A resource-aware embedded commucation system for highly dynamic networks

Xunxing Diao

N^o d'ordre D.U : 2127
EDSPIC : 522

UNIVERSITE BLAISE PASCAL - CLERMONT II
ECOLE DOCTORALE
SCIENCES POUR L'INGENIEUR DE CLERMONT-FERRAND

THÈSE

Présentée Par

Xunxing DIAO
Master en Informatique

Pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ
Spécialité : INFORMATIQUE

A Resource-aware Embedded Communication System for Highly Dynamic Networks

---

Soutenue publiquement le 27 Mai 2011 devant le jury :

M. Alain QUILLIOT	*Président*
M. Bernard TOURANCHEAU	*Rapporteur*
Mme. Houda LABIOD	*Rapporteur*
M. Haiying ZHOU	*Examinateur*
M. Jean-Pierre CHANET	*Invité*
M. Kun-Mean HOU	*Directeur de thèse*
Mme. Jian-Jin LI	*Directeur de thèse*

Résumé

Chaque année en Europe, 1.300.000 accidents de la route ont comme conséquence 1.700.000 blessés. Le coût financier d'accidents de la route est évalué à 160 milliards d'euros (approximativement le même coût aux Etats-Unis). VANET (Vehicular Ad-hoc NETwork) est une des technologies clés qui peut permettre de réduire d'une façon significative le nombre d'accidents de la route (e.g. message d'urgence signalant la présence d'un obstacle ou d'un véhicule en cas de brouillard). En plus de l'amélioration de la sécurité et du confort des conducteurs et des passagers, VANET peut contribuer à beaucoup d'applications potentielles telles que la prévision et la détection d'embouteillages, la gestion d'infrastructure de système de transport urbain (e.g. système de transport intelligent multimodal) etc.

Dans cette thèse, je présenterai un système embarqué dédié à la communication inter-véhicule particulièrement pour les applications sécuritaires de passagers et de conducteurs. Nos efforts de recherche et de développement sont centrés sur deux principaux objectifs : minimiser le temps de latence intra-nœud et le délai de communication inter-véhicule en prenant en compte le changement dynamique du VANET. De ce fait pour atteindre ces objectifs, des nouvelles approches (e.g. inter-couche 'Cross-layering') ont été explorées pour respecter les contraintes de ressource (QoS, mémoire, CPU et énergie de la communication inter-véhicule) d'un système embarqué à faible coût.

Le système de communication embarqué proposé comporte deux composants logiciels principaux : un protocole de communication dénommé CIVIC (Communication Inter Véhicule Intelligente et Coopérative) et un système d'exploitation temps réel appelé HEROS (Hybrid Event-driven and Real-time multitasking Operating System). CIVIC est un protocole de communication géographique à faible consommation énergétique et à faible temps de latence (délai de communication). HEROS gère contextuellement l'ensemble du système (matériel et logiciel) en minimisant le temps de latence et la consommation des ressources (CPU et mémoire). En outre, le protocole de communication CIVIC est équipé d'un système de localisation LCD-GPS (Low Cost Differential GPS). Pour tester et valider les différentes techniques et théories, la plateforme matérielle LiveNode (LImos Versatile Embedded wireless sensor NODE) a été utilisée. En effet, la plateforme LiveNode permet de développer et de prototyper rapidement des applications dans différents domaines. Le protocole de communication CIVIC est basé sur la technique de 'broadcast' à un saut ; de ce fait il est indépendant de la spécificité du réseau. Pour les expérimentations, seule la norme d'IEEE 802.15.4 (ZigBee) a été choisie comme médium d'accès sans fil. Il est à noter que le médium d'accès sans fil ZigBee a été adopté comme le médium standard pour les réseaux de capteurs sans fil (RCSFs) et le standard 6LoWPAN ; car il est peu coûteux et peu gourmand en énergie.

Bien que le protocole de communication à l'origine soit conçu pour répondre aux exigences

de VANET, ses domaines d'application ne sont pas limités à VANET. Par exemple il a été utilisé dans différents projets tels que MOBI+ (système de transport urbain intelligent) et NeT-ADDED (projet européen FP6 : agriculture de précision). Les VANETs et les RCSFs sont les réseaux fortement dynamiques, mais les causes de changement topologique de réseau sont différentes: dans le réseau VANET, il est dû à la mobilité des véhicules, et dans le RCSF, il est dû aux pannes des nœuds sans fil. Il est à noter que le VANET et le RCSF sont généralement considérés comme un sous-ensemble du réseau MANET (réseau ad-hoc mobile). Cependant, ils sont réellement tout à fait différents du MANET classique, et leurs similitudes et différences seront expliquées en détail dans la thèse.

