracked by sensor x currently. For the purpose of contradiction, we assume that all trees are shortest-path trees but a cycle $\langle c_0, c_1, \dots, c_k, c_0 \rangle$ exists in G'_o . Let c_j be the vertex in the cycle with minimum $dist_G(x, c_j)$. The fact that (c_j, c_{j+1}) is an edge in the cycle implies that $o \in DL_{c_j}(c_{j+1})$. This means that there exists a tree, say T_{σ_i} that contains the edge (c_j, c_{j+1}) , which can lead to x . Because $dist_G(x, c_{j+1}) \geq dist_G(x, c_j)$, T_{σ_i} must not be a shortest-path tree. This contradicts our assumption that all trees are shortest-path trees. Therefore, G'_o must not contain a cycle. \square

After the query reaches the sensor currently tracking the queried object, the sensor can reply to the sensor initiating the query through a shortest path. In the case that the user is capable of mobility, the user should update with the initiating sensor its position until a reply is received. This would solve the mobility problem.

4.3 Multi-Tree Construction Algorithms

The above derivations have suggested that trees rooted at sinks should be shortest-path trees to avoid the cycle problem. In addition, following the derivation of Eq. 4.2, these trees should be constructed carefully to reduce communication costs. Below, we propose two distributed multi-tree construction algorithms, given $\sigma_1, \sigma_2, \dots, \sigma_m$ as the sinks.

4.3.1 The MT-HW Algorithm

From Eq. 4.3, we observe that when an edge (u, v) becomes an edge of T_{σ_i} , the events occurring on (u, v) do not cause any message to be reported for updating T_{σ_i} . Therefore, in MT-HW (multi-tree construction with the high-weight-first property) algorithm, an edge (u, v) with higher weight will be considered for being included into a tree earlier.

First, we define the term *candidate parents*. A sensor y is called a candidate parent of x for sink σ_i , if y is x 's neighbor and $dist_G(\sigma_i, x) = dist_G(\sigma_i, y) + 1$. We assume that when the network is initiated, each sink σ_i will flood a message in the network, which helps each sensor x to derive $dist_G(\sigma_i, x)$ and thus x 's candidate parents. The MT-HW algorithm works as follows. Each sensor x will sort its neighbors in a decreasing order according to the event rates between it and its neighbors. Then, for each sink σ_i , x will pick one neighbor y as its parent that has the highest event rate among x 's candidate parents for σ_i and set $y = p_i(x)$.

Theorem 5. *If G is connected, the trees constructed by the MT-HW algorithm must be connected shortest-path trees.*

Proof. Since G is connected, for each T_{σ_i} , a sensor x ($x \neq \sigma_i$) can always find one candidate parent as its parent in T_{σ_i} . Thus, T_{σ_i} will be a connected tree. Now, we further show that T_{σ_i} will be a shortest-path tree. By the definition of the candidate parent, the parent must be closer to σ_i than the node itself. Therefore, all T_{σ_i} are shortest-path trees. \square

4.3.2 The MT-EO Algorithm

From Eq. 4.4 and Eq. 4.5, we observe that if we can increase the number of the tree edges that overlap with each other, the value of $SC(v)$ may increase and U can be reduced. The MT-EO (multi-tree construction with the edge-overlap-first property) algorithm is designed to increase the level of the overlap among tree edges.

As the MT-HW algorithm, each sensor x will determine all candidate parents for each sink σ_i . Each of x 's neighbors is associated with an *overlap counter* for x . The counter is increased by one whenever a neighbor of x is considered as a candidate parent for a sink. Then, x selects the neighbor, say y , whose overlap counter is the largest. For each sink σ_i where y is a candidate parent of x , we set $y = p_i(x)$ for T_{σ_i} . Then, the overlap counters of all x 's neighbors are recomputed for those sinks for which x has not yet determined its parents. Again, the neighbor y whose overlap counter is the largest is selected as x 's parent for the corresponding sinks. This procedure is repeated until x has determined its parents for all sinks.

Theorem 6. *If G is connected, the trees constructed by the MT-EO algorithm must be connected shortest-path trees.*

Proof. The proof is similar to that of Theorem 5. The theorem holds because a non-sink node can always find a parent that is closer to the sink. \square

In fact, we can easily combine the MT-HW algorithm with the MT-EO algorithm and vice versa. Whenever there is a tie (either the same event rate or the same overlap counter value), the other algorithm can be used.

4.4 Simulation Results

We have simulated a sensing field of size 256×256 , where 1024 sensors are deployed in the sensing field. Two deployment models are considered. In the

regular deployment model, sensors are regularly deployed as a 32×32 grid-like network. In the random deployment model, sensors are randomly deployed. In both models, sinks are determined by uniformly partitioning the sensing field into equal-size grids according to the number of sinks given and choosing the sensor that is the nearest to the center of the grid as the sink. Further, the event rates of links are generated based on the *modified city mobility model* presented in Sec. 3.3. Queries could be issued from any sensor. The query rate is defined as the number of queries generated in the network per unit time. We compare our schemes with other two schemes called *QF* and *MC* respectively. In the QF scheme, no update message will be sent. When a user intends to query an object's location, the query message will be flooded in the network. In the MC scheme, when an object moves to a new sensor, a multicast spanning tree will be formed from the new location of the object to all sinks and the update message containing the up-to-date location information of the object is sent to all sinks. In this scheme, any query only needs to be sent to its nearest sink. Based on the tree construction algorithms and the forwarding schemes, four schemes proposed by us are compared with the QF and the MC schemes. Specifically, in the HW-B scheme, the MT-HW algorithm and the broadcast forwarding scheme are used. In the HW-U scheme, the MT-HW algorithm and the unicast forwarding scheme are used. In the EO-B scheme, the MT-EO algorithm and the broadcast forwarding scheme are used. Finally, in the EO-U scheme, the MT-EO algorithm and the unicast forwarding scheme are used.

As mentioned above, when an object moves from one sensor to another, if no packet loss arises and the update procedure can be completed within a period during which the object does not move again, our proposed update mechanism can ensure that the *Detected_Lists* of sensors are fresh. However, packet loss is a common phenomenon in a wireless network and transmission delay should also be taken into consideration. In order to investigate the impact of packet loss, we develop an event-oriented simulator using C language in which the unslotted CSMA defined in IEEE 802.15.4 [12] is implemented. Because we observe that the collision phenomenon is very severe, we assume that a node has to wait $10 \sim$

Table 4.2: Parameters used in the simulation for multi-sink systems.

Buffer Size	10
The length of DATA	30 Bytes
The length of ACK	17 Bytes
Bit rate	250 kb/s
Symbol rate	62.5 ksymbol/s
aUnitBackoffPeriod	20 symbols
aTurnaroundTime	12 symbols
macMinBE	3
aMaxBE	5
macMaxCSMABackoffs	4
The maximum number of retransmission	5
Simulation Time	1 hour
Number of Objects	128

60 milliseconds to start a new transmission after it successfully transmits a packet in order to avoid multiple sensors transmit packets at the same time. Finally, we assume each sensor's sending buffer is limited such that for a sensor, if there are too many packets to be sent simultaneously, some of packets will be discarded. The related parameters are shown in Table 6.1.

4.4.1 Impact of Objects' Speeds

First, we consider the scenario in which the update cost dominates the overall communication cost. To achieve this, we compare all schemes under various objects' speeds. Higher the speed is, more events are generated; thus, the update cost will dominate the performance. In Fig. 4.4, sensors are deployed regularly and four sinks are deployed. The query rate is set to be 1 query/second in this experiment. Fig. 4.4(a) shows the communication cost (i.e., the number of packets transmitted in the network) of these schemes with the value of object speed varied. As can be seen in Fig. 4.4(a), the update cost is constant in the QF scheme because no update packet has to be sent. The update costs of all other schemes will grow when the speed becomes higher since more update packets have to be sent. The update cost of the MC scheme grows enormously, because no in-network process-

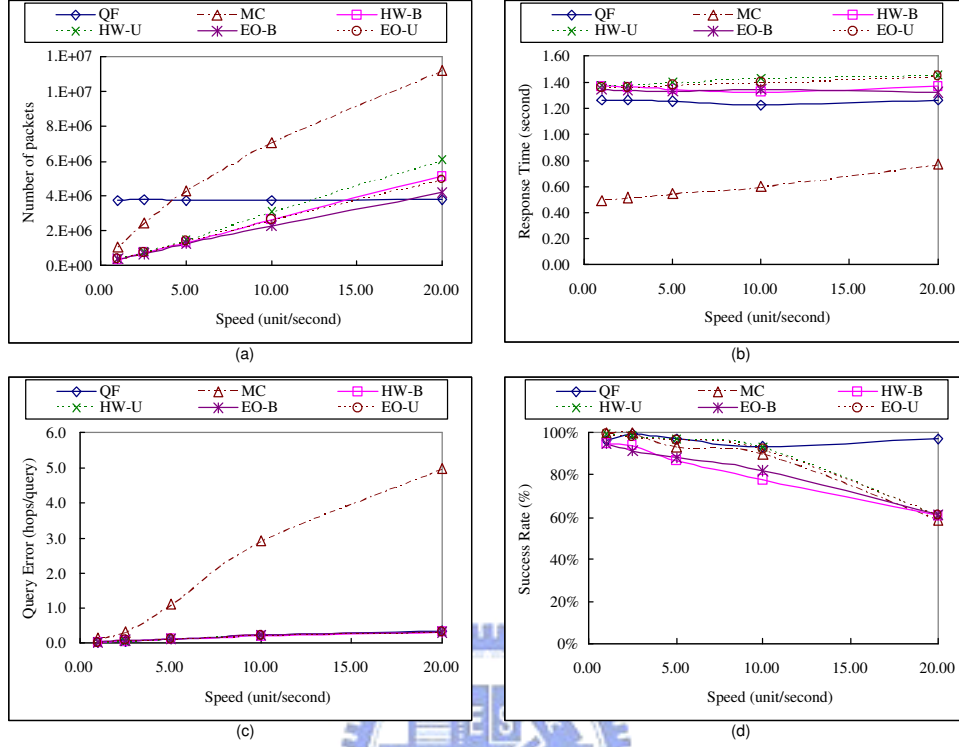


Figure 4.4: Performance study with objects' speeds varied, where sensors are deployed regularly and four sinks are deployed.

ing technique is applied. Our proposed schemes outperform the QF scheme and the MC scheme when the speed is lower than 10 units/second. Since the sensing radius of a sensor is 4 units, 10 units/second is relatively high. We further give an insight into our proposed scheme. Obviously, the broadcast forwarding scheme has lower update cost than the unicast scheme has. However, as can be seen later, the unicast scheme has higher query success rate than the broadcast scheme has. Besides, we can see that the MT-EO scheme outperforms the MT-HW scheme slightly, because more packets are saved due to the overlap of tree edges.

Fig. 4.4(b) shows the query response time of these schemes, where the query response time is defined as the time elapsed between the time at which the query issued and the time at which the query result returned. The MC scheme is the best

because any query only has to be forwarded to the sink. Our proposed schemes are slightly worse than the QF scheme because two phases are required in our schemes. Although the MC scheme has the best performance in terms of query response time, the query result may not be the most up-to-date one. This problem becomes further severe when packet loss happens. A measurement, query error, is defined as the number of hops between the real location of the object and the location carried by the query reply at the time at which query is returned to the user. In Fig. 4.4(c), it can be seen that the MC scheme suffers from higher query errors. Finally, Fig. 4.4(d) shows the query success rates under different schemes. Note that a query may fail due to packet collision, packet loss, buffer overflow and contaminated *Detected Lists*. More packets transmitted in the network usually means more collision. Thus, our proposed scheme and the MC scheme perform worse than the QF scheme does eventually, but all schemes have similar performance under reasonable speed. Note that the broadcast forwarding scheme has the worst performance due to the contaminated *Detected List* problem; however, the unicast forwarding scheme can be used to solve this problem.

Since the number of sinks is an important issue in this chapter, the scenario used in Fig. 4.4 is applied again in Fig. 4.5 except that 256 sinks are deployed now. It is observed that if the number of sinks is large, a considerable amount of update messages will be generated. Thus, when the update cost dominates the communication cost, using less sinks is better. Finally, experiments with the random deployment model is investigated in Fig. 4.6, where the number of sinks is 4. We can see that the success rates under the random deployment model are lower than that under the regular deployment model, because the collision phenomenon is very severe in the random deployment model. When a node has many neighbors, this node usually suffers severe collision due to the contention and the hidden terminal problem. Therefore, we further compute the average number of neighbors of a sensor. The average numbers of neighbors of a sensors under the regular deployment model and the random deployment model are 3.875 and 5.666 respectively. Thus, we conjecture that the severe collision phenomenon in the ran-

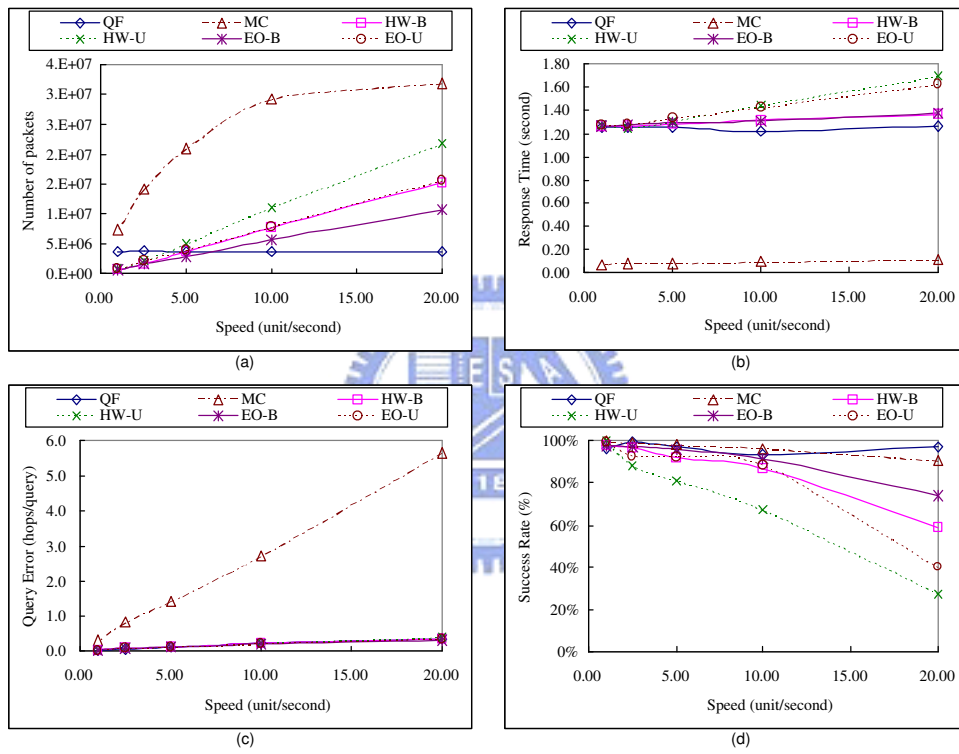


Figure 4.5: Performance study with objects' speeds varied, where sensors are deployed regularly and 256 sinks are deployed.

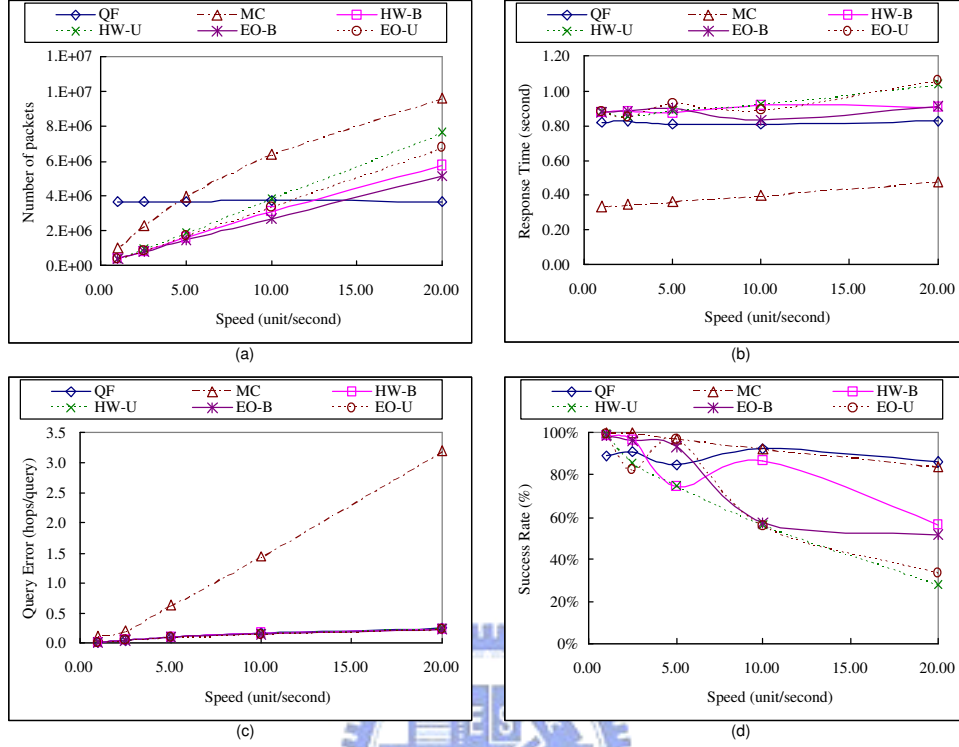


Figure 4.6: Performance study with objects' speeds varied, where sensors are deployed randomly and four sinks are deployed.

dom deployment model is caused by the hidden terminal problem and the higher contention between sensors. We further give an insight into our proposed scheme. We can find that the performance of the unicast forwarding scheme is very bad due to the buffer overflow problem. The reason can be explained as follows: when an event occurs, there are averagely 5.666 update packets will be injected into the sending buffer and the length of sending buffer is 10 only. Thus, the length of the sending buffer should be designed carefully. Other most observations made under the regular deployment model could be applied to the random deployment model. In the following experiments, we only show the results under the regular deployment model.