La contribution principale de mes travaux est le protocole CIVIC, qui échange des messages en basant sur l'information géographique des nœuds (position). Les travaux relatifs de la thèse se concentreront sur les techniques, les problèmes et les solutions de routage géographique, mais d'autres techniques de routage seront également adressées. Quelques projets relatifs au protocole de communication ont été étudiés mais leur implémentation et les aspects d'expérimentation n'ont pas été détaillés. Enfin la thèse ne présente pas simplement les techniques et concepts adoptés, et les résultats de simulation, mais en outre, elle expliquera les aspects techniques importants pour la réalisation et l'expérimentation des différentes applications ainsi que les résultats concrets obtenus.

Abstract

Each year in Europe, 1,300,000 vehicle accidents result in 1,700,000 personal injuries. The financial cost of vehicle accidents is evaluated at 160 billion Euros (approximately the same cost in the USA). VANET (Vehicular Ad-Hoc NETwork) is a key technology that can enable hazard alarming applications to reduce the accident number. In addition to improve the safety for drivers and passengers, VANET can contribute to many potential applications such as detecting and predicting traffic jams, auto-optimizing the traffic flow, and helping disabled passengers to access public transports.

This thesis will present an embedded communication system dedicated to VANET especially for the safety-related applications. Our design mainly tries to achieve two requirements: as one can imagine, the embedded communication system for VANET requires extra effort *to deal with the highly dynamic network topology* caused by moving vehicles, thus to shorten the intra-node system latency and inter-node network delay is essential requirement for such embedded communication system. Besides, a fundamental requirement for any practical embedded system is *resource-awareness*. Although the embedded communication system on vehicles may gain better hardware supports, the characteristics of embedded hardware still have to cope with resource constraints in terms of QoS, memory, CPU and energy.

The embedded communication system involves two major software components: a routing protocol called CIVIC (Communication Inter Véhicule Intelligente et Coopérative) and an embedded operating system called HEROS (Hybrid Event-driven and Real-time multitasking Operating System). The former is a quick reaction and low resource consumption geographic protocol for inter-vehicle message transmissions; and the latter controls the whole system and assures intra-node resource awareness. In addition, the system can use a localization software solution called LCD-GPS (Low Cost Differential GPS) to improve the accuracy of locations. The hardware platform is LiveNode (LImos Versatile Embedded wireless sensor NODE), which is a versatile wireless sensor node enabling to implement rapidly a prototype for different application domains. The communication system is based on the one-hop broadcast, thus it does not have a strict limitation on network specification. For the experiments only, the IEEE 802.15.4 standard is chosen as the underlying wireless access medium. The standard is well known as a low-power consumption standard requiring low-cost devices. Notice that the IEEE 802.15.4 standard is also the wireless access medium of 6LoWPAN.

Although the embedded communication system is originally designed to meet the requirements of VANET, but its application domains are not limited to VANET. For example, another network which can use the embedded communication system is WSN (Wireless Sensor Network). CIVIC was used to implement different real-world projects such MOBI+ (intelligent urban transportation system) and EU-FP6 NeT-ADDED (precision agriculture). Both VANET

and WSN are highly dynamic networks, but the causes of changing network topology are different: the former is because of the high-mobility feature of vehicles, and the latter is because of the fault of wireless sensors. Note that, although VANET and WSN are both commonly considered as the subset of MANET (Mobile Ad-hoc NETwork), they are actually quite different from the classical MANET, and the similarities and differences will be further explained in the thesis.

The major contribution of my works relates to the CIVIC protocol, which routes messages based on the geographic information. The related works of the thesis will focus on the geographic routing techniques, problems and solutions, but other related techniques will also be addressed. Note that, although some related projects were investigated but their implementation and experiment aspects were not detailed. Finally, the thesis will not only introduce the system design and provide simulation results, but also explain some of the important implementation issues, give the theoretical evaluation results and provide the real-world experiment results.

Acknowledgement

Throughout my years as a PhD student, many people have kindly provided me with their help. I have to thank them all and I am particularly indebted to the following people.

First and foremost, I would like to express my sincere gratitude to my advisors, Prof. Kun-Mean HOU and Dr. Jian-Jin LI, for their guidance, support and patience during my PhD study period in the LIMOS laboratory, Blaise Pascal University. I am very fortunate to have them as my advisors.

I am very grateful to the members of the dissertation jury for their valuable time and feedback. My special appreciation goes to Prof. Bernard TOURANCHEAU and Dr. Houda LABIOD.

I would also like to thank all colleagues and friends for their contribution on this dissertation. Here I would mention Mr. Hongling SHI, Dr. Messaoud KARA and Dr. Jing WU.

Finally, a special thanks to my family. Without their understanding and encouragement I could not have begun my graduate study and it would be much more difficult for me to finish my PhD degree.

Contents

List of Figures

List of Tables

List of Acronyms

6LoWPAN ipv6 over LOw power Wireless Personal Area Networks.

AODV Ad hoc On-Demand Distance Vector.

ARM Advanced Risc Machine.

BE Backoff Exponent.

BLR Beacon-less Routing.

CCA Clear Channel Assessment.

CIVIC Communication Inter Véhicule Intelligente et Coopérative.

CPI Cycles Per Instruction.

CR Compass Routing.

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance.

CTS Clear To Send.

DANKAB Directional Area Neighbour Knowledge Adaptive Broadcast.

DG Distance-based Greedy.

DLL Data Link Layer.

DREAM Distance Routing Effect Algorithm for Mobility.

DSR Dynamic Source Routing.

DSRC Dedicated Short-Range Communication.

DSSS Direct Sequence Spread Spectrum.

DTN Delay Tolerant Network.

DV Distance-Vector.

EOS Embedded Operating System.

GeOpps GEographical OPPortuniStic Routing.

GFG Greedy-Forward-Greedy.

GG Gabriel Graph.

GHLS Geographic Hashing Location Service.

GHT Geographic Hash Table.

GLS Grid's Localization Service.

GPS Global Positioning System.

GPSR Greedy Perimeter Stateless Routing.

GSM Global System for Mobile Communications.

HEROS Hybrid Event-driven and Real-time multitasking Operating System.

IFS Inter-Frame Space.

IGF Implicit Geographic Forwarding.

IPC Inter-Process Communication.

IVC Inter-Vehicle Communication.

LAR Location-Aided Routing.

LCD-GPS Low Cost Differential GPS.

LER Last Encounter Routing.

LiveNode LImos Versatile Embedded wireless sensor NODE.

LS Link-State.

MAC Media Access Control.

MANET Mobile Ad-hoc NETwork.

MCU MicroController Unit.

MFR Most Forward progress within Radius.

MMRS Multi-support, Multi-service Routers and Servers.

MPDU Mac Protocol Data Unit.

MSDU Mac Service Data Unit.

NB Number of Backoff periods.

NFP Nearest with Forward Progress.

PAN Personal Area Network.

PHY PHYsical layer.

PPDU Phy Protocol Data Unit.

PSDU Phy Service Data Unit.

QoS Quality of Service.

RF Radio Frequency.

RNG Relative Neighborhood Graph.

RTS Request To Send.

SDREAM Super-small Distributed REAl-time Microkernel.

SF Simple Flooding.

SOS Sensor Operating System.

SPI Serial Peripheral Interface.

TS Tuple Space.

UART Universal Asynchronous Receiver/Transmitter.

USART Universal Synchronous/Asynchronous Receiver/Transmitter.

UTRA-TDD Umts Terrestrial Radio Access Time Division Duplex.

VADD Vehicle-Assisted Data Delivery.

VANET Vehicular Ad-Hoc NETwork.

VII Vehicle Infrastructure Integration.

Wi-Fi WIreless FIdelity.

WSN Wireless Sensor Network.

WSNOS Wireless Sensor Network Operating System.

ZigBee Zonal Intercommunication Global-standard, where Battery life was long, which was Economical to deploy, and which exhibited Efficient use of resources.