4.4.2 Impact of Query Rates

Now we consider the scenario in which the query cost dominates the overall communication cost. To achieve this, we compare all schemes by adjusting query rates. When the query rate is high, the query cost will dominate the performance. The object's speed is set to be 1 unit/second in this experiment. 4 and 256 sinks are deployed in Fig. 4.7 and Fig. 4.8 respectively. First, we compare the communication costs under different schemes. As shown in Fig. 4.7(a) and Fig. 4.8(a), the QF scheme is the worst one, because queries are disseminated by flooding. On the contrary, in our proposed schemes, queries are disseminated by unicasting. Thus, our proposed schemes have the best performance. We can further observe that when the number of sinks increases from 4 to 256, the communication cost of the MC scheme also grows due to higher update costs. However, our proposed schemes can achieve almost the same cost when the number of sinks increases. This is because using multiple sinks can reduce the query cost by a shorter query path and the saved query cost can be used to compensate the increased update cost. Thus, the advantage of using multiple sinks can be achieved when the query cost dominates the performance. In addition, when the number of sinks increases (i.e., from 4 in Fig. 4.7 to 256 in Fig. 4.8), it can be seen that the query response time of our proposed schemes in Fig. 4.8(b) is slightly smaller than that in Fig. 4.7(b) due to shorter query paths. As shown in Fig. 4.7(c) and Fig. 4.8(c), although the MC scheme is the best one in terms of query response time, it is the worst one in terms of query error. Finally, in Fig. 4.7(d) and Fig. 4.8(d), we can see that the QF scheme is the worst one in terms of success rate, because of the collision incurred by the flooding.

4.4.3 Impact of the Number of Sinks

From the previous experimental results, it can be seen that when the query cost dominates the communication cost, using multiple sinks can achieve better performance. Thus, we further investigate the impact of the number of sinks on the

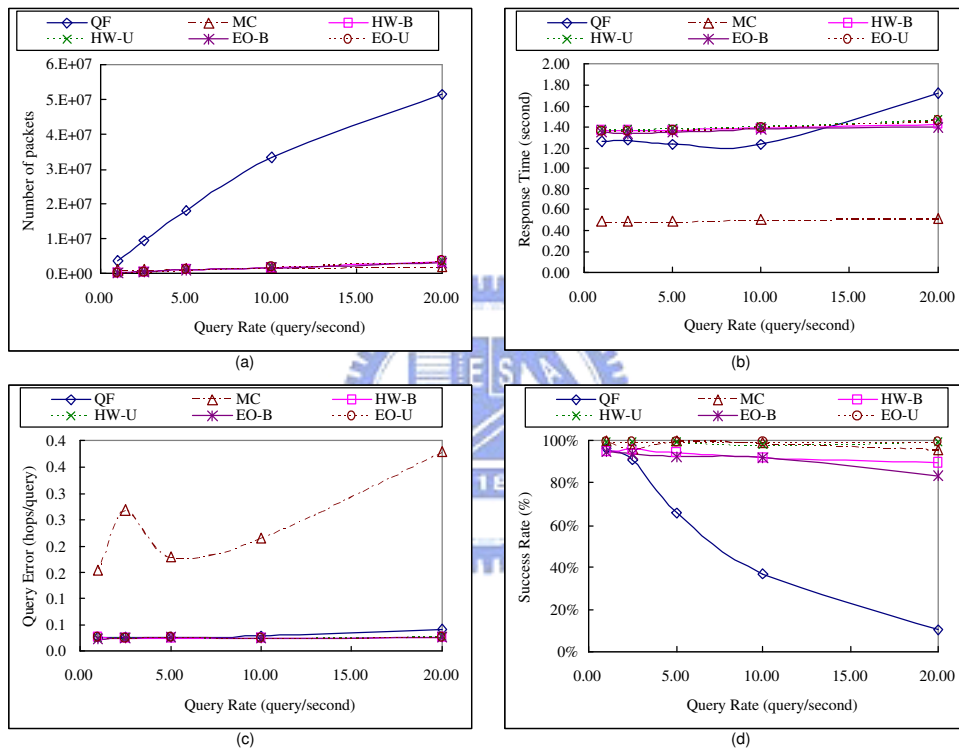


Figure 4.7: Performance study with query rates varied, where sensors are deployed regularly and four sinks are deployed.

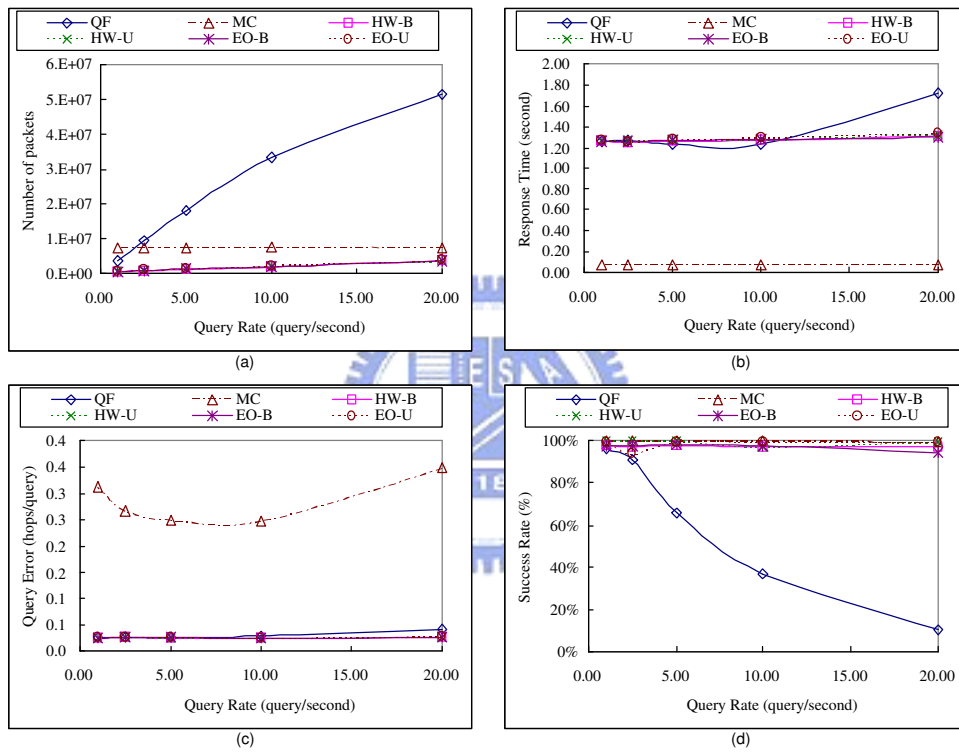


Figure 4.8: Performance study with query rates varied, where sensors are deployed regularly and 256 sinks are deployed.

performance. The query rate is set to be 10 queries/second and the objects' speed is set to be 0.333 and 0.111 respectively. In Fig. 4.9(a) and Fig. 4.10(a), it can be seen that the communication costs almost do not increase when the number of sinks increases, because the increased update cost can be compensated by lower query cost. As can be seen in Fig. 4.9(b) and Fig. 4.10(b), using multiple sinks can reduce the query response time slightly due to shorter query paths. Fig. 4.9(c) and Fig. 4.10(c) show the values of the standard deviation of the number of packets transmitted by each sensor. It is observed that when the number of sinks increases, the values of the standard deviation are reduced. This is because queries are dispersed to multiple sinks rather than a single sink. Thus, load balance can be achieved by using multiple sinks. Finally, in Fig. 4.9(d) and Fig. 4.10(d), it can be seen that using multiple sinks is able to increase the success rate, because shorter query paths could result in less collision.

4.4.4 Multi-Sink Systems with Partial Storage

As mentioned above, using multiple trees will increase the update cost. A simple way to reduce the update cost while achieving the advantage of load balance at the same time is to explore the *partial storage technique*. The partial storage technique is motivated by GHT [23]. The basic idea is that each object's location will be stored in only some of the sinks. In our simulation, the partial storage technique is implemented as follows.

First, we evenly divide the sensing field into m zones, each of which has a unique ID. For each zone, the sensor closest to the center of the zone is designated as the sink. Then, each object is hashed into l zones, where $l (< m)$ is a predefined number, and an object only needs to update its location with the sinks of these l zones.

Now, we demonstrate the benefit of the partial storage technique by simulation. The query rate is set to be 2 queries/second and the objects' speed is set to be 1 unit/second. We compare the EO-B and the EO-U schemes with α sinks

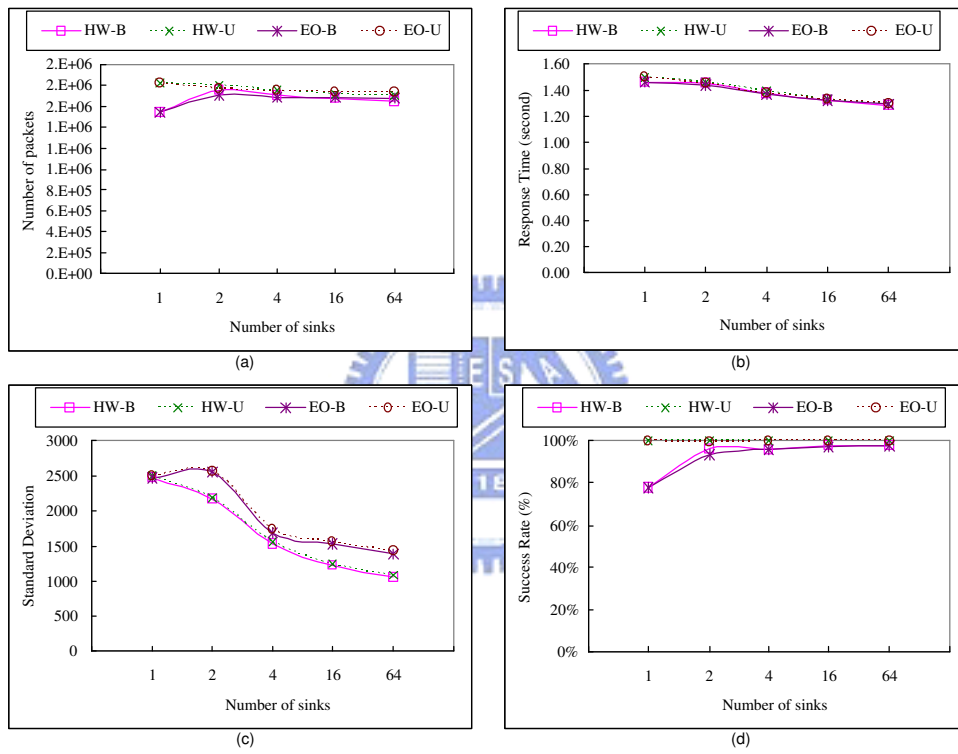


Figure 4.9: Performance study with the number of sinks varied, where the objects' speed is set to be 0.333 unit/second.

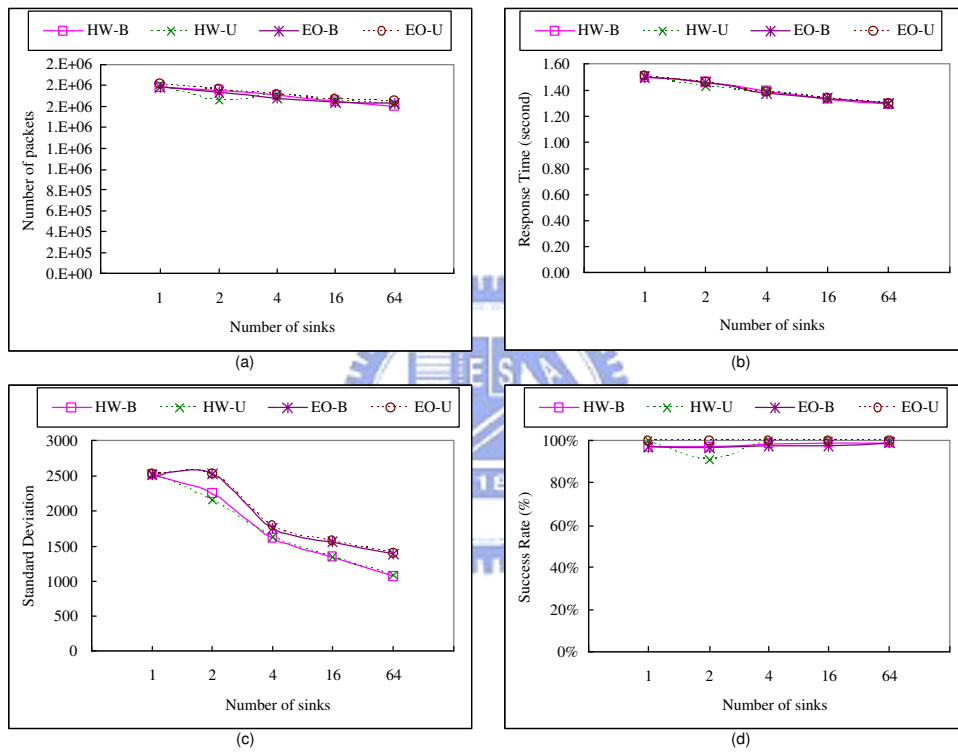


Figure 4.10: Performance study with the number of sinks varied, where the objects' speed is set to be 0.111 unit/second.

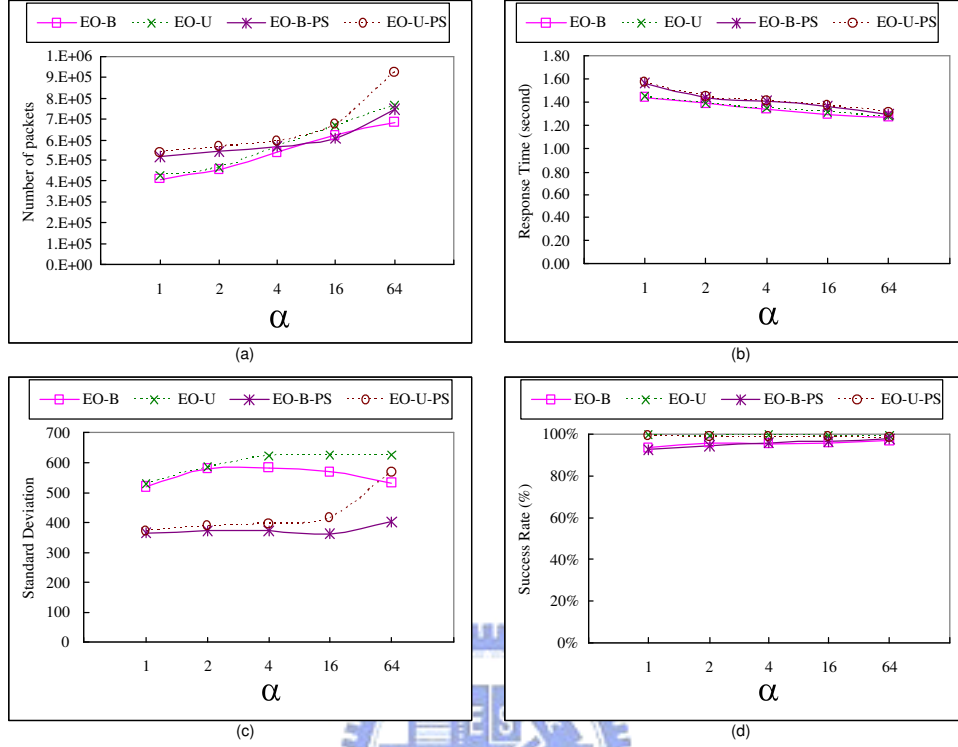


Figure 4.11: The performance of the partial storage technique.

against the EO-B-PS and the EO-U-PS schemes (which mean the EO-B and the EO-U schemes extended with the partial storage technique) with 1024 zones and α hashed zones per object. Fig. 4.11 shows the results with the value of α varied. It can be observed that, although the communication costs of the EO-B-PS scheme and the EO-U-PS scheme are higher, the values of the standard deviation of the numbers of packets transmitted by each sensor are lower. Thus, using the partial storage technique can achieve better load balance.

4.5 Summary

In this chapter, we have proposed an in-network update and query algorithm for a multi-sink WSN. This algorithm strikes the tradeoff between the update and query costs. Having multiple sinks is important when the network scale is large

or when the query rate is high. The corresponding update cost is formulated formally. Based on the formulation, we have presented two distributed algorithms to construct multiple trees. We have verified the benefits of a multi-sink WSN from different aspects, including the total (update plus query) cost, the number of sinks, query response time, query success rate, and load balance factor.



Chapter 5

Imprecision-tolerant Location Management Model

Since inaccuracy, or even error, of sensing data is inherent for WSNs, applications of WSNs usually have to tolerate some degree of imprecision. This property has been exploited in the design of network protocols for WSNs. For example, precision-constrained data aggregation is considered in [28], and a storage system that supports drill-down queries with different precision levels is proposed in [11]. Similarly, in moving object environments, maintaining the exact locations of objects anytime is almost infeasible [8, 33]. Not only the positioning results are error-prone, but also the data transfer delay and object mobility make the locations of objects inaccurate. Fortunately, imprecision is tolerable in many object tracking applications. For example, when life scientists intend to track an animal, it may be sufficient to know its moving direction rather than its exact location. In addition, the location information recorded several hours ago, instead of at the current time, may still be helpful for the life scientists to understand the animal's daily life. Therefore, modeling in-network location management to support imprecision-tolerant queries is desirable for object tracking sensor networks.

A location management scheme supporting imprecision-tolerant queries for object tracking sensor networks has been studied in [33]. The location information of an object is stored in a centric storage node and a local storage node. When a user intends to know the location of an object, the query will be forwarded from

the querying node to the centric storage node of that object. If the precision level is satisfactory, the centric storage node will reply to this query. Otherwise, the query will be forwarded to the local storage node, which has more precise location information of that object. This scheme has two major drawbacks. First, when the querying node is very close to the local storage node of the queried object, the query will still be forwarded to the centric storage node, which may be far from the querying node. Second, only two precision levels are provided.

5.1 Preliminaries

5.1.1 Background and Motivations

In this chapter, we propose an in-network location management scheme to support imprecision-tolerant queries for object tracking sensor networks. Two types of imprecision are considered. *Spatial imprecision* means that an object could be located *near* the location answered by the WSN rather than *at* the location answered by the WSN. *Temporal imprecision* means that the location answered by the WSN may be recorded *near* the current time rather than *at* the current time. For both spatial imprecision and temporal imprecision, we argue that an imprecision-tolerant location management solution should achieve two desirable goals. First, multiple precision levels should be provided. Second, the query cost should be proportional to the precision level. For example, for spatial imprecision, the answer provided by node C should be more accurate than that provided by node A, because node C is farther from the sink (Fig. 5.1(a)). Similarly, for temporal imprecision, the location answered by node C should be newer than that answered by node A (Fig. 5.1(b)).

We observe that the tree-based location management schemes proposed in Chapter 3 could achieve these two goals naturally. For example, Fig. 5.2(a) shows a tree used for location management. In the tree-based location management scheme, when an object moves from one sensor to another, the update message will be forwarded to the lowest common ancestor of those two sensors. Thus,

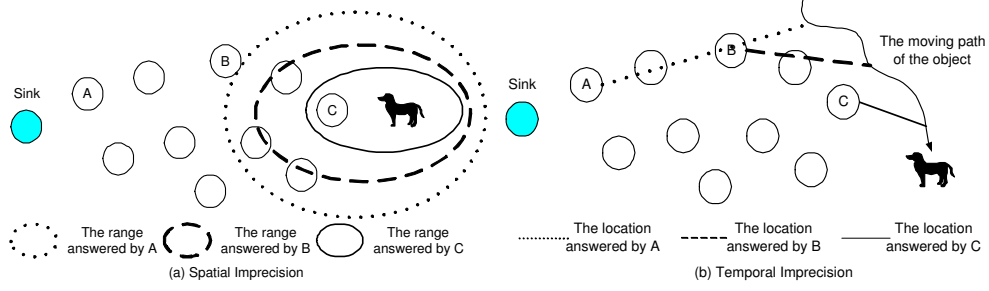


Figure 5.1: Examples of spatial imprecision and temporal imprecision.

when an object originally located outside the spatial range of the subtree rooted at y moves into the range of A at time t_0 , node x (the parent of y) will be updated. Thus, x knows that the object is located at A at time t_0 . When a user receives such an answer provided by x , the user can only drive that the object is located at some sensor belonging to a descendant of y . On the contrary in Fig. 5.2(b), if the query is forwarded to y , y can provide that the object is located at B at time t_1 , and the user can derive that the object is located at some sensor belonging to a descendant of z . Therefore, we can see that a user can get more precise location information when the query is forwarded more deeply down the tree. Further, if the tree is a *deviation-avoidance* tree defined in Chapter 3, it ensure that the hop count between y and the sink will be less than the hop count between z and the sink. It implies that the query cost will be proportional to the precision level. (Note that when a query is not issued from the sink, it is possible that the querying node is close to the object rather than the sink, and it needs to forward the query to the sink first. This may violate this goal. In this case, the multi-sink system proposed in Chapter 4 can be used to solve this problem, because each query is sent to the nearest sink.) In addition, because of its hierarchical structure, a tree-based solution can provide multiple precision levels easily.

Therefore, we propose a tree-based location management model to support imprecision-tolerant queries. To begin with, we define the format of imprecision-tolerant queries and describe how such queries are processed. The proposed query model can be applied to any tree structure. We then make some observations

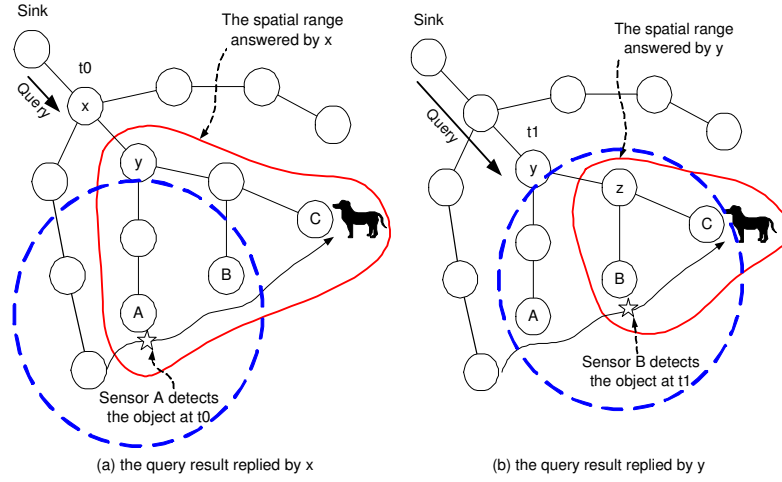


Figure 5.2: A tree-based location management scheme.

regarding the relationship between query cost and tree structure, and propose a tree construction algorithm to facilitate the proposed imprecision-tolerant location management model by reducing the query cost while minimizing the increment of the update cost. Finally, performance studies are conducted via simulations.

5.1.2 Network Model

The network model used in this chapter is the same with that proposed in Chapter 3. We consider a WSN to be used for object tracking. We adopt a simple *nearest-sensor tracking* model, in which the sensor that receives the strongest signal from an object is responsible for tracking the object (this can be achieved by [7] and we omit the details). Therefore, the sensing field can be modelled by a *Voronoi graph* [4], where each sensor's responsible area is the polygon containing itself. Two sensors are called *neighbors* if their sensing ranges share a common boundary on the Voronoi graph. Multiple objects may be tracked concurrently by the network, and we assume that from mobility statistics, it is possible to collect the frequency that objects move between each pair of neighboring sensors, called the *event rate*.

5.2 Imprecision-tolerant Location Management Model

We will propose a tree-based, imprecision-tolerant location management model in this section. Below, we will first introduce the update and query mechanisms. Since the update and query mechanisms can be applied to any tree structure, we will then discuss how to reduce the communication cost (i.e., update cost plus query cost) by adjusting the tree structure. We observe that *uncorrelated sensors* should not be put together under a subtree to reduce the query cost while *correlated sensors* should be put together to minimize the increment of update cost, where the formal definition of correlation of sensors will be introduced later. We will discuss how to collect query statistics to identify the correlation of sensors. Finally, based on query statistics, we propose a tree construction algorithm *IQT* (*Imprecision-tolerant Query Tree*) to reduce the query cost while minimize the increment of the update cost.

5.2.1 Imprecision-tolerant Update and Query Mechanisms

In this subsection, we will present the update and query mechanisms used in the imprecision-tolerant model. We assume a tree T rooted at the sink has been constructed. Each sensor x will maintain an object list OL_x that stores the object information known by x . For each object o in OL_x , three data are recorded:

- *o.next*: This information is used for forwarding the query to find the object. If o is tracked by x , then *o.next* is x itself. Otherwise, *o.next* will be a child of x , and o is tracked by a sensor that is a descendant of *o.next*.
- *o.location*: This information is stored the last location information of o known by x .
- *o.time*: This information is stored the newest update time of the information of o .

Now we describe the update mechanism. The main idea is forwarding update packets to the lowest common ancestor. When a sensor x receives an update

packet $Update(o, a, b, t)$ (i.e, object o moves from a to b at time t), x will take the following actions (For simplicity, we say a node is also a descendants of itself.):

- If b is not a descendant of x , then x will remove o from OL_x , because the queries of o will not be forwarded to x anymore. Then, x will further forward the $Update(o, a, b, t)$ to its parent.
- If b is a descendant of x but a is not a descendant of x , then x will add o 's information into OL_x . If $x = b$, then $o.next$ will be set to b . Otherwise, $o.next$ is set to the child of x that sends $Update(o, a, b, t)$ to x . In addition, $o.location$ and $o.time$ are to b and t respectively. Then, x will further forward the $Update(o, a, b, t)$ to its parent.
- If both a and b are descendants of x (i.e., x is the lowest common ancestor of a and b), then x will modify o 's information in OL_x . If $x = b$, then $o.next$ will be set to b . Otherwise, $o.next$ is set to the child of x that sends $Update(o, a, b, t)$ to x . In addition, $o.location$ and $o.time$ are to b and t respectively.

Next, we define the query format and the query mechanism. The query used in this work can be represented as $Query(o, tolerant_radius, tolerant_interval)$, where $tolerant_radius$ is used for supporting spatial imprecision, and $tolerant_interval$ is used for supporting temporal imprecision. Before describing the query mechanism, we first define $CIRCLE(x, r)$ as the circle area that is centered at sensor x and is with radius r .

The imprecision-tolerant query mechanism operates as follows. When a sensor x (including the sink) receives a query $Query(o, tolerant_radius, tolerant_interval)$, x will check OL_x and take the following actions:

- If the queried object is tracked by x currently (i.e., $o.next = x$), then x will reply the query immediately.

- If the queried object is not tracked by x currently, then x will check whether both of the following conditions are true or false. We assume that $o.next = y$, where $y \neq x$.

1. $\forall z \in Subtree(y), z$ is inside $CIRCLE(o.location, tolerant_radius)$,
and
2. $Current_Time \geq o.time \geq Current_Time - tolerant_interval$,

where $Subtree(y)$ is the set of sensors that are members of the subtree rooted at y (note that $y \in Subtree(y)$), and $Current_Time$ denotes current time. Based on the check, x will act as follows.

- If both of these two conditions are true, x will reply the query so that the user will know that o is located at $o.location$ at time $o.time$.
- Otherwise, the query will be further forwarded to y until the object is found or both of the above conditions are true.

We further explain these two conditions. The first condition is for spatial imprecision. Theorem 7 shows that if the first condition is true, the distance between $o.location$ and the real location of o will be less than or equal to $tolerant_radius$. Thus, $o.location$ is the acceptable answer for this query. The second condition is for temporal condition. Intuitively, $o.time$ is acceptable only when its value is larger than or equal to $Current_Time - tolerant_interval$. An example is shown in Fig. 5.2, where we assume a query $Query(Dog, tolerant_radius, tolerant_interval)$ is issued. (Note that the dotted circles shown in Fig. 5.2(a) and Fig. 5.2(b) are $CIRCLE(A, tolerant_radius)$ and $CIRCLE(B, tolerant_radius)$, respectively.) In the case of Fig. 5.2(a), we can see that x cannot reply this query even $Current_Time > t_0 > Current_Time - tolerant_interval$, because C is one of y 's descendants and C is not located in $CIRCLE(A, tolerant_radius)$. On the contrary, in the case of Fig. 5.2(b), y can reply the query if $Current_Time > t_1 > Current_Time - tolerant_interval$, because z and all of its descendants are located in $CIRCLE(B, tolerant_radius)$.

Theorem 7. For an object o known by sensor x , assume that $o.next = y$. If $\forall z \in Subtree(y), z$ is inside $CIRCLE(o.location, tolerant_radius)$, then the distance between $o.location$ and the real location of o will be less than or equal to $tolerant_radius$.

5.2.2 Tree Optimization

Query Statistics

Later, we will show that *uncorrelated sensors* should not be put together under a subtree to reduce the query cost while *correlated sensors* should be put together to minimize the increment of update cost. Thus, we first present how to collect query statistics to identify the correlation of sensors.

The statistics is done by the sink. The sink will maintain a counter $be_queried$ for each sensor x . After a query $Query(o, tolerant_radius, tolerant_interval)$ returns to the sink with the result indicating that the object is located at sensor x , the sink will increase $x.be_queried$ by 1. In addition, the *correlated sensors* of x will also be recorded. Based on the query format defined in Sec. 5.2.1, we define the correlated sensors are those located in $CIRCLE(x, \min(tolerant_radius, tolerant_interval \times avg_speed))$, where avg_speed denotes the average speed of the queried object.

The query statistics will be used for the tree construction algorithm. Thus, a question is how to do the query statistics before the tree is constructed. We propose two approaches. The first approach is called *ideal-collection*. In this approach, if a query is issued at time t and the queried object is located at sensor x at time t , then $x.be_queried$ will be increased by 1. This approach can collect the most precise data, but this approach is unrealistic, because the exact location of the object at time t is hard to get precisely.

The second approach is called *tree-collection*, in which a *collection tree* (e.g., the DAT tree presented in Chapter 3) will be used initially to do query statistics. Then, by this query statistics, the tree optimized by considering the imprecision-

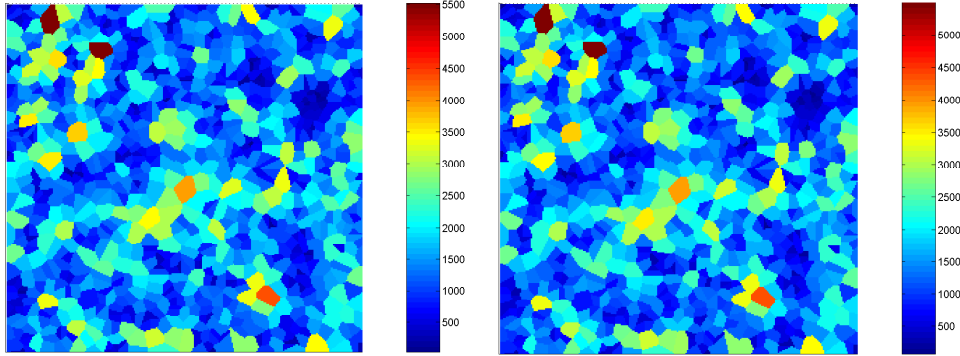


Figure 5.3: The values of *be_queried* of sensors collected by (a) ideal-collection and (b) tree-collection, where the DAT tree is used.

tolerant queries can be constructed. However, because imprecision is unavoidable in the tree-collection scheme, the result of statistics may be different from that collected by ideal-collection. Fig. 5.3 shows the values of *be_queried* of sensors collected by ideal-collection and tree-collection respectively, where 1024 sensors are randomly deployed in a 256×256 field with uniform distribution and each sensor is represented as a Voronoi cell. We can see that the results are similar. Even the result collected by the IQT (Imprecision-tolerant Query Tree) tree that will be described in the next subsection is also similar to those collected by ideal-collection and the DAT tree. The reason is that the query is just replied early in the tree-collection approach and the error is acceptable even although the replied result may be imprecise. Thus, we can construct a tree initially to do query statistics, and then construct a new tree by considering imprecision-tolerant queries to reduce the communication cost further.

The only one remaining question is the rule used for determining the correlation between two nodes. Based on the query statistics, we define a function $Confidence(a, b)$ as $N(a, b)/a.be_queried$, where $N(a, b)$ is the number of queries in which a is queried and b is one of correlated sensors. If $Confidence(a, b) > min_confidence$, we say that b is a correlated sensor of a .

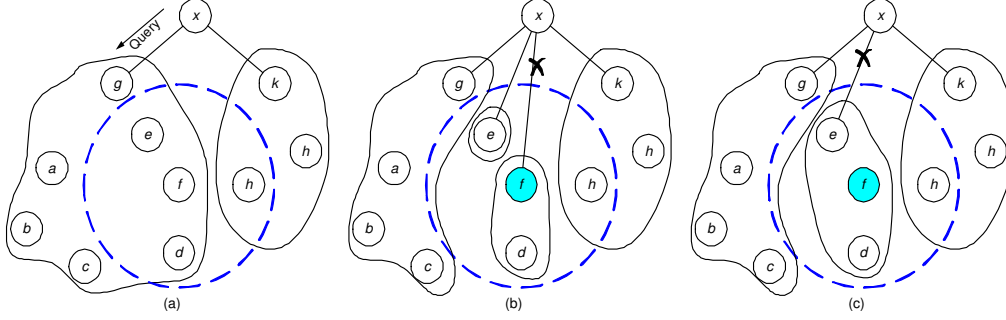


Figure 5.4: Some observations.

Imprecision Query Tree Construction Algorithm

We argue that uncorrelated sensors should not be put together under a subtree to reduce the query cost while correlated sensors should be put together to minimize the increment of update cost. We use examples to explain the reasons.

First, we focus on the spatial-imprecision. In Fig. 5.4(a), where the dotted circle denotes $CIRCLE(f, tolerant_radius)$, when x receives a query for object o and $o.location$ is f , it needs to forward the query to g , because not all members of $Subtree(g)$ are located in $CIRCLE(f, tolerant_radius)$. On the other hand, in Fig. 5.4(b), we can see that when uncorrelated sensors (i.e., a, b, c and g) are removed from the subtree, x can reply this query now. Thus, the query cost is reduced. However, in Fig. 5.4(b), we can see that when an object moves from f to e , update packets have to be sent to x . Thus, when we put correlated sensors together, the increment of update cost can be minimized.

For the temporal-imprecision, similar results can be derived. Now, we assume the dotted circle shown in Fig. 5.4 is $CIRCLE(f, tolerant_interval \times avg_speed)$, where avg_speed denotes the average speed of the queried object. This means that after $tolerant_interval$, the object may be located at the outside of the circle. However, if the object is still located in the subtree of g , $o.time$ will not be updated. On the contrary, when we remove the uncorrelated sensor of f (e.g., Fig. 5.4(b) and Fig. 5.4(c)), $o.time$ will be updated so that x could reply this query.

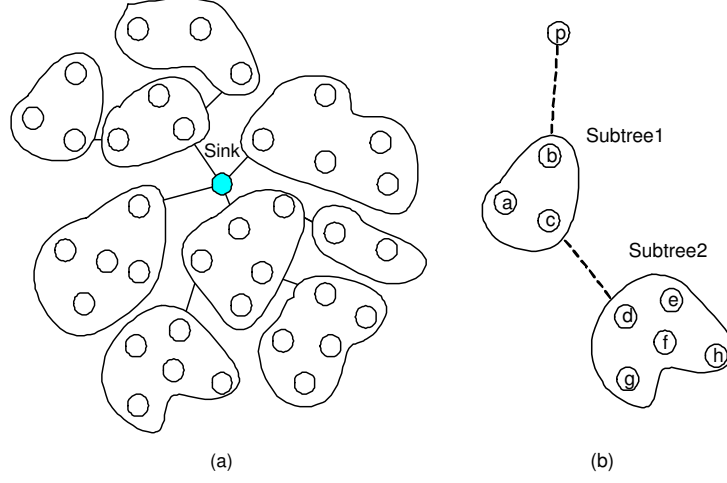


Figure 5.5: (a) The basic idea of constructing a tree. (b) The problem arising when connecting two subtrees.

Based on the observations mentioned above, we develop a tree construction algorithm IQT. The basic idea is dividing sensors into subtrees formed by correlated sensors and then connecting those subtrees into a tree as shown Fig. 5.5(a). However, in Fig. 5.5(b), we can note that when *Subtree2* connects to *Subtree1*, the composed members of *Subtree1* are changed. Now when sensor p receives a query for object o and $o.location$ is a , the probability that p cannot reply this query is high, because some uncorrelated sensors (i.e., those sensors in *Subtree2*) are attached to *Subtree1*.

One way to solve this problem is connecting all subtrees to the sink. Obviously, this solution is not scalable, because when an object moves from one subtree to another subtree, update messages have to be sent to the sink. Thus, we propose a backbone-based solution to solve this problem. First, some sensors will be selected to be backbone nodes. These backbone nodes will form a backbone tree. On the other hand, other non-backbone nodes will form subtrees according to the query statistics. Finally, all subtrees are connected to the backbone to form a single tree. The pseudo-code of the IQT algorithm is shown in Algorithm 3, where G denotes the modeled network graph and QS denotes the query statistics. The

details of each procedure will be further described in the following paragraphs.