Chapter 1

Introduction

1.1 System Overview

When talking about an embedded system, it normally refers to the tiny computer system designed to cope with stringent resource constraints and perform several dedicated functions. The hardware and software designs of embedded systems are very different from the ones of general-purpose personal computers. The major cause is from the limited hardware characteristics of the embedded system in terms of energy, CPU and memory. Moreover, an embedded communication system will have to accomplish more than that if the network is purely supported by the embedded hardware. Besides the previous hardware limitations, the network limitations such as bandwidth and transmission distance will have to be taken into account. Therefore, the *resource-awareness* is always the basic requirements for any practical embedded communication system.

The traditional embedded communication systems are usually only needed to handle static and lowly dynamic networks, but with the increase of vehicular safety applications such as hazard alarming and cooperative driving, a new requirement to solve *highly dynamical networks* has started to emerge for the embedded communication systems. The highly dynamical network in the thesis refers to the network with a frequently changed network topology.

The highly dynamical network in the thesis mainly refer to *VANET (Vehicular Ad-Hoc NETwork)*, which uses moving vehicles as network nodes to create a *MANET (Mobile Ad-hoc NETwork)*. Besides, another type of highly dynamical network is *WSN (Wireless Sensor Network)*. However, the causes of their network topology changes are different: the former is caused by the high-mobility feature of vehicles, and the latter is caused by the fault of wireless sensors. Note that, the WSN in the thesis is purposed for used in the sub-domains including smart home and greenhouse monitoring, thus it can be expected to have higher hardware capability (e.g. equipped with *GPS (Global Positioning System)*).

The embedded communication system involves two major software components: a geographic routing protocol called *CIVIC (Communication Inter Véhicule Intelligente et Coopérative)* and an embedded operating system called *HEROS (Hybrid Event-driven and Real-time multitasking Operating System)*. The former is a quick reaction and low resource consumption geographic routing protocol for inter-vehicle message transmissions; and the latter controls the whole system and assures intra-node resource awareness.

Because the embedded communication system is designed for the practical use in a near

future, some trade-offs in the software designs have been made to solve the highly dynamic problems with the resource constrains including the embedded hardware and network. Our software design rationale is to provide a relatively optimal software result with the minimum resource, instead of providing the most optimal software result with a non-practical computation amount under the resource constrains. The examples of the software design rationale can be found from the routing computation results in CIVIC protocol and the component flexibility in the HEROS operating system.

The hardware component of the embedded communication system is *LiveNode (LImos Versatile Embedded wireless sensor NODE)*, which is a versatile wireless sensor node enabling to implement rapidly a prototype for different application domains. The current network component for the experiments is the *IEEE 802.15.4 standard*. The embedded communication system does not have a strict limitation on network specification, and actually one of the designs of the embedded communication system is to try to makes it adaptive to multiple wireless network mediums. In addition, the system can use a localization software solution called *LCD-GPS (Low Cost Differential GPS)* to improve the accuracy of locations.