Algorithm 3 $\text{IQT}(G, QS)$

- 1: *Backbone-Construction*(G, QS)
 - 2: *Subtrees-Formation*(G, QS)
 - 3: *Connecting-Subtrees-To-Backbone*(G)
-

Backbone-Construction(). The first step of *Backbone-Construction()* is selecting backbone nodes. Recall that the major task of backbone nodes is to reply the queries early so that the queries do not need to be forwarded to the subtrees. Thus, two principles should be followed when selecting backbone nodes:

- *The backbone node should be close to the sink.* We can see that when the backbone node is close to the roots of the subtrees, the saved query cost is limited. On the contrary, when the backbone node is close to the sink, the backbone node can reply the queries more early.
- *The value of $be_queried$ of a backbone node should be low.* The IQT algorithm will attach subtrees to the backbone nodes later and most nodes of the subtrees are not the correlated node of backbone nodes. Thus, when the result of a query is a backbone node (recall that the value of $be_queried$ of the backbone node will be added by 1 in this case), this query usually will be forwarded to that backbone node (i.e., no query cost can be saved). Therefore, this is the reason that the sensor that is queried rarely should be selected.

Procedure 4 shows the pseudo-code of the *Backbone-Construction* procedure and the backbone nodes selection procedure is from line 1 to line 13, where two parameters are used. The first parameter α ($0 \leq \alpha \leq 1$) is used to limit the values of $be_queried$ of backbone sensors. We can see that the sensors that are queried rarely will be selected. (Note that we can also see that at most $\lfloor \alpha \times |V_G| \rfloor$ backbone nodes will be selected.) On the other hand, the second parameter β ($0 \leq \beta \leq 1$) is used to limit the distance between the backbone node and the

sink so that the backbone nodes can be the nodes close to the sink. (Note that $MAX_HOPCOUNT = \max\{hop_count(x, sink) | \forall x \in V_G\}$)

After selecting the backbone nodes, these backbone nodes form a *backbone subgraph* (denoted by $BG = (V_{BG}, E_{BG})$), in which V_{BG} is the set of backbone nodes and an edge belongs to E_{BG} if both of its two incident nodes are backbone nodes. Then, the DAT algorithm proposed in Chapter 3 is run on BG . The reason to do so is to connect backbone nodes and minimize the update cost locally. Because BG may be disconnected, it is possible that some backbone nodes do not have parents after running the DAT algorithm. Obviously, a backbone node should choose a parent that is also a backbone node. Thus, a backbone node x that does not have a parent will choose a backbone node y as its parent so that $hop_count(sink, x) = hop_count(sink, y) + hop_count(x, y)$. This ensures that the tree will be deviation-free. If there are more than one such backbone node, then the sensor y that makes $hop_count(x, y)$ minimum will be selected. The reason why minimum is chosen is to reduce the update cost. Procedure 4 shows the related pseudo-code (from line 14 to line 24).

Subtrees-Formation(). For non-backbone nodes, they should form subtrees based on the query correlation as we mentioned above. To begin with, we sort non-backbone nodes by $be_queried$ in increasing order. The reason why increasing order is used is explained as follows: the best tree for reducing query cost is the one where each non-backbone node connects to a backbone node alone. In this case, we can see that for a non-backbone node, no other uncorrelated sensors will exist. However, this will incur higher update cost. So we prefer to form correlated sensors into subtrees to minimize the increment of update cost. Thus, when we examine the non-backbone nodes in increasing order, the sensors with lower $be_queried$ will form subtrees first and the sensors with higher $be_queried$ have higher opportunity to be alone. (This ensures that the sensors with higher $be_queried$ have no uncorrelated sensors for each query.)

Then, we examine each node in L . If a node $x \in L$ is not examined yet, then it will form a subgraph $ST = (V_{ST}, E_{ST})$ first. Thus, the major task of

Procedure 4 Backbone-Construction(G, QS)

```
1:  $V_{BG} \leftarrow \{sink\}$ 
2:  $y \leftarrow$  the sensor whose  $be\_queried$  is the  $\lfloor \alpha \times |V_G| \rfloor$ -th least one among all
   sensors
3: for each node  $x \in V_G$  except for the sink do
4:   if ( $hop\_count(x, sink) < \beta \times MAX\_HOPCOUNT$ )  $\wedge$  ( $x.be\_queried < y.be\_queried$ ) then
5:      $V_{BG} \leftarrow V_{BG} \cup \{x\}$ 
6:   end if
7: end for
8:  $E_{BG} \leftarrow \phi$ 
9: for each  $e \in E_G$  do
10:  if both of  $e$ 's two incident nodes belong to  $V_{BG}$  then
11:     $E_{BG} \leftarrow E_{BG} \cup \{e\}$ 
12:  end if
13: end for
14:  $DAT(BG)$ 
15: for each node  $x \in V_{BG}$  except for the sink that does not determine its parent
    do
16:    $cp \leftarrow \phi$ 
17:   for each node  $y \in V_{BG}$  do
18:     if  $hop\_count(sink, x) = hop\_count(sink, y) + hop\_count(x, y)$  then
19:        $cp \leftarrow cp \cup \{y\}$ 
20:     end if
21:   end for
22:   choose a node  $p$  such that  $hop\_count(p, x) = \min\{hop\_count(y, x) | \forall y \in cp\}$ 
23:    $x$ 's parent  $\leftarrow p$ 
24: end for
```

the *Subtrees-Formation* procedure is to determine the composed members of V_{ST} (i.e., x 's correlated sensors). To begin with, we can see that for a subtree, the node that has the highest *be_queried* should be considered first (i.e., we should find its correlated sensors). Thus, we define the leader of a subtree (denoted by *st_leader*) as the node that has the highest *be_queried* among the sensors in V_{ST} . Initially, V_{ST} only contains x and x is the initial leader. Then, x 's neighbors that are non-backbone and not examined yet will be considered to be added into V_{ST} . The set of these nodes is called *candidate_list* denoted by cl . Now we need to decide whether a node y in cl is a correlated node of the subtree being examined. First, y will check whether $y.be_queried$ is larger than $st_leader.be_queried$. If so, then y may become the new leader and it should check whether all of the members of the subtree are its correlated nodes. If the answer is affirmative, y will be added into V_{ST} and cl also will be updated by considering y 's neighbors. On the other hand, if $y.be_queried$ is less than or equal to $st_leader.be_queried$, then the leader will check whether y is its correlated nodes. Again, if the answer is affirmative, y will be added into V_{ST} and cl also will be updated by considering y 's neighbors. The same procedure will be performed on all nodes in cl until cl becomes empty. So far, V_{ST} is determined. An edge belongs to E_{ST} if both of its two incident nodes are in V_{ST} ; thus, E_{ST} is also determined. Again, we run the DAT algorithm to construct a subtree from ST . The pseudo-code of the *Subtrees-Formation* procedure is shown in Procedure 5.

Connecting-Subtrees-To-Backbone(). Because subtrees are formed separately, some non-backbone nodes do not have parents yet after the *Subtrees-Formation* procedure. From the example shown in Fig. 5.5(b), we can know that a subtree formed by non-backbone nodes should connect to a backbone node. Recall that, in the *Backbone-Construction* procedure, the backbone nodes that do not have parents after running the DAT algorithm has a procedure to choose their parents (from line 15 to line 24). The similar approach can be used to connect subtrees to the backbone and the *Connecting-Subtrees-To-Backbone* procedure is shown in Procedure 6. We modify the original procedure slightly. Specially, we add a

Procedure 5 *Subtrees-Formation*(G, QS)

```
1: Sort non-backbone nodes into a list  $L$  by  $be\_queried$  in increasing order
2: for each node  $x$  in  $L$  do
3:    $x.examined \leftarrow 0$ 
4: end for
5: for each node  $x$  in  $L$  do
6:   if  $x.examined = 0$  then
7:      $V_{ST} \leftarrow \{x\}$ 
8:      $st\_leader \leftarrow x$ 
9:      $cl \leftarrow \{y | y \in L \wedge y \in Neighbor(x) \wedge y.examined = 0\}$ 
10:    while  $cl \neq \phi$  do
11:      Extract a sensor  $y$  from  $cl$ 
12:      if  $y.be\_queried > st\_leader.be\_queried$  then
13:        if  $\forall z \in ST, Confidence(y, z) > min\_confidence$  then
14:           $st\_leader \leftarrow y$ 
15:           $V_{ST} \leftarrow V_{ST} \cup \{y\}$ 
16:           $cl \leftarrow cl \cup \{z | z \in L \wedge z \in Neighbor(y) \wedge z.examined = 0\}$ 
17:           $y.examined \leftarrow 1$ 
18:        end if
19:        else if  $y.be\_queried \leq st\_leader.be\_queried$  then
20:          if  $Confidence(st\_leader, y) > min\_confidence$  then
21:             $V_{ST} \leftarrow V_{ST} \cup \{y\}$ 
22:             $cl \leftarrow cl \cup \{z | z \in L \wedge z \in Neighbor(y) \wedge z.examined = 0\}$ 
23:             $y.examined \leftarrow 1$ 
24:          end if
25:        end if
26:      end while
27:       $E_{ST} \leftarrow \phi$ 
28:      for each  $e \in E_G$  do
29:        if both of  $e$ 's two incident nodes belong to  $V_{ST}$  then
30:           $E_{ST} \leftarrow E_{ST} \cup \{e\}$ 
31:        end if
32:      end for
33:       $DAT(ST)$ 
34:    end if
35:  end for
```

distance constraint in line 4, where a parameter γ ($0 \leq \gamma \leq 1$) is used to limit the distance between the root of the subtree and the root's parent. We can see that when γ is small, the distance between the root and its parent will be large and larger distance results in low query cost (because the parent can reply the queries more early) but high update cost.

Procedure 6 *Connecting-Subtrees-To-Backbone(G)*

```

1: for each non-backbone node  $x$  that does not determine its parent do
2:    $cp \leftarrow \phi$ 
3:   for each node  $y \in V_{BG}$  do
4:     if ( $\text{hop\_count}(\text{sink}, x) = \text{hop\_count}(\text{sink}, y) + \text{hop\_count}(x, y)$ )  $\wedge$ 
        ( $\text{hop\_count}(\text{sink}, y) < \gamma \times \text{hop\_count}(\text{sink}, x)$ ) then
5:        $cp \leftarrow cp \cup \{y\}$ 
6:     end if
7:   end for
8:   choose a node  $p$  such that  $\text{hop\_count}(p, x) = \min\{\text{hop\_count}(y, x) | \forall y \in cp\}$ 
9:    $x$ 's parent  $\leftarrow p$ 
10: end for

```

Correctness. Finally, we show the correctness of the IQT algorithm. (A tree is deviation-free if for all $x \in V_G$ the hop count of the tree path from x to the sink is equal to the minimum hop count between x and the sink.)

Theorem 8. *If G is connected, the tree constructed by algorithm IQT is a connected deviation-avoidance tree rooted at the sink.*

5.3 Simulation Results

We have developed a simulator to demonstrate the efficiency of our proposed imprecision-tolerant location management model. A sensing field with size 256×256 units is simulated, in which 1024 sensors are deployed randomly with uniform distribution. The sensor located at one of corners of sensing filed is selected to be the sink.

Table 5.1: Parameters used in the simulation for imprecision-tolerant location management model.

Simulation Time	2592000 seconds
<i>MAX_TOLERANT_RADIUS</i>	30 units
<i>MAX_TOLERANT_INTERVAL</i>	3600 seconds
Number of objects	128

The event rates of links are generated based on the modified city mobility model presented in Chapter 3. Two query scenarios are simulated. In the first scenario, each object is queried evenly. In the second scenario, some objects will be queried frequently such that there are some query hotspots in the sensing field. Besides, for each query, the value of *tolerant_radius* is selected randomly from 0 to *MAX_TOLERANT_RADIUS* with uniform distribution, and the value of *tolerant_interval* is selected randomly from 0 to *MAX_TOLERANT_INTERVAL* with uniform distribution. The related parameters used in the simulation are shown in Table 6.1.

To begin with, we consider the scenario in which each object is queried evenly. In Fig. 5.6, we observe the impact of objects' speeds. (The settings of parameters $\langle \alpha, \beta, \gamma, min_confidence \rangle$ used in IQT1, IQT2, IQT3, IQT4 are $\langle 0.1, 0.3, 0.3, 0.9 \rangle$, $\langle 0.3, 0.3, 0.3, 0.9 \rangle$, $\langle 0.1, 0.5, 0.5, 0.9 \rangle$, and $\langle 0.1, 0.3, 0.3, 0.6 \rangle$.) Higher speed means higher update cost. To begin with, a DAT tree optimized by minimizing the update cost is constructed. We can see that when our proposed imprecision-tolerant query model is applied to the DAT tree (i.e., the DAT-I scheme in Fig. 5.6), the saved cost is limited, because most queries still need to be forwarded to the sensors that are tracking the queried objects. The proposed IQT tree optimized by reducing the query cost incurred by imprecision-tolerant queries can be used to solve this problem. Especially, when the query cost dominates the communication cost (i.e., when objects' speed is low), the IQT trees can reduce the communication cost significantly. We can further find that when the query rate increases from 0.4 to 0.2, the total costs of IQT trees almost do not increase, because the IQT tree can make the query cost as low as possible.

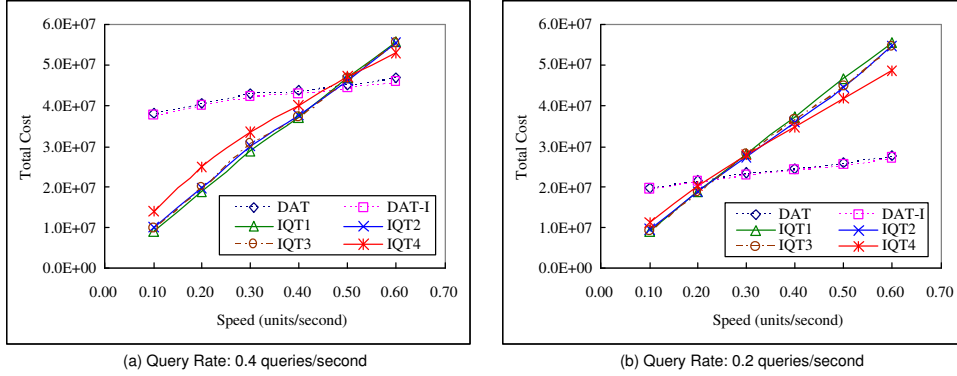


Figure 5.6: The impact of objects' speeds.

On the other hand, when the query rate increases from 0.4 to 0.2, the total costs of DAT trees are doubled. However, when the speed is high enough, the DAT trees still outperform the IQT trees, because the DAT tree is optimized by minimizing the update cost.

To get further insight into the performance of IQT, four IQT trees with different settings are compared with each other in Fig. 5.6. We can see that when the query cost dominates the total cost, the value of α should be low, because more sensors will be non-backbone nodes that will be considered to reduce the query cost. In addition, the value of γ should be low, because this will make sensors' parents close to the sink and reduce the length of query paths. Finally, the *min_confidence* should be high enough such that the queries can indeed be responded early. On the contrary, when the update cost dominates the total cost, the values of α , β and γ should be large, and the value of *min_confidence* should be low such that the constructed tree will be like to the DAT one. Later, we will investigate more settings under different query scenarios.

We also observe the impact of query rates in Fig. 5.7. We can see that the total costs even are decreased slightly by using IQT trees optimized by reducing the query cost. On the other hand, the total costs of DAT trees are increased when the query rate becomes high.

We find that by optimizing the query cost, the IQT tree also benefits from low

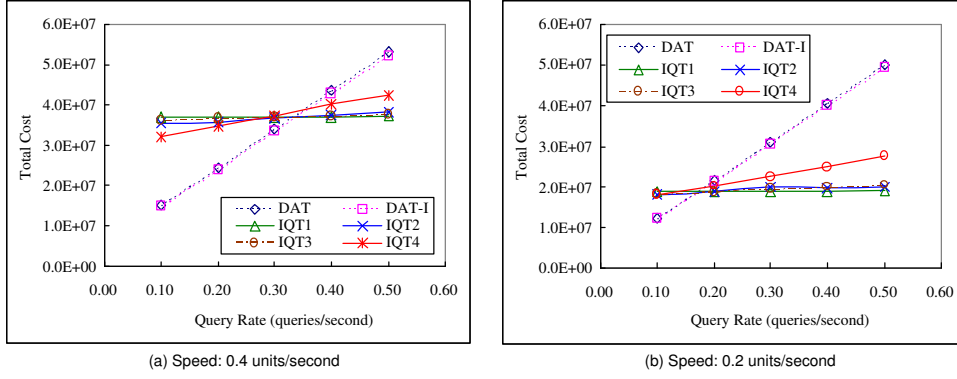


Figure 5.7: The impact of query rates.

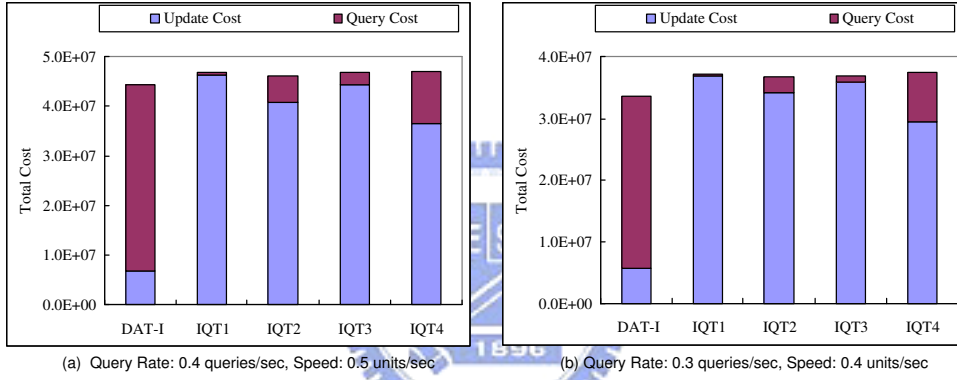


Figure 5.8: Comparison of ratios of update cost to query cost.

query response time. In Fig. 5.8, we consider two cases in which the IQT tree and the DAT tree have similar performances in terms of total cost. We find that the query cost of IQT trees can be reduced significantly. This implies low query response time to which users are sensitive.

Now we consider a scenario in which some objects will be queried frequently such that there are some query hotspots in the sensing field. Fig. 5.9(a) and Fig. 5.10(a) show the results of the scenario in which each object is queried evenly. (Note that the settings of parameters $\langle \alpha, \beta, \gamma, min_confidence \rangle$ used in IQT5 and IQT6 are $\langle 0.5, 1.0, 0.3, 0.9 \rangle$, and $\langle 0.8, 1.0, 0.3, 0.9 \rangle$ respectively. Further note that in Fig. 5.9, the query rate is set to 0.4 queries/second and in Fig. 5.10, the objects' speed is set to 0.4 units/second.) On the other hand, Fig. 5.9(b) and

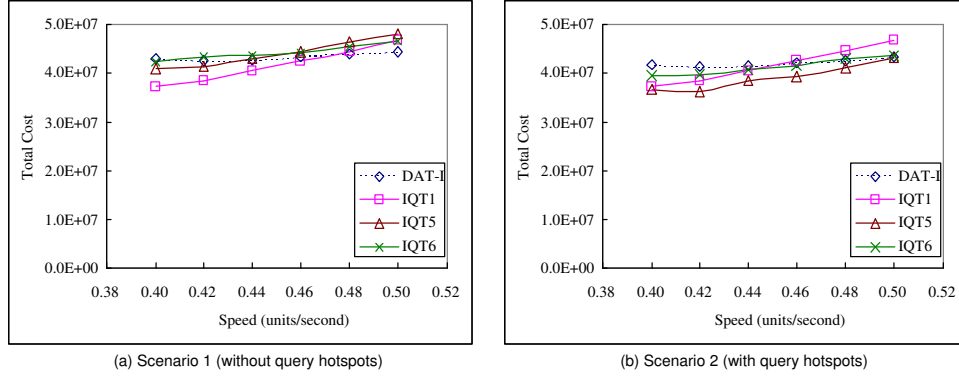


Figure 5.9: The impact of objects' speeds under two different query scenarios

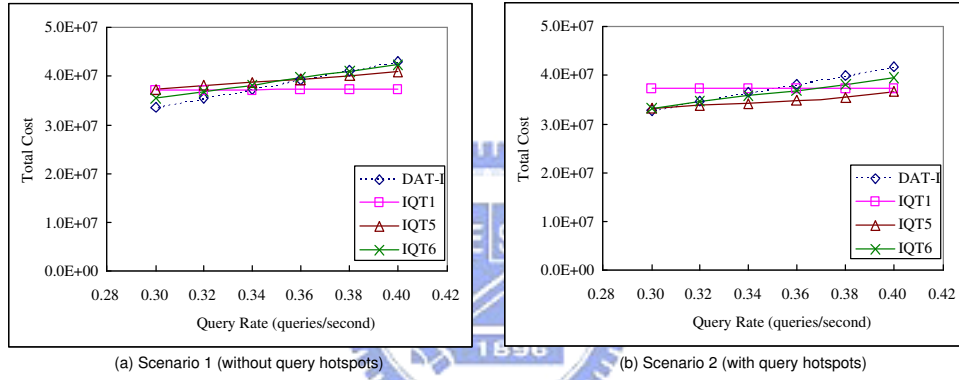


Figure 5.10: The impact of query rates under two different query scenarios.

Fig. 5.10(b) show the result of the scenario in which some objects are queried frequently. In the first scenario, one may argue that γ should be larger than or equal to β , because some backbone nodes will be useless (i.e., no non-backbone nodes will connect to them) when γ is less than β . However, in the second scenario, it is useful to make γ less than β . Sensors in the query hotspots are queried frequently. Thus, it is better to select them to be non-backbone nodes and select other sensors to be backbone node. Thus, even no non-backbone node connects to some backbone nodes, sometimes it is still better to keep them as backbone nodes.

5.4 Summary

By exploiting the nature of imprecision of sensor data, we propose an imprecision-tolerant location management model for object tracking sensor networks. The proposed model consists of imprecision-tolerant update and query mechanisms that can be used to support imprecision-tolerant queries. By exploiting the feature of the tree-based location management schemes, the proposed model can provide multiple imprecision levels and ensure that the quest cost will be proportional to the imprecision level. In addition, we develop a tree construction algorithm to facilitate the proposed location management model, which can reduce the query cost while minimize the increment of update cost. Finally, we have demonstrated the efficiency of the proposed model by simulation.



Chapter 6

A Link-layer Protocol for Event-driven WSNs

By simulation, we observe that packet loss may make the location information incorrect in object tracking sensor networks. Thus, we also propose a link-layer protocol to relieve the contention and collision problems for event-driven WSNs. We solve these problems by jointly considering two subissues. One is exploiting the spatial correlation of data reported by sensors in the event area, and the other is designing a specific MAC protocol.

6.1 Preliminaries

6.1.1 Background and Motivations

Depending on the reporting behavior, wireless sensor networks (WSNs) can generally be classified into two categories: time-driven and event-driven. In a time-driven WSN, sensors report their sensed data periodically to the sink. Such behavior usually exhibits a unique *funneling* effect [1], where sensors near the sink may suffer from higher contention. Some approaches have been developed to solve this problem [1, 32]. On the contrary, in an event-driven WSN, sensors report only when they detect events. In such behavior, the traffic near the sink may not be heavy, but sensors in the *event area* may suffer from higher contention, because they are likely to detect, and thus intend to report, events simultaneously.

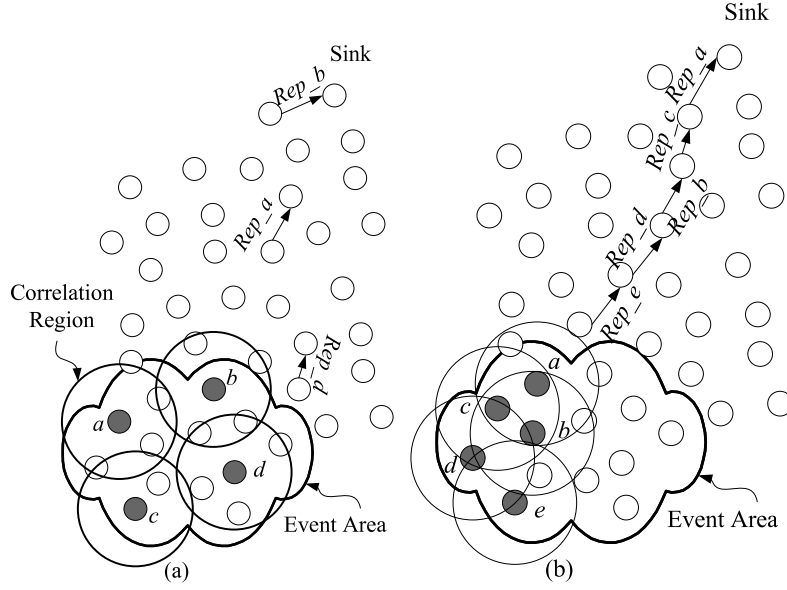


Figure 6.1: Two examples of event reporting in an event-driven WSN.

Our goal is to solve the contention problem of event reporting in event-driven WSNs. The proposed approach joints two subissues. One is exploiting the spatial correlation of data reported by sensors in the event area and the other is designing a specific MAC protocol. Due to the spatial correlation of sensor data, nearby sensors typically have similar values. Thus, it may not be necessary for every sensor to report its sensed data. Exploiting the spatial correlation of sensor data to reduce redundant reports has been studied in [19, 21, 25, 31, 36]. For example, in Fig. 6.1(a), the cloud area denotes the event area, and each sensor x is associated with a correlation region, in which sensors' readings are highly correlated with x (the big circles denote the correlation regions of the gray sensors). We can see that it is sufficient to have sensors a , b , c , and d to report to cover the event area. In addition, we can also note that the selection of reporting sensors needs to be done carefully. For example, in Fig. 6.1(b), the five gray sensors are insufficient to cover the whole event area.

The second subissue is to design a specific MAC protocol. After reporting sensors are selected, we need to reduce the contention and collision among these

reporting packets and to minimize the latency in transmitting these packets. Contention and collision are likely because these sensors may be close to each other. In addition, because packets in WSNs are typically small, using the RTS/CTS mechanism to avoid the hidden terminal problem is not preferred. This also makes the collision problem more severe. Besides, since these packets are likely to share common paths when moving toward the sink, we would like to see *pipeline effect* such that these packets are separated spatially (e.g., Fig. 6.1(a)), but move sequentially (e.g., Fig. 6.1(b)), along these paths (we will elaborate more on this later). Thus, designing a specific MAC protocol for event-driven WSNs is required.

In this dissertation, we propose a schedule-based approach to exploit the spatial correlation of sensor data on the link layer. We do not modify the MAC protocol directly. On the contrary, we develop a scheme for making report decision and a protocol for transmitting reporting packets on the link layer. By doing so, any CSMA-based protocol designed for WSNs could be adopted as the underlying MAC protocol; thus, we can simply leave some issues (e.g., power saving) to the MAC protocol itself. In our approach, a node has two modes: *ES (Event-Source) mode* and *NES (Non-Event-Source) mode*. Initially, each sensor is in the NES mode. On detecting an event, a sensor will enter the ES mode and adopt a schedule-based protocol to transmit its packets. (The schedule-based protocol can be regarded as a TDMA-based protocol, but strictly speaking it is built on top of a CSMA-based protocol.) The rationale behind this is to avoid contention and to form the pipeline effect as illustrated in Fig. 6.1(a). This schedule-based protocol has some characters that makes it different from conventional TDMA-based protocol. First, the TDMA part is based on very loose time synchronization and is triggered by the appearance of events. Second, the slot assignment strategy associated with the TDMA part takes the spatial correlation of sensor data into consideration and thus allows less strict slot allocation than conventional TDMA schemes. Interestingly, by intentionally allowing one-hop neighbors to share the same time slot, the number of slots required per frame is significantly reduced. Third, by enlarging the slot size on purpose, our scheme enforces packets, after

leaving the event area, to form a pipeline in such a way that packets flows are like streams, each of which is separated sufficiently in distance to avoid interference. In addition, a scheme is devised to exploit the spatial correlation of sensor data. Specifically, by exploiting TDMA's features, redundant reports can be further reduced with and without the aid of overhearing. On the other hand, because not all sensors in the NES mode have to transmit packets, sensors in the NES mode will adopt the original CSMA-based protocol to minimize the delay. Finally, we will also discuss how to achieve energy efficiency by combining our protocol with the LPL (Low Power Listening) technique proposed in the B-MAC [20].

6.1.2 Some Observations

In this section, we assume that CC-MAC [31] is adopted in an event-driven WSN but the RTS/CTS mechanism is removed. In order to motivate our work, we make some observations from the interference and the spatial correlation aspects.

From the interference aspect, we raise two scenarios to show that the hidden terminal problem will be very serious in an event-driven WSN. First, as shown in Fig. 6.2(a), when two sensors two-hops apart detect an event at the same time, their reports may collide even though their receivers are different. Second, even for sensors not in the event area, collisions are inevitable as packets move toward the sink. Fig. 6.2(b) shows an example with a report tree. Without the aid of RTS/CTS, we can see that *Report_1* could collide with *Report_2* at sensor *D*, *Report_3* and *Report_4* could collide at sensor *G*, and *Report_5* could collide with *Report_6* at sensor *I*. Thus, the interference is serious for sensors in the event area, as well as those far away from the event area. We can see that designing a specific MAC protocol for event-driven WSNs in which the RTS/CTS mechanism is disabled is required.

From the spatial correlation aspect, we argue that using inter-distance between sensors is insufficient to decide who shall report. First, a simple example is illustrated in Fig. 6.3(a), where sensor *y* is near the boundary of sensor *x*'s correlation

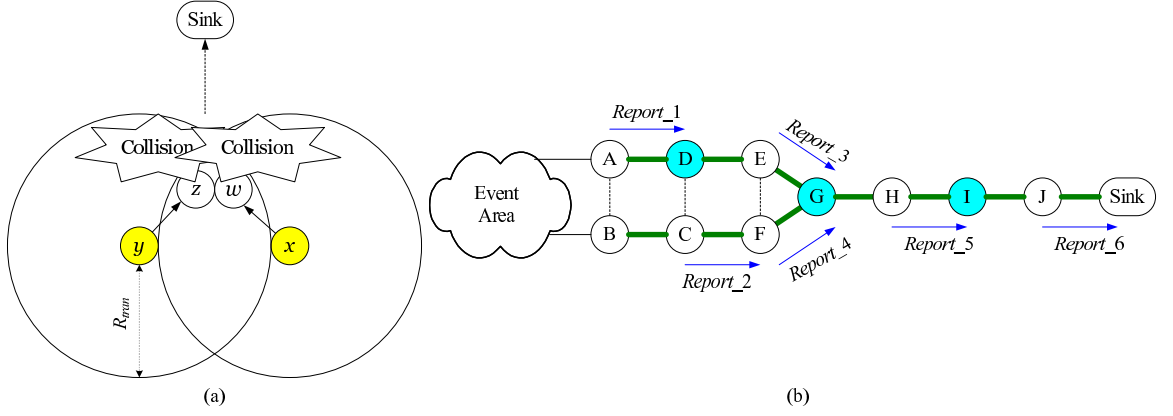


Figure 6.2: The hidden terminal problem, where R_{tran} denotes the transmission range of sensors.

region. With CC-MAC, y has to report no matter whether it overhears x 's report or not. We can see that the overlap area of x 's and y 's correlation regions is about 39% of one correlation region, which is high. A more sophisticated example is further shown in Fig. 6.3(b). Assuming that sensors a , b , c , and x have already reported, we consider two scenarios. First, if y does not overhear any of those reports (we can see that y is not in any of the transmission regions of a , b , c , and x), then y will report. However, y 's report does not contribute any additional area to existing reports. Second, even if y can overhear x 's report (this is possible when f forwards x 's report), CC-MAC will enforce y to report, because the distance between x and y is larger than R_{corr} . Therefore, a more sophisticated report reduction scheme is required. In addition, because overhearing is opportunistic sometimes, this sophisticated scheme should not highly rely on overhearing.

6.2 The Proposed Schedule-based Approach

6.2.1 Overview

We assume that a CSMA-based MAC protocol is adopted as the underlying MAC protocol. In order to solve the contention problem and the hidden terminal problem in the event area, a schedule-based (or a TDMA-like) approach is proposed.

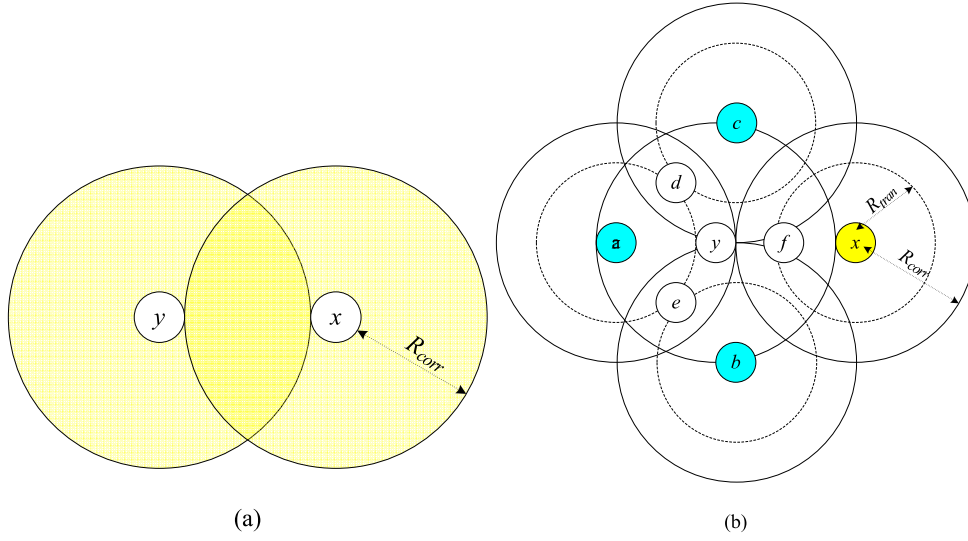


Figure 6.3: The redundancy problem.

However, for those sensors not in the event area, because not all sensors need to help forward packets, assigning slots to those sensors that do not intend to transmit any packet is unnecessary and could increase delay. This means that the TDMA-like approach may not be suitable for sensors not in the event area. Instead, they will take a CSMA-based MAC approach. Therefore, our scheme can be regarded as a hybrid TDMA/CSMA protocol.

Each node has two modes: event-source (ES) mode and non-event-source (NES) mode. Sensors in the ES mode will adopt a schedule-based approach to transmit packets. Issues involved in the ES mode include: (i) when to enter the ES mode, (ii) how to design a good slot assignment strategy, (iii) how to determine proper slot size, (iv) what will lead to synchronization error and how to conduct time synchronization, (v) how to exploit the spatial correlation of sensor data, and (vi) when to leave the ES mode.

On the other hand, sensors in the NES mode will adopt the original CSMA-based protocol to reduce report latency. An issue involved in this mode is how to alleviate congestion when multiple events occur simultaneously.

6.2.2 Operations in the ES Mode

Entering the ES Mode

As we mentioned above, a CSMA-based MAC protocol is adopted as the underlying MAC protocol, but sensors in the ES mode will adopt a TDMA-like protocol. To implement this TDMA-like protocol, we divide the time into slots at the link layer. When the network was deployed, a slot assignment algorithm will be run so that each sensor will be assigned a slot. Slots are numbered from 1 to *MAXSLOT*, and the first slot (i.e., slot 1) will be started on-the-fly in an event-driven manner.

Initially, each sensor is in the NES mode. When a sensor detects an event, it will enter the ES mode by starting slot 1. Then, it will count slots until its slot arrives. If its slot arrives and it intends to transmit a packet, it will perform the access mechanisms defined in the underlying CSMA-based MAC protocol (such as the backoff and CCA mechanisms) as usual to access the channel. If a sensor cannot send its packet in its current slot, it will wait for its slot in the next cycle (note that it is also possible that the sensor will suspend its packet and not retry again).