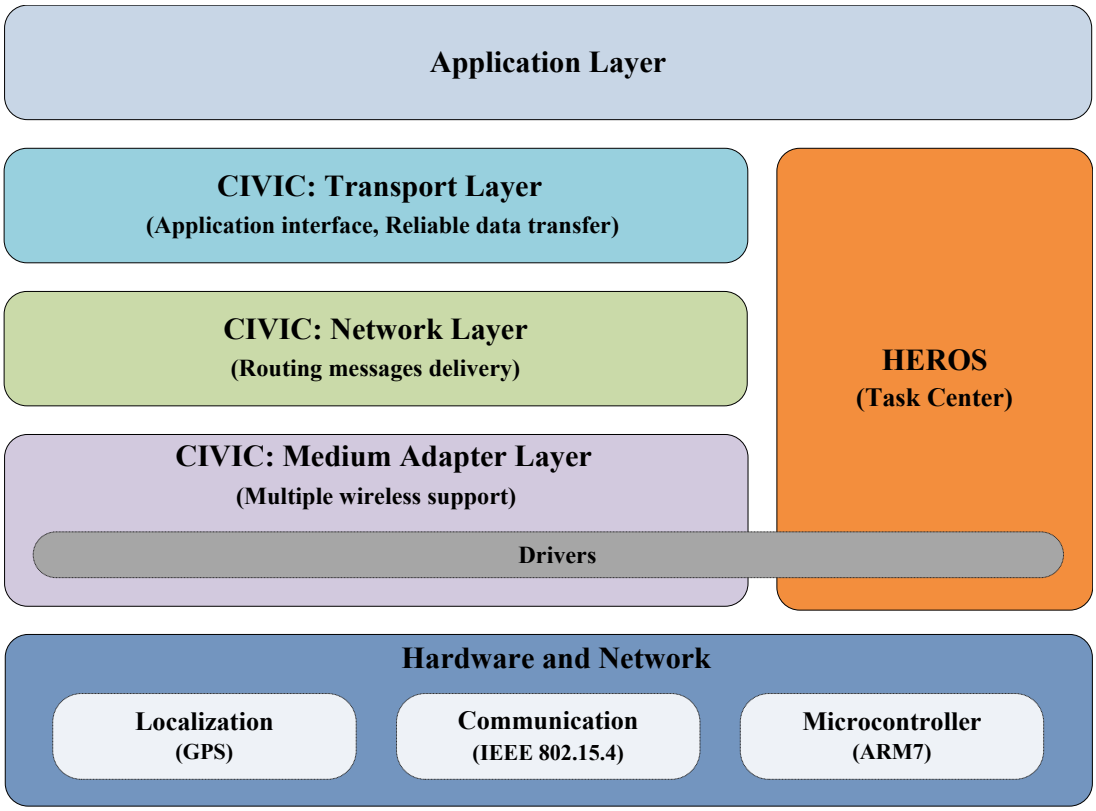


Figure 1.1: Stack Architecture of the Communication System

The stack architecture of these components is shown in Figure 3.1 and they will be explained in details in the later chapters.

The main focus of the thesis is the routing aspects on the software and networking side, which are mainly relating to the CIVIC protocol, but before presenting CIVIC protocol and the other aspects of the communication system, the following section will first introduce our targeting networks: VANET, WSN and MANET, along with our motivation. As the full name of CIVIC suggesting, the VANET will be explained in more details. Although VANET and

WSN are both commonly considered as the subset of MANET, they are actually quite different from the general-purpose MANET, and the similarities and differences will be further explained the MANET section.

1.2 Network Overview

1.2.1 VANET

The number of vehicles on roads was continually increasing for the recent years in Europe, but the improvement on road conditions and the training to drivers were not followed up enough. Each year, vehicle accidents result in 1,700,000 personal injuries, and the financial cost of vehicle accidents is evaluated at 160 billion Euros (approximately the same cost in the USA). VANET is considered to be one of the key technologies that can enable hazard alarming applications to reduce of the accident numbers. In addition to improve the safety for drivers and passengers, VANET can contribute to many potential applications such as detecting and predicting traffic jams, auto-optimizing the traffic flow, and helping disabled passengers to access public transports.

Comparing with general-purpose *MANET* communications, the *IVC (Inter-Vehicle Communication)* for VANETs has its unique features that have not been fully explored.

- As one can imagine, it requires extra effort to deal with real-time event and network delay under the highly dynamic topology caused by highly mobile vehicles.
- The network size could be very large in big cities, thus it requires the ability for the localized configuration. The traditional client/server systems are not appropriate.
- The density of vehicular network are much more variable, thus there is a requirement for a new routing protocol to minimize the administrative overhead.
- The distribution of vehicular network is generally along roads. It provides the opportunity for IVC to deploy the roadside infrastructure for supporting network access and *QoS (Quality of Service)*, but as well, it raises the expectation for the adaptability to different wireless standards.
- The position and direction of network nodes could be obtained by *GPS* on vehicles.
- Last but not least, the hardware system must be in a low price to enable the broader adaptability.

These unique IVC features have been considered for our communication system. All the major factors that cause the message delay and loss rate of an embedded communication system in our design have been evaluated carefully. The CIVIC protocol is a geographic routing protocol including the configurable proactive and reactive approaches to make it suitable for different VANETs. The possibility of using infrastructure supports is also taken into account. The communication system is based on the one-hop broadcast, thus it does not have a strict limitation on network specification.

The resource-awareness is not only in the design of the CIVIC protocol. In our embedded communication system, CIVIC constitutes a quick-response inter-node communication stack on HEROS, while HEROS provides CIVIC with intra-node mechanisms to run hybrid tasks and manage hardware. The designs have been implemented in a low-price LiveNode sensor board. The network for our simulations and experiments are IEEE 802.15.4 standard, which is well known as a low-power consumption standard. Notice that IEEE 802.15.4 standard is also adopted by *6LoWPAN (ipv6 over LOw power Wireless Personal Area Networks)*.