Slot Assignment Strategy

Conventional TDMA-based protocols will assign each node a slot different from those of its one-hop and two-hop neighbors. We argue that when the spatial correlation of sensor data is taken into consideration, such a strategy may not be efficient, because not every sensor needs to report. It is easy to see that if a node finally decides not to report, then its assigned slot is wasted. In addition, our TDMA-like protocol is built on top of a CSMA-based protocol. Thus, with proper backoff, assigning the same slot to neighboring nodes does not necessarily lead to collision. Even, in case neighboring nodes share the same slot, the backoff mechanism can be used to determine who should report and the losers may suspend their reports due to the spatial correlation of sensor data when they overhear the packet sent by the winner. Therefore, we even intentionally assign a slot used

by a node's one-hop neighbors to that node, but the node still should be assigned a slot different from those used by its two-hop neighbors to avoid the hidden terminal problem without the aid of RTS/CTS. Another advantage of our proposed slot assignment strategy is that the value of *MAXSLOT* required can be reduced significantly.

We develop a simple distributed slot assignment algorithm to complement this slot assignment strategy. Note that the algorithm is run only once when sensors are first deployed. We make some assumptions. First, the network topology is static (otherwise, the slot assignment algorithm has to be run after topology change). Second, each sensor has a unique ID. Third, each sensor can correctly discover all its one-hop and two-hop neighbors. Finally, the number of two-hop neighbors of a node is finite.

Each sensor will maintain a *slot usage table* to record the slots used by its one-hop and two-hop neighbors. Each sensor that does not own a slot will select its slot in a distributed way. Thus, we only describe the behavior of a sensor x . First, x will send a *request* to all of its two-hop neighbors. Any two-hop neighbor y of x receiving the *request* will act as follows.

- If y does not own a slot yet and $y.ID < x.ID$, then y will reply a *grant* to x . Because y does not own a slot yet, null slot information will be carried on the *grant*.
- Otherwise, y will do nothing.

Once x receives *grants* from all of its two-hop neighbors, x will select a slot by the following rule. To begin with, x will check whether there exists a slot such that this slot has been assigned to x 's one-hop neighbors but has not been assigned to x 's two-hop neighbors. If such slots exist, x will pick up the most-used one among those slots. (Recall that we will intentionally assign a slot used by a node's one-hop neighbors to that node.) Otherwise, x will select the smallest slot that has not been used by its two-hop neighbors. (The reason is to minimize *MAXSLOT*.)

After selecting its own slot, x will send a *grant* with its selected slot to each of its two-hop neighbors. Then, when another node z receives such a *grant* on which a selected slot is carried from one of its two-hop neighbors, z will modify its slot usage table accordingly and check whether it has received all grants from all of its two-hop neighbors.

Finally, when a sensor determines its slot, it will notify the sink so that the sink can determine the value of $MAXSLOT$. Then the sink will announce $MAXSLOT$ to all sensors. Note that when no packet loss occurs, the proposed slot assignment algorithm guarantees that each node will receives *grants* from all of its two-hop neighbors. However, when packet loss cannot be avoided, a sensor may loss *grants* and this will result in deadlock. In order to overcome the packet loss problem, a sensor can actively ask its two-hop neighbors to resend their *grants* (if allowed) when it waits passively for a long period.

Theorem 9. *The proposed slot assignment algorithm ensures that each sensor will select a slot different from those used by its two-hop neighbors.*

Proof. For simplicity, we assume that packet loss will not occur. Because each sensor has the same behavior, we only consider a sensor, say x . We assume that x has n two-hop neighbors and the ID of x is the k -th largest one among these $n + 1$ sensors, where $1 \leq k \leq n + 1$. We will show that x can select a slot different from those used by all of its two-hop neighbors no matter what the value of k is.

We consider two cases. In the first case, we assume $k = 1$. In this case, when x sends a *request* to all of its two-hop neighbors, each of x 's two-hop neighbors will reply a *grant* to x , because x has the largest ID. Thus, x will choose slot 1 (i.e., the smallest slot) to use. It is easy to see that all of x 's two-hop neighbors cannot select their slots because they cannot get x 's grant. Then, x will send a *grant* with the slot number selected by x to all of its two-hop neighbors; thus all of x 's two-hop neighbors will not select slot 1 to use. Finally, we can see that the *grant* sent by x will make some x 's two-hop neighbors whose IDs are the second largest among their two-hop neighbors get all required *grants* and start to select

their own slots.

Now we consider the second case. In this case, we assume $1 < k \leq n + 1$. In this case, when x sends a *request* to all of its two-hop neighbors, those x 's two-hop neighbors whose IDs are smaller than $x.ID$ will send a *grant* to x . However, those x 's two-hop neighbors whose IDs are larger than $x.ID$ will send a *grant* to x only when they have determined their slots. Thus, x will not use the same slot with them. (Note that it is possible that two of x 's two-hop neighbors use the same slot if these two nodes are not two-hop neighbors with each other.) After x selects its slot, x will send a *grant* with the slot number selected by x to all of its two-hop neighbors; thus those x 's two-hop neighbors whose IDs are smaller than $x.ID$ will not select the slot the same with x . Finally, we can see that the *grant* sent by x may make some sensors get all required *grants* and start to select their own slots. □

To verify the efficiency of the proposed slot assignment strategy, a simple simulation is conducted. 4096 sensors are randomly deployed in a 256×256 field with uniform distribution. As we can see in Fig. 6.4, when the transmission range of sensors increases, the value of *MAXSLOT* increases from 45 to 301 when a node needs to have a slot different from those used by its one-hop and two-hop neighbors. (Note that our slot assignment algorithm can be easily modified to support this strategy.) However, the value of *MAXSLOT* only increases from 14 to 23 when a sensor only needs to differentiate from its two-hop neighbors. Note that a lower *MAXSLOT* means a lower report latency.

Slot Size

In conventional TDMA protocols, the slot size is usually set to the maximum one-way message delay denoted by d . (Note that in our approach, d should include the maximum backoff delay, the time to perform CCA, and so on.) Below, we will show that the hidden terminal problem could be alleviated by prolonging the slot size. Before that, we define a term *flow*.

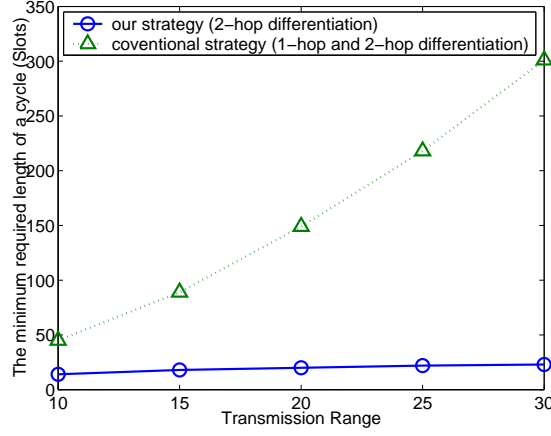


Figure 6.4: Comparison of slot assignment strategies.

Definition 1. Consider any sensor x that is in the ES mode and located at the event boundary. When x transmits a packet to a neighboring sensor that is in the NES mode, we say a flow is generated.

In the proposed approach, the slot size is set to $\ell \times d$, where ℓ is a real number larger than or equal to 1. Note that ℓ is usually set to be 1 in most TDMA-based protocols. Fig. 6.5 shows an example of the advantage of $\ell > 1$. In Fig. 6.5, E and F are in the ES mode and their assigned slots are i and $i + 1$ respectively. Suppose that in some cycle E generates a flow in slot i and F generates a flow in slot $i + 1$. When C receives the packet sent by E , it will run a CSMA-based protocol immediately to forward the packet because it is in the NES mode. Fig. 6.5(b) shows the case of $\ell = 1$, where the transmission of F could easily collide with the transmission of C at D due to the hidden terminal problem. However, as Fig. 6.5(c) shows, if we set $\ell = 2$, the transmission of C will occur within slot i , thus avoiding the hidden terminal problem.

The purpose of prolonging the slot size is to separate flows in the time domain. The advantage can be illustrated by Fig. 6.2, where we assume that *Report_1*, *Report_3*, and *Report_5* belong to flow 1, and *Report_2*, *Report_4*, and *Report_6* belong to flow 2. We can see that the hidden terminal problem in the non-event area can be avoided when these two flows are separated. Fig. 6.6 shows a more

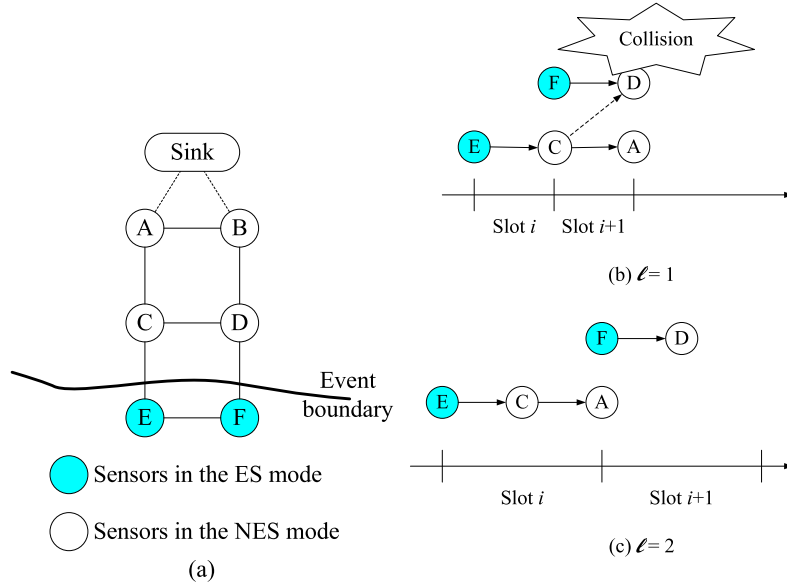


Figure 6.5: The impact of slot size beyond event areas.

general example, where the pipeline effect will be formed when the ℓ is large enough. (As we can see that in Fig. 6.6(a), when ℓ is small, packets will move sequentially; on the other hand, in Fig. 6.6(b), when ℓ is large, the *pipeline effect* will be formed.) However, a larger slot size also incurs longer delay in the event area. Therefore, determining a proper value of ℓ is an important question. We will investigate how to choose a proper ℓ by simulation.

Synchronization

Time synchronization should be done in a strict way in conventional TDMA-based protocols. However, tight clock synchronization is not required in our protocol. There are two major reasons. First, our TDMA-like protocol is built on top of a CSMA-based MAC protocol. This means that the backoff scheme and the CCA (Clear Channel Assessment) scheme can remove most of the collisions caused by synchronization error. Second, we use longer slot size to separate flows. Thus, we can tolerate a certain degree of synchronization error. For example, in Fig. 6.7(a), where $\ell = 3$ and two sensors do not synchronize with each other, we can see that

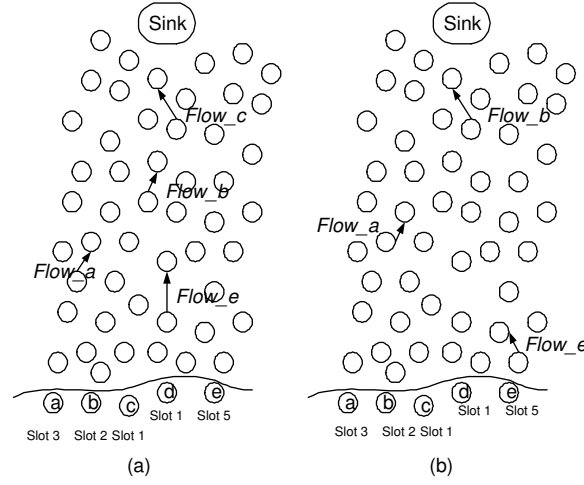


Figure 6.6: The advantage of separating flows in time.

no matter which slots are used by A and B , there is no collision between A and B .

In our scheme, sensors are assumed to be synchronized by the occurrence of events, which trigger them to enter the ES mode. This scheme has two problems that may lead to synchronization error.

- Sensors may not enter ES-Mode simultaneously due to the event propagation delay.
- When multiple events occur close in time and space, some sensors may detect multiple events. In our scheme, when a sensor in the ES mode detects another event, we will allow it to continue its slot counting, instead of resetting to slot 1. On the contrary, some sensor may only detect one event and enter slot 1. This will also lead to synchronization error.

In order to solve these two problems, a simple adjustment scheme is proposed. We assume that each sensor will count how many slots have passed after it entered the ES mode. This counter is denoted by s . When a sensor x transmits a packet in the ES mode, the counter $x.s$ will be carried in the packet. Each of x 's neighbor sensors, say y , that overhears the packet will react as follows. (Note that when

y receives the packet, it can easily know the packet is transmitted by x in the $((x.s - 1) \bmod \text{MAXSLOT}) + 1$ -th slot of a cycle.)

- If $y.s > x.s$, then y will do nothing.
- If $y.s = x.s$, then y will fine-tune itself as follows. First, y will estimate the start time of x 's slot. If x is slower than itself, then nothing will be done. Otherwise, y will shorten its current slot to synchronize with x . An example is shown in Fig. 6.7(b).
- If $y.s < x.s$, then y will estimate the start time of x 's slot, adjust its current slot to $((x.s - 1) \bmod \text{MAXSLOT}) + 1$, set $y.s$ to $x.s$, and fine-tune itself. An example is shown in Fig. 6.7(c).

With our adjustment scheme, when multiple events occur close in time and space, the sensors that detect the earliest event will dominate the clock in the ES mode. Although collisions could occur during the adjusting, the backoff and CCA mechanisms and the design of longer slot size can alleviate the collision problem.

Finally, one should note that we cannot use the slot number assigned to sensors to correct the synchronization error. For example, in Fig. 6.7(d), when B overhears the packet transmitted by A , it will switch to slot 10. Later on, when B overhears the packet transmitted by C , it will switch back to slot 9. Thus, the counters rather than the slot number should be used in the adjustment scheme.

Exploiting the Spatial Correlation of Sensor Data

So far, we mainly focus on the medium access issue. Next, we discuss how to exploit the spatial correlation of sensor data. We assume that a correlation radius R_{corr} is given by applications and our goal is to minimize redundant reports under distortion constraints. We propose a report reduction scheme that provides two advantages. First, by exploiting TDMA's features, we adopt a probability to reduce redundant reports without the aid of overhearing. Second, compared to

CC-MAC, when a sensor overhears a packet, the area of the overlap of correlation regions will be taken into consideration.

Our report reduction scheme consists of three steps. The first step is executed when a sensor detects an event. The sensor will use a probability to determine whether it should report or not (note that if the sensor decides not to report, it still needs to enter the ES mode). Then, it will enter the second step, during which it will try to overhear others' packets before its slot arrives. With overhearing, some reports can be further discarded in this step. Finally, when the sensor's slot arrives, it will enter the third step in which it will transmit one of the packets in its buffer, if any. The details of this scheme are described as follows.

Step 1: Because overhearing is opportunistic or even impossible sometimes, it is hard for a sensor to collect enough information to judge whether it should report or not. Thus, a probability is adopted to help sensors to decide whether they should report or not. In our scheme, when a sensor detects an event, it will report this event with a probability α^{S-1} , where $0 < \alpha \leq 1$ and S is the slot number assigned to that sensor. This means that a sensor with larger slot number may tend to not report. To motivate this design, let's reconsider the example shown in Fig. 6.3(b), where we assume that sensors a , b , c , and x have already reported. When our TDMA-like protocol is applied, this means that y may have a larger slot number than a , b , c , and x do. Recall that we have shown that y 's report is redundant. Based on this observation, sensors with larger slot number should tend to not report, because their neighbors may have reported. Note that this step can reduce redundant reports without the aid of overhearing.

An important issue in the first phase is determining a proper α , which needs to consider many factors, for example, the ratio of R_{corr} to R_{tran} , network density and so on. It can be observed that when the value of R_{corr}/R_{tran} becomes larger, the value of α should be decreased, because the probability that a sensor suspends its report by overhearing becomes less. Besides,

network density will also affect the optimal value of α . When the network is dense, the value of $MAXSLOT$ may become larger. This will impact the distribution of slot numbers assigned to sensors and then the value of α . (Recall that the probability depends on the slot number assigned to the sensor.) Therefore, we can know that there are many factors needed to be considered to determine the optimal value of α . To simplify this problem, we suggest that α should be a tunable parameter. Given a distortion constraint, when the sink receives many redundant reports, it can decrease the value of α and announce the new value of α to sensors. (Note that although the sink cannot know the exact boundary of event area, it can compute the overlap area of correlation regions to judge the redundancy level.) Besides, when the distortion is the major concern, the sink can just set $\alpha = 1$.

Step 2: No matter whether a sensor decides to report or not, the sensor will execute the procedure in the second step before its own slot arrives. To begin with, we define the *reporter* of a packet. The reporter of a packet is the sensor that first initiates this report packet by detecting an event. (Note that the sender of a packet may not be the reporter of that packet.)

The procedure in the second step is as follows. A sensor x will try to overhear others' packets. When x overhears a packet (whose reporter is denoted by $r_{received}$), x will check all of the packets in its buffer. We assume the reporters of the packets in x 's buffer are denoted by $\{r_1, r_2, \dots, r_k\}$, where k is the number of packets in x 's buffer. According to the distance between $r_{received}$ and r_i , where $i = 1, \dots, k$, three cases are considered separately:

- If the distance is smaller than R_{corr} , then the packet reported by r_i will be removed from x 's buffer.
- From Fig. 6.3(a), we can observe that when the distance between two reports is smaller than $2 \times R_{corr}$, their correlation regions will overlap. Based on this observation, when the distance between $r_{received}$

and r_i is larger than or equal to R_{corr} but smaller than $2 \times R_{corr}$, x will determine whether the packet reported by r_i should be removed or not with a probability $INTC(d)/\pi(R_{corr})^2$, where d denotes the distance between $r_{received}$ and r_i and $INTC(d)$ is the intersection area of the two circles centered at $r_{received}$ and r_i . $INTC(d)$ can further be represented by $4 \int_{d/2}^{R_{corr}} \sqrt{(R_{corr})^2 - x^2} dx$.

- If the distance is larger than or equal to $2 \times R_{corr}$, nothing will be done.

Three notes regarding the second step should be addressed. First, if the packet overheard by x is destined to x itself, the packet will be inserted into x 's buffer. Second, the procedure in step 2 will also be run on the sensors that are in the NES mode. More precisely, the procedure in step 2 will be executed whenever a sensor overhears a packet. Third, compared to CC-MAC, we argue that our TDMA-based design can increase the opportunity of overhearing inherently. We can see that before a sensor reports its data, it has to wait until its own slot arrives. During the waiting period, the sensor could overhear a packet from other sensors and suspend its report.

Step 3: When the sensor's slot arrives, it will enter the third step in which it will transmit one of the packets in its buffer, if any.

Leaving the ES Mode

The final issue is how long a sensor should stay in the ES mode. Basically, a sensor can return to the NES mode when it has reported or decided not to report the detected event. However, it is possible that some of its one-hop/two-hop neighbors need it to help forward their packets. If the sensor returns to the NES mode too quickly, the possibility of interference and collision may increase when it helps forward packets. Thus, we suggest that an ES mode sensor can return to the NES mode when it does not have any packet in its buffer and does not receive/overhear any packet during a cycle.

6.2.3 Operations in the NES Mode

Sensors in the NES mode will adopt a CSMA-based protocol to reduce the report latency. Recall that in the ES mode, we have tried to separate flows to proceed in a pipeline manner. Thus, collision and congestion should have been avoided. However, when multiple events occur close in time, flows that belong to different events may collide with each other. In addition, a late flow may pursue an earlier flow when congestion occurs. This makes the pipeline effect no longer available.

A way to solve this problem is to use a mechanism similar to that in Z-MAC [24], that is, to adopt a TDMA-based scheme when a sensor experiences high contention. However, control packets are required to achieve. In this dissertation, motivated by [9], we propose an alternative simple scheme called *wait-and-fusion* that does not require any control packet. When a sensor, say y , in the NES mode experiences congestion ([24] has proposed some approaches to help a node determine whether it experiences congestion), y will buffer the received packets and wait an opportunity for fusion. Specifically, when y receives a packet whose reporter is a , it will check whether there exists a packet in its buffer whose reporter, say b , has a distance less than $\lambda \times R_{corr}$ from a and $|data(a) - data(b)| < \beta$, where $data(\cdot)$ denotes the reading of a sensor. If such a packet exists, these two packets will be fused into one report with a reading $f(a, b)$. Note that function f and the values of λ and β are application-specific. Such a fusion behavior will continue for a while until the congestion problem is relieved. With the wait-and-fusion scheme, we expect that the collision and congestion problems can be alleviated.

6.2.4 Extension for Achieving Energy Efficiency

Due to the power constraint of sensor nodes, energy efficiency is also an important issue for WSNs. As mentioned above, most CSMA-based MAC protocols can be adopted as the underlying MAC protocol. Below, we will use B-MAC [20] as our choice and show how to utilize its LPL (Low Power Listening) technique to achieve energy efficiency.

In LPL, a sensor normally stays in the sleep state and wakes up periodically. When a sensor wakes up, it will turn on its radio for a very short duration and check for any activity. If a preamble is detected, the sensor will stay awake to capture the incoming packet. Since nodes are not synchronized and thus wake up at different times, the preambles of data packets should be longer than the check interval of sleeping nodes to ensure that sleeping nodes will not miss incoming packets. More details of LPL can be found in [20].

Below, we make some notes about the combination of our scheme with LPL technique. First, both the TDMA-like protocol and the periodical wake-up scheme need timers. These two timers should be run independently. Besides, the maximum one-way message delay (i.e., d) should include the preamble length. Fig. 6.8 shows an example, where $\ell = 1$. Second, recall that CCA needs to be run before any transmission. If the CCA outlier algorithm observes that the channel is not clear, the sensor should switch to the receive mode instead of going back to sleep. The reason is that our scheme depends on overhearing for inhibiting reporting and increasing data fusion opportunity.

6.3 Simulation Results

We have developed a simulator to demonstrate the efficiency of our proposed approach. A sensing field with size 256×256 units where 4096 sensors are deployed randomly with uniform distribution is simulated. The sensor with ID 0 is selected to be the sink. In order to simulate the events arising in the network, a simple event generation model is proposed. In this model, we use four parameters to control the generation of events:

- **MAX INTERVAL:** This parameter defines the maximum time interval between two events.
- **WIDTH and MAX LEVEL:** In our model, an event area is represented by multiple concentric circles. The number of concentric circles is determined

by **MAX_LEVEL**. The first circle is the one with radius **WIDTH**, the second circle is the one with radius $2 \times \mathbf{WIDTH}$, and so on.

- **PROPAGATION_DELAY**: This parameter is used to simulate the event propagation delay. When an event occurs, if sensors in the i -th annulus of the event area detect this event at t_i , then sensors in the $i + 1$ -th annulus will detect this event at $t_i + \mathbf{PROPAGATION_DELAY}$.

Now, we describe the procedure of this event generation model. The first event will be triggered at the beginning of simulation. As we mentioned above, an event area is represented by multiple concentric circles. Therefore, a point in the sending field will be selected randomly as the center of those circles. Sensors in the first circle will detect this event first. Then, after **PROPAGATION_DELAY**, sensors in the second annulus will also detect this event. This detection procedure will continue until sensors in the **MAX_LEVEL**-th annulus detect this event. Finally, when an event e_j arises initially, the next event (i.e., event e_{j+1}) will also be triggered after t , where $0 \leq t \leq \mathbf{MAX_INTERVAL}$ and t is determined randomly with uniform distribution.

Three metrics are used to evaluate the performance of medium access schemes. We count *the number of packets transmitted*. Usually, fewer packets means that sensors can stay in sleep mode longer. Thus, less energy is consumed. We also measure *the success rate of packet transmission* defined as the ratio of the number of packets received by the intended receiver to the number of packets transmitted by the sender. Success rate can be used to evaluate the efficiency of a MAC protocol. Higher success rate means less collision. *Average delay* is defined as the average delay of report packets received by the sink. Besides, two metrics are used to evaluate the performance of report reduction schemes. *Coverage* is defined as $A_{corr_reg_union}/A_{event_area}$, where $A_{corr_reg_union}$ denotes the area of the field united by the correlation regions of reporters whose reports are received by the sink, and A_{event_area} is the area of event area. Higher coverage means that the sink has more accurate information regarding events. Finally, for a unit area in

Table 6.1: Parameters used in the simulation for our proposed link-layer protocol.

Buffer Size	10
The length of DATA	30 Bytes
Bit rate	250 kb/s
Simulation Time	1 hour
<i>MAX INTERVAL</i>	10 seconds
<i>WIDTH</i>	10 units
<i>PROPAGATION DELAY</i>	5 milliseconds
<i>MAX LEVEL</i>	5 (Default)
ℓ	1.5 (Default)

the event area, if it is covered by n sensors' correlation regions, where $n > 1$, then we define that the *redundancy* of that unit area is $(n - 1) \times 100\%$.

First, we compare our proposed schedule-based approach with a CSMA-based protocol. Also, two report reduction schemes and two slot assignment schemes will be applied separately. The detail will be described later. Then, we further investigate the impact of two parameters used in our proposed approach, that is, α and ℓ . The related parameters used in the simulation are shown in Table 6.1.

6.3.1 Evaluation of SC-MAC

In this section, we compare our proposed schedule-based approach called SC-MAC (a MAC protocol with Spatial Correlation consideration) with several schemes. In the CSMA scheme, a CSMA-based MAC protocol without any spatial correlation consideration is adopted. In the CSMA-SSC (a CSMA-based protocol with Simple Spatial Correlation consideration) scheme, a CSMA-based MAC protocol with a simple report reduction scheme is adopted. This report reduction scheme works as that used in CC-MAC does. More precisely, when a node, say x , overhears a packet whose reporter is y , x will judge whether the distance between itself and y is smaller than correlation radius or not. If the answer is affirmative, x will suspend its report. Otherwise, x will continue its report. Thus, the CSMA-SSC scheme can be viewed as the simplified version of CC-MAC. In the SCMAC-SCC-TSA scheme, the aforementioned report reduction scheme and our

proposed schedule-based approach will be adopted; however, the Traditional Slot Assignment strategy (i.e., a node needs to own a slot different from those used by all of its one-hop and two-hop neighbors) is used. In the SCMAC-TSA scheme, our proposed schedule-based approach with the traditional slot assignment strategy is adopted. In the SCMAC-SSC, our proposed schedule-based approach with the simple report reduction scheme is adopted. Finally, one should note that the RTS/CTS mechanism is not used in all schemes. In addition, the acknowledgement scheme is also disabled in the simulation.

Fig. 6.9 shows the results of the case where $R_{corr} > R_{tran}$ (R_{corr} is 15 units, R_{tran} is 10 units, and α is set to be 0.5 for the SCMAC-TSA and the SCMAC schemes). To begin with, we focus on the CSMA and CSMA-SSC schemes. Although they have the best performance in terms of average delay, they have the worst performance in terms of coverage, especially when the event area becomes larger. The reason is high contention and collision that make the sink receive fewer reports than expected. This can be further verified by Fig. 6.9(b). We can see that the success rate is low when the CSMA and the CSMA-SSC schemes are adopted. Then, we focus on the SCMAC-SSC-TSA and SCMAC-SSC schemes. In Fig. 6.9(a), we can see that both of them will transmit many packets in the network but they do not provide better coverage than the SCMAC scheme does. The reason can be explained by Fig. 6.9(d). We can see that redundant reports are too much when these two schemes are applied. This means that the simple report reduction scheme does not perform well enough. In addition, because many reports have to be sent, these two schemes also do not perform well in terms of average delay. Finally, we focus on the SCMAC-TSA and SCMAC schemes. In fact, it is hard to compare these two schemes fairly. Although both of them set α to be 0.5, different slot assignment strategies may result in different performances. However, it is not hard to see that the SCMAC scheme has better performance in terms of average delay (note that we can see that the SCMAC scheme transmits more packets than the SCMAC-TSA scheme does). This demonstrates that the proposed slot assignment strategy can reduce the report latency significantly (this

is because the value of $MAXSLOT$ is reduced). Briefly, our proposed SCMAC scheme has the best performance. It provides high success rate, high coverage, low redundancy, reasonable average delay, and reasonable amount of packets.

Fig. 6.10 shows the results of the case where $R_{corr} < R_{tran}$ (R_{corr} is 5 units, R_{tran} is 10 units, and α is set to be 1.0 for the SCMAC-TSA and the SCMAC schemes). In this case, we observe that the coverage will be low when α is set to be smaller than 1.0. Thus, α is set to be 1.0. We can see that the SCMAC scheme is still the best one. In addition, we can further note that the advantage of our proposed slot assignment strategy is revealed thoroughly in this experiment. In Fig. 6.10(c), we can see that our proposed slot assignment strategy can reduce the average delay. This also influences the coverage. In Fig. 6.10(d), we can see that the coverage will become lower when the event area becomes larger. One reason is buffer overflow (note that we assume that each sensor's sending buffer is limited such that for a sensor, if there are too many packets to be sent simultaneously, some of packets will be discarded), because more report packets have to be sent when the event area becomes larger. Long delay will worsen the buffer overflow problem, because packets will be queued in a sensor for a long time. Thus, the performance of the SCMAC and SCMAC-SSC schemes is better than that of the SCMAC-SSC-TSA and SCMAC-TSA schemes.

To conclude, the advantages of our proposed SCMAC scheme can be summarized as follows:

- Our proposed medium access scheme can relieve the collision problem without the aid of RTS/CTS mechanism. This can be verified by high success rate.
- Our proposed report reduction scheme can reduce more redundant reports than the simple report reduction scheme does.
- Our proposed slot assignment strategy can shorten the average delay. This also relieves the buffer overflow problem.

6.3.2 Evaluation of Parameters in SC-MAC

In this section, we further explore the impact of two parameters used in our proposed approach, that is, α and ℓ . To begin with, α is inspected. From the previous subsection, we can draw two conclusions regarding α . First, the optimal value of α depends on the slot distribution. For example, when the traditional slot assignment strategy is adopted, the slot numbers owned by nodes may be large. Thus, higher α should be used to achieve reasonable coverage. Second, when $R_{corr} < R_{tran}$, α should be set to be 1 in order to achieve reasonable coverage, because α is mainly used in the situation where redundant reports cannot be removed by overhearing.

In this section, we further investigate the impact of the ratio of R_{corr} to R_{tran} on the value of α . Fig. 6.11 shows the results where $N1/N2$ denotes that R_{corr} is $N1$ units and R_{tran} is $N2$ units. Note that the set of values of α is $\{0.01, 0.25, 0.50, 0.75, 1.00\}$. From Fig. 6.11(a), we can see that when the ratio of R_{corr} to R_{tran} increases from 1.0 to 2.0, we can use lower α to achieve enough coverage. The reason is that when R_{corr}/R_{tran} becomes larger, the probability that two sensors that are within each other's correlation region and cannot overhear each other's packets directly becomes larger. Thus, the value of α should be small in order to reduce redundant packets. Then, we can note that when R_{corr}/R_{tran} is fixed, larger R_{tran} can lead to better coverage, because more sensors can overhear the reports. This also implies that a smaller value of α should be used when the network density is large. However, from Fig. 6.11(b), we observe that when the R_{tran} becomes larger, the redundancy is high even a small value of α is used. Thus, we suggest that R_{tran} should not be too large and we can increase the value of α to achieve required coverage.

Next, ℓ is investigated. In Fig. 6.12(a), we can see that prolonging the slot size can increase the success rate, because the hidden terminal problem is alleviated. Although the increment is small, the improvement will be large when the amount of packets transmitted becomes large. In addition, in Fig. 6.12(b), it can be seen that prolonging the slot size can also enhance coverage, because more packets

are transmitted to the sink successfully. However, in Fig. 6.12, we can see that prolonging the slot size will be penalized by long delay. Therefore, determining a proper value of ℓ may depend on application requirement.

When the network load is high, prolonging the slot size may not be a good idea, because long delay will worsen the buffer overflow problem. As we can see in Fig. 6.13, although the success rate becomes higher when ℓ becomes larger, the coverage becomes lower, because many packets are dropped due to buffer overflow. Thus, the network load also should be taken into consideration when we need to decide the proper value of ℓ .

6.4 Summary

We have shown how to exploit the spatial correlation of sensor data on the link layer for event-driven WSNs. A hybrid TDMA/CSMA protocol is proposed. The protocol has three features that makes it very efficient. First, the TDMA part is triggered only when sensors detect an event. By doing so, the protocol enjoys the benefits of collision-free transmission of TDMA and low latency transmission of CSMA. Second, the slot assignment strategy associated with the TDMA part takes the spatial correlation of sensor data into consideration. By intentionally allowing one-hop neighbors to share the same time slot, the number of slots required per frame is significantly reduced. Thus, the transmission latency is also reduced. Third, by enlarging the slot size on purpose, an interesting effect of pipeline transmission is formed, and thus the interference problem in the non-event area is alleviated. In addition, redundant reports are significantly reduced by our proposed report reduction scheme. We also discuss how to combine our scheme with LPL to achieve energy efficiency. Simulation results have demonstrated the efficiency of our scheme. We believe that our approach can be built on top of most CSMA-based MAC protocols.

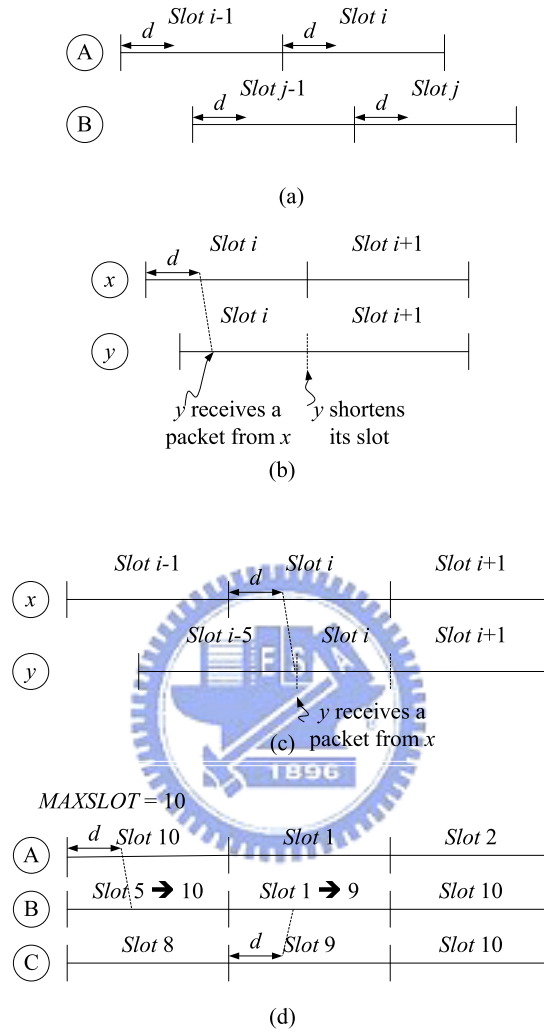


Figure 6.7: The synchronization problem.

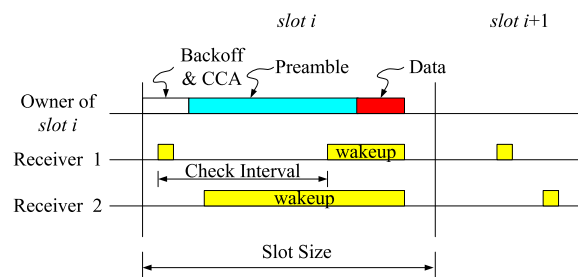


Figure 6.8: Combining our scheme with LPL.