1.2.2 WSN

WSNs are usually grouped by large numbers of low-cost sensor nodes with one or more sinks. A sink with high hardware capabilities holds the connection between sensor nodes and data proxy. The hardware characteristics of sensor nodes have to cope with more stringent resource constraints in terms of CPU, energy, memory, bandwidth and transmission distance.

The deployment of WSNs is hard to predict. In some applications, sensor nodes are randomly deployed by aircraft. Faults in sensor nodes are a common fact because of resource constraints and unanticipated environment variations. Owning to the unique features, auto-configuration in WSN is particularly important. It is impractical to manually initialize or reconfigure hundreds of low-cost sensor nodes. In some applications such as disaster monitoring and battlefield surveillance, the sensor nodes are often required to operate in dangerous environment where accessibility is highly restricted.

An auto-clustering algorithm is purposed in the lightweight management protocol of our embedded communication system. The algorithm autonomously divides sensor nodes into a set of single-level clusters by using only locally-available information.

1.2.3 MANET

MANET is a structureless network of mobile nodes. Nodes in general-purpose MANETs are usually battery-operated, which makes energy efficiency one of important requirements, but it is not as important as in WSNs. The movement pattern of MANET is different from the one in VANET. The nodes in MANETs are considered to be randomly moving instead of moving along the roads as in VANETs. The common feature of them is self-configuring. Table 1.1 shows the differences of three types of networks.

	VANET	WSN	MANET
Network size	Very Large	Large	Medium
Node's general capability	Medium or High	Very Low	Medium
Node's faults	Medium	Very Frequently	Possible
Node's mobility	Very High	Static or Low	Medium
Mobile pattern	Along the roads	Normally Static	Randomly
Location usage	Required	Not Required	Possible

Table 1.1: Comparisons properties among MANET, VANET and WSN

1.3 State Contributions

My major contribution and the main focus of the thesis is in the software and networking sides of the embedded communication system. Mainly, it is about the CIVIC routing protocol, along with the works in the HEROS operating system.

The related works of the thesis will present the state of art of routing technique researches: the techniques, their problems and some solutions. The focus is put on the geographic routing techniques, and there are two reasons for that: firstly, the main researches for solving the lowly/highly dynamic routing problems are based on the geographic techniques; secondly, the CIVIC protocol, which is a major software component of our embedded communication system, is designed to be a geographic protocol. Note that, as for using in highly dynamic networks under the resource constrains, none of the current routing techniques can be considered to be exactly adaptable and reliable. Most of the solutions are not very well suitable for current embedded hardware capabilities (e.g. CPU and memory) and/or the features of the highly dynamic networks like VANETs (e.g. without considering the context-awareness). However, these researches could be used for the further development of our embedded communication system.

As previously mentioned, the embedded communication system is developed for real-world applications. The thesis will present all the aspects needed by developing such an embedded communication system including design concepts, algorithms, evaluations, simulations, implementations and real-world experiments. Although some relating projects were investigated but the implementation and experiment aspects were not detailed.

The final results from simulations and real-world experiments contained in the thesis do not completely prove our original thoughts for using the system designs in highly dynamic networks when considering high-density network nodes. Further improvements are required in the future. However, since the isolated elements in our design have been carefully evaluated and they are individually proved to be the suitable ones, it is greatly possible that the embedded communication system can be actually adapted to real-world applications with the further improvements. Moreover, as for the aspects of message delay, system latency and memory consumption, the design is proved to be a very good one.

The CIVIC protocol and the HEROS operating system have been implemented as a prototype or conceptually adapted in different types of dynamic real-world projects such MobiPlus (intelligent urban transportation system), *EU-FP6 NeT-ADDED* (precision agriculture) and LiveCare (telemedicine).