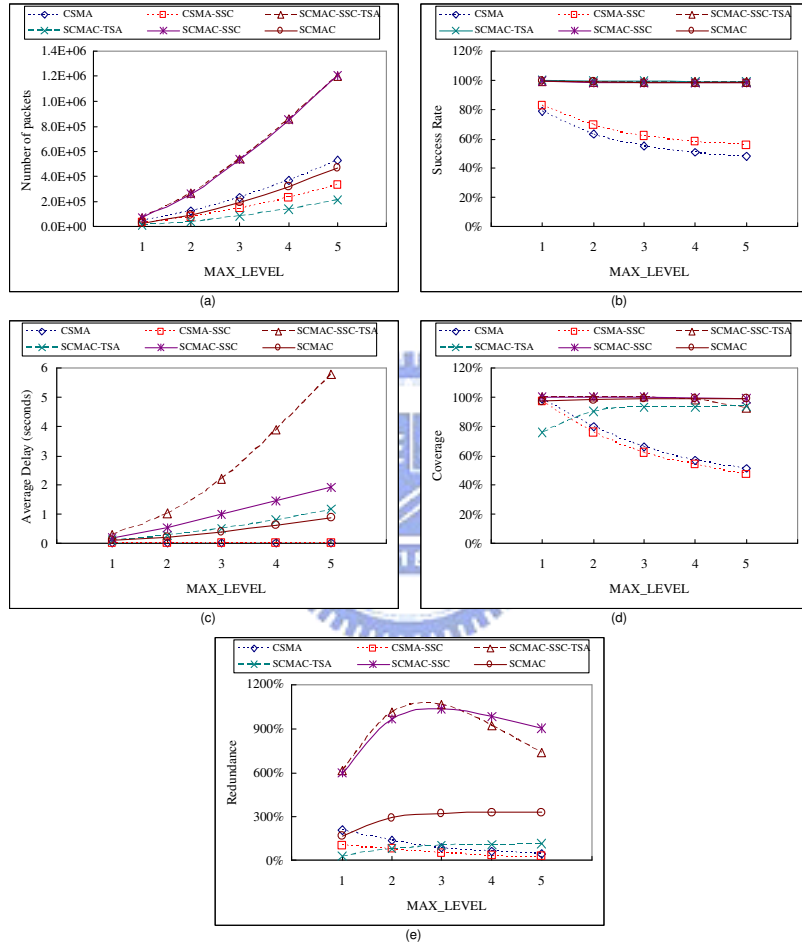


Figure 6.9: Comparison of different schemes, where $R_{corr} > R_{tran}$.

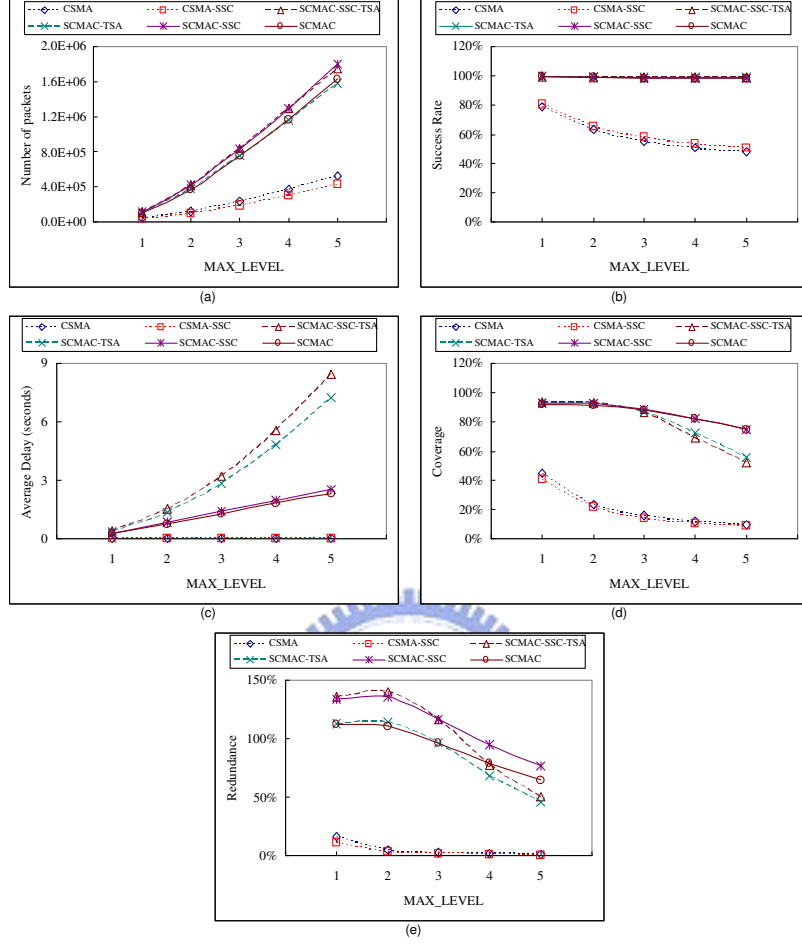


Figure 6.10: Comparison of different schemes, where $R_{corr} < R_{tran}$.

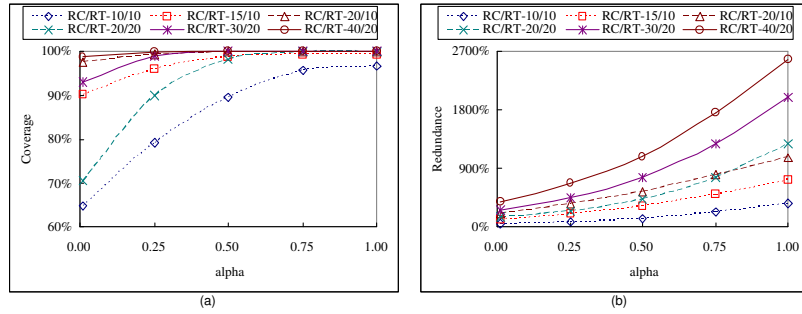


Figure 6.11: The impact of α under different ratios of R_{corr} to R_{tran} .

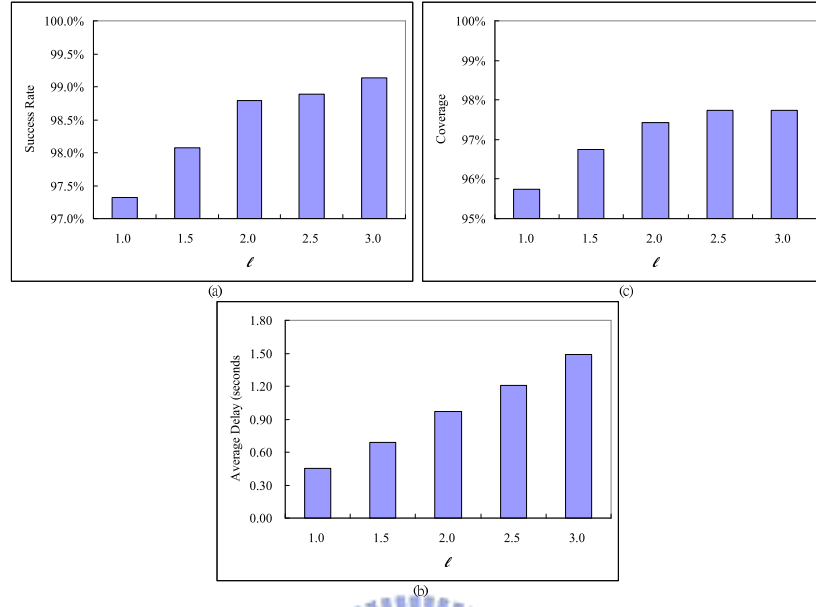


Figure 6.12: The impact of ℓ , where R_{corr} is 15 units, R_{tran} is 10 units, and the value of α is 0.3.

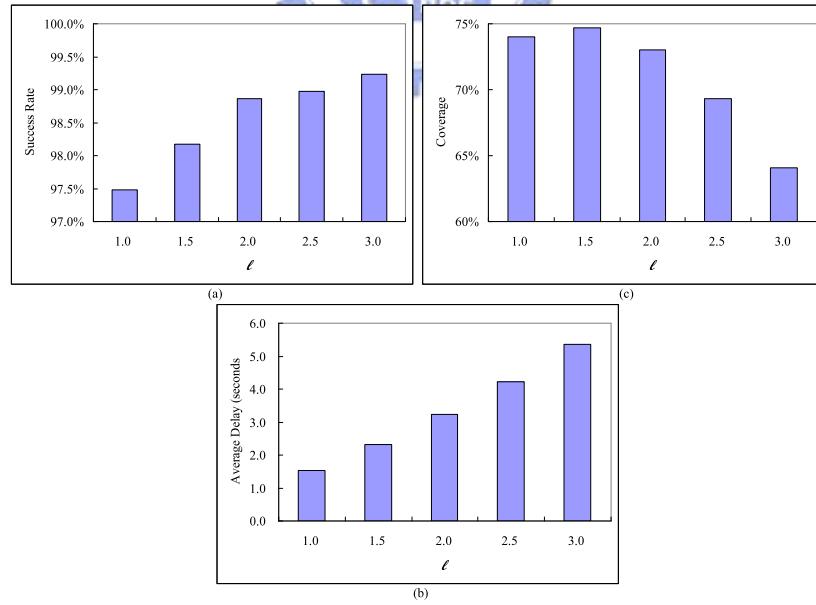


Figure 6.13: The impact of ℓ , where R_{corr} is 5 units, R_{tran} is 10 units, and the value of α is 1.0.

Chapter 7

Conclusions and Future Directions

Object tracking is an important application of WSNs and location management is one of the key steps involved in object tracking. In this dissertation, we propose several tree-based location management schemes to reduce the communication cost. We also address the contention and collision problems for event-driven WSNs (e.g., object tracking sensor networks). The significant results with future works are summarized as follows.

In Chapter 3, we have developed several efficient ways to construct a logical object tracking tree for a single-sink sensor network. We have shown how to organize sensor nodes as a logical tree so as to facilitate in-network data processing and to reduce the total communication cost incurred by object tracking. For the location update part, our work can be viewed as the extension of the work in [14], and we enhance the work by exploiting the physical structure of the sensor network and the concept of deviation avoidance. In addition, we also consider the query operation and formulate the query cost of an object tracking tree given the query rates of sensors. In particular, our approach tries to strike a balance between the update cost and query cost. Performance analyses are presented with respect to factors such as moving rates and query rates. Simulation results show that by exploiting the deviation-avoidance trees, algorithms DAT and Z-DAT are able to reduce the update cost. By adjusting the deviation-avoidance trees, algorithm QCR is able to significantly reduce the total cost when the aggregate query

rates is high, thus leading to efficient object tracking solutions.

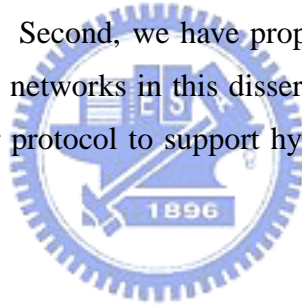
Chapter 4 further explores the possibility of having multiple sinks in the network. One advantage of having multiple sinks is to reduce the response time of queries. In addition, using multiple sinks can also relieve the traffic congestion problem associated with a single-sink system. We extend the single-sink location management scheme proposed in Chapter 3 by constructing multiple trees to support multi-sink WSNs. The corresponding update cost is formulated formally. Based on the formulation, we have presented two distributed algorithms to construct multiple trees. We have verified the benefits of a multi-sink WSN from different aspects, including the total (update plus query) cost, the number of sinks, query response time, query success rate, and load balance factor.

By exploiting the inherent property of imprecision of sensor data, Chapter 5 presents an imprecision-tolerant location management model for object tracking sensor networks. The proposed model consists of imprecision-tolerant update and query mechanisms that can be used to support imprecision-tolerant queries. By exploiting the feature of the tree-based location management schemes, the proposed model can provide multiple imprecision levels and ensure that the query cost will be proportional to the imprecision level. In addition, we develop a tree construction algorithm to facilitate the proposed location management model, which can reduce the query cost while minimize the increment of update cost.

By simulation, we observe that packet loss may make the location information incorrect in object tracking sensor networks. Thus, a protocol is proposed in Chapter 6 to support the location management schemes from the link layer. We have shown how to exploit the spatial correlation of sensor data on the link layer for event-driven WSNs. A hybrid TDMA/CSMA protocol is proposed. The protocol has three features that makes it very efficient. First, the TDMA part is triggered only when sensors detect an event. By doing so, the protocol enjoys the benefits of collision-free transmission of TDMA and low latency transmission of CSMA. Second, the slot assignment strategy associated with the TDMA part takes the spatial correlation of sensor data into consideration. By intentionally allow-

ing one-hop neighbors to share the same time slot, the number of slots required per frame is significantly reduced. Thus, the transmission latency is also reduced. Third, by enlarging the slot size on purpose, an interesting effect of pipeline transmission is formed, and thus the interference problem in the non-event area is alleviated. In addition, redundant reports are significantly reduced by our proposed report reduction scheme. We also discuss how to combine our scheme with LPL to achieve energy efficiency. Simulation results have demonstrated the efficiency of our scheme.

The future work includes two aspects. First, due to the fault-prone property of sensor nodes, developing a fault-tolerant mechanism for the tree-based location management is required. We will investigate a virtual-tree system, in which an implicit tree is still used. However, because no explicit tree exists, the fault-tolerant can be done easily. Second, we have proposed a link-layer protocol to support event-driven sensor networks in this dissertation. In the future, we will further develop a link-layer protocol to support hybrid time-driven/event-driven sensor networks.



Bibliography

- [1] G.-S. Ahn, E. Miluzzo, A. T. Campbell, S. G. Hong, and F. Cuomo. Funneling-mac: A localized, sink-oriented mac for boosting fidelity in sensor networks. In *SenSys '06: Proceedings of the 4th international conference on embedded networked sensor systems*. ACM Press, 2006.
- [2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.
- [3] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus. Tracking a moving object with binary sensors. In *Proc. of ACM SenSys*. ACM Press, Nov. 2003.
- [4] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, Sept. 1991.
- [5] S. Blackman and R. Popoli. *Design and Analysis of Modern Tracking Systems*. Artech House, 1999.
- [6] J. Burrell, T. Brooke, and R. Beckwith. Vineyard computing: sensor networks in agricultural production. *IEEE Pervasive Computing*, 3(1):38–45, 2004.
- [7] W.-P. Chen, J. C. Hou, and L. Sha. Dynamic clustering for acoustic target tracking in wireless sensor networks. In *Proc. of IEEE Int'l Conf. on Network Protocols (ICNP)*, Nov. 2003.

- [8] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1112 – 1127, Sept. 2004.
- [9] K.-W. Fan, S. Liu, and P. Sinha. On the potential of structure-free data aggregation in sensor networks. In *Proc. of IEEE Infocom*, 2006.
- [10] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *ACM SIG-MOBILE Mobile Computing and Communications Review*, 5(4):11–25, Oct. 2001.
- [11] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. An evaluation of multi-resolution storage for sensor networks. In *Proc. of ACM SenSys*. ACM Press, Nov. 2003.
- [12] 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.
- [13] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *ACM Int'l Conf. on Mobile Computing and Networking (MobiCom)*, Aug. 2000.
- [14] H. T. Kung and D. Vlah. Efficient location tracking using sensor networks. In *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*, Mar. 2003.
- [15] D. Li, K. Wong, Y. Hu, and A. Sayeed. Detection, classification, and tracking of targets. *IEEE Signal Processing Magazine*, 19(2):17–29, Mar. 2002.

- [16] C.-Y. Lin and Y.-C. Tseng. Structures for in-network moving object tracking in wireless sensor networks. In *IEEE Int'l Conf. on Broadband Networks (Broadnets)*, pages 718–727, Oct. 2004.
- [17] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. of ACM Int'l Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 88–97, Sept. 2002.
- [18] K. Mechitov, S. Sundresh, and Y. Kwon. Cooperative tracking with binary-detection sensor networks. *University of Illinois at Urbana-Champaign, Technical Report UIUCDCS-R-2003-2379*, 2003.
- [19] S. Pattem, B. Krishnmachari, and R. Govindan. The impact of spatial correlation on routing with compression in wireless sensor networks. In *Symposium on Information Processing in Sensor Networks (IPSN)*, 2004.
- [20] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on embedded networked sensor systems*, pages 95–107, Baltimore, MD, Nov. 2004.
- [21] S. S. Pradhan, J. Kusuma, and K. Ramchandran. Distributed compression in a dense microsensor network. *IEEE Signal Processing Magazine*, 19(2):51–60, Mar. 2002.
- [22] V. Rajendran, K. Obraczka, and J. Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. *Wireless Networks*, 12(1):63–78, Feb. 2006.
- [23] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A geographic hash table for data-centric storage. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, Atlanta, GA, Sept. 2002.

- [24] I. Rhee, A. Warrier, M. Aia, and J. Min. Z-mac: a hybrid mac for wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on embedded networked sensor systems*, pages 90–101, New York, NY, USA, 2005. ACM Press.
- [25] A. Scaglione and S. Servetto. On the interdependence of routing and data compression in multi-hop sensor networks. *ACM/Kluwer Mobile Networks and Applications (MONET) Journal of SPECIAL ISSUE on Mobility of Systems, Users, Data and Computing*, Sept. 2003.
- [26] E. Shih, S.-H. Cho, N. Ickes, R. Min, A. Shnha, A. Wang, and A. Chandrakasan. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In *ACM Int'l Conf. on Mobile Computing and Networking (MobiCom)*, pages 272–287, July 2001.
- [27] K. Stone and M. Colagrosso. Efficient duty cycling through prediction and sampling in wireless sensor networks. *Wireless Communications and Mobile Computing*, 2007.
- [28] X. Tang and J. Xu. Extending network lifetime for precision-constrained data aggregation in wireless sensor networks. In *Proc. of IEEE INFOCOM*, Apr. 2006.
- [29] T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *SenSys '03: Proceedings of the 1st international conference on embedded networked sensor systems*, pages 171–180, Los Angeles, CA, Nov. 2003. ACM Press.
- [30] L. van Hoesel and P. Havinga. A lightweight medium access protocol (LMAC) for wireless sensor networks. In *INSS04*, Tokyo, Japan, June 2004.
- [31] M. C. Vuran and I. F. Akyildiz. Spatial correlation-based collaborative medium access control in wireless sensor networks. *IEEE/ACM Transactions on Networking*, 14(2):316–329, 2006.

- [32] C.-Y. Wan, S. B. Eisenman, A. T. Campbell, and J. Crowcroft. Siphon: Overload traffic management using multi-radio virtual sinks. In *SenSys '05: Proceedings of the 3rd international conference on embedded networked sensor systems*. ACM Press, 2005.
- [33] J. Xu, X. Tang, and W.-C. Lee. EASE: An energy-efficient in-network storage scheme for object tracking in sensor networks. In *IEEE Sensor and Ad Hoc Communications and Networks (SECON)*, Sept. 2005.
- [34] Y. Xu and W.-C. Lee. On localized prediction for power efficient object tracking in sensor networks. In *Proc. of Int'l Workshop on Mobile Distributed Computing (MDC)*, May 2003.
- [35] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proc. of IEEE Infocom*, pages 1567–1576, June 2002.
- [36] S. Yoon and C. Shahabi. Exploiting spatial correlation towards an energy efficient clustered aggregation technique (CAG). In *International Conference on Communications (ICC)*, 2005.
- [37] W. Zhang and G. Cao. DCTC: Dynamic convoy tree-based collaboration for target tracking in sensor networks. *IEEE Transactions on Wireless Communication*, 3(5):1689–1701, Sept. 2004.