1.4 Structure of Thesis

The remainder of the thesis is organized as follows:

- The next chapter, *Related Works on Geographic Routing Protocols*, presents the current routing techniques.
- After that, it is a chapter, named *Communication System: Designs and Evaluations*, for the detailed description of design concepts and routing algorithms. In the same chapter, theoretical evaluations (with a fog experiment at LRPC test center) and the simulations on *Shawn* will also be given.
- Then, there are two chapters are about the our specific applications, named *Applications: Inter-vehicle Communication* and *Applications: WSN Precision Agriculture*. The implementations and real-world experiments will be detailed in the first one, and the MATLAB simulation results will be presented in the second one.
- The last chapter is *Conclusions and Ongoing Works*.

Chapter 2

Related Works on Geographic Routing Protocols

This chapter presents the routing techniques relating to our embedded communication system. The focus in this chapter is the geographic routing and its application for highly dynamic networks like VANETs, but other related general routing techniques will also be addressed. This chapter is organized as follow:

In the beginning, Section 2.1 gives an overview of the related routing techniques, problems and solutions. Because the embedded communication system can also be adapted to WSNs, which is also a type of high dynamic network, Section 2.1.2 briefs the routing requirement for WSNs. The network layer of the CIVIC protocol uses the geographic routing, which is starting to become a common routing technique for VANETs. Section 2.2, Section 2.3, Section 2.4 and Section 2.5 explains in details about the geographic routing for VANETs. Section 2.6 introduces the VANET projects using geographic routing.

Moreover, this chapter tries to summarize the features of these routing techniques, and compare the routing techniques only based on the summarized features. This research field on highly dynamic networks like VANETs is relatively new, thus few routing techniques provide full required functions. In the latter section, the simulations for the key features will be given.

2.1 Overview

The *routing* in MANETs is the process for nodes themselves to discover the path for sending network traffic. A *routing protocol* operates at the network layer of OSI reference model, and it implements the specifications and mechanisms (e.g. message format and routing algorithm) used for the auto-configured communications between nodes. A *routing algorithm* is for comparing the potential routing paths and providing a routing decision. A network topology in routing algorithms is often described as a weighted graph with metrics representing nodes and edges representing wireless links. The metrics that can be used to calculate the edge weight in MANETs include hop count, remaining energy, delay, received signal strength, bandwidth, load balance, reliability, etc.

The result of the routing algorithm is a decision of routing path (route). Mostly, a *single-path* routing is used. In the cases of more than one routing paths, it is called a *multi-path* routing. The multi-path can also be acquired by the flooding-based techniques. Moreover, there are

two types of routing paths depending on the configuration methods: static and dynamic. A *static routing path* is manually stored by the network administrator to nodes within a one-hop administrative distance. The technique is not practical for dynamical or large networks, thus the following description will not include it. A *dynamic routing path* is discovered and maintained by the nodes themselves through routing protocols. The *route discovery* normally includes two processes: a routing request and a route reply. The traditional routing request is implemented by a flooding from the source node, and the routing reply is more often to be an unicast directed from the destination node.

Routing protocols described in the thesis can be divided into three broad categories: *topological*, *hierarchical* (*clustered-based*) and *geographic* (or *position-based*) routing protocols. The topological routing protocols use the metrics in a weighted graph that mainly relates to the link status. The hierarchical routing protocols organize nodes into clusters, thus inter-cluster and intra-cluster can use different routing techniques. The geographic routing protocols consider the physical position of nodes (or regions) as the principle routing parameter. A large number of routing protocols adopt more than one type of routing techniques, but if possible, the thesis does not consider the hybrid routing protocols as a standalone category. If a routing protocol combines different routing techniques, its major feature determines which of the three categories it belongs to.

2.1.1 Topological Routing

The topological routing is not the focus of this thesis, but some routing techniques that first appear at topological routing protocols are commonly used by geographic and hierarchical routing protocols. This section will introduce these common routing techniques. Some routing protocols are explained in more details if they are typical ones, or they are used by the geographic or hierarchical routing protocols introduced in the following sections.

There is one important reason why these classical topological routing techniques cannot be adopted to highly dynamical MANETs independently: the infrastructure of topological routing is more often based on the result of connections in a network topology (i.e. route discovery, maintenance and deletion), thus at least one end-to-end connection (i.e. a routing path) must be found before the data delivery. In the other words, these topological routing mainly work in the connection-oriented principle: A basic transmission technique for the electronic data networks including MANETs is *packet switching*, which is in either *connection-oriented* or *connectionless* principle. The packet transmission in a connection-oriented principle is prefixed with a connection setup stage, thus this mode is more reliable in static and low-mobility networks. On the other hand, a packet in a connectionless principle is routed individually by a network node only depending on the packet header information. Although a connectionless principle may require additional header information, it can normally achieve a small delay during a transmission, thus this mode has a bigger advantage in a highly dynamical MANETs.

2.1.1.1 Link-state or Distance-vector Strategy

The topological routing normally selects a next-hop forwarder by the *shortest-path (or minimum-weight path)* strategy. Depending on principle algorithms, the strategy can be categorized into two categories: *LS (Link-State)* and *DV (Distance-Vector)*. Both classical LS

and DV routing protocols have disadvantages for the highly dynamical MANETs with a low resource consumption requirement.

- In *LS routing*, nodes are informed about the neighbor links of an entire network. The information updates is more often implemented by periodic flooding. Since a network topology with all kinks is available for each node, a node can utilize the *Dijkstra's algorithm* as the principle algorithm independently; therefore the classical LS routing protocols in an ideal scenario can provide an accurate result faster than DV routing protocols. However, in a practical MANET, such centralized methods are easy to generate a higher routing control overhead, and they normally have a slower reaction to the outdated routing paths.
- *DV routing* decentralizes the shortest path strategy by restricting the vision of nodes within neighbors. The routing information is gradually spread via neighbor broadcasts. The principle algorithm for classical DV routing protocols is the *Bellman-Ford algorithm*. Instead of requiring the information of an entire network, the algorithm advertises the link information as a list of vectors with distance (i.e. the total edge weights to a destination) and direction (i.e. a next-hop forwarder). Comparing with the Dijkstra's algorithm, it requests less memory to store the routing information, it is easier to be implemented, and it can detect the existence of negative cycles. DV routing protocols are more practical for regular MANETs, but because the nodes themselves control the updates of routing information, and the vision of nodes is limited to neighbors, the classical DV routing protocols suffer from slower routing convergence, *routing loop* problem and *count-to-infinity* problem.

2.1.1.2 Proactive, Reactive, or Hybrid Scheduling

The topological routing protocols can be divided into proactive, reactive and hybrid categories based on the *route discovery scheduling*:

- A *proactive scheduling* maintains up-to-date routing tables for partial or entire network. In order to have correct routing paths, each node needs to explore network routing periodically. The approach keeps the end-to-end delay low because data can be sent to a destination node without an immediate routing request. However, this scheduling technique needs to maintain the routing paths even they are not currently used, thus it may not be suitable for the MANET nodes with low memory resource. Moreover, to maintain correct paths only by proactive scheduling increases the routing overhead significantly in highly dynamical MANETs. The typical proactive examples are the LS-based protocols OLSR [1] and TBRPF [2], and the DV-based protocol DSDV [3]. The *Destination-Sequenced Distance-Vector* (DSDV) routing protocol uses classical flooding-based Bellman-Ford algorithm, but it avoids the looping-related problems by adding a destination sequence number to each entry in a routing table.
- The *reactive scheduling* does not maintain a full routing table at anytime. They discover routing paths only when a demand is received. The scheduling technique is more efficient in terms of memory utilization but along with additional end-to-end delay. Moreover, the technique may still generate a high routing overhead in highly dynamical MANETs.

The typical examples are AODV [4] [5], DSR [6], TORA [7], and SSA [8]. The *Ad hoc On-Demand Distance Vector (AODV)* routing protocol is based on DSDV but with a reactive routing discovery. To avoid redundancy in the flooding-based routing request, a node in AODV only forwards the same routing request once. The *Dynamic Source Routing* (DSR) protocol is similar to AODV in the flooding-based process, but it is a LS-based protocol. Moreover, instead of caching the routing paths in the passing nodes (routers), the routing paths in DSR are contained directly in the messages for the routing request and reply. The source node can select and then add the best routing path to the data delivery packets. Because the source node specifies the routing path but not the routers, this process is called as *source routing*. The *Temporally-Ordered Routing Algorithm* (TORA) is neither DV nor LS, but a link-reversal algorithm, which builds and maintains a *Directed Acyclic Graph (DAG)* tree which is rooted from a destination node to all its source nodes. When a link fails, a related node traces back to the source nodes by the DAG tree, and rebuild only a limited part of the tree. The *Signal Stability-based Adaptive* (SSA or SSR) routing protocol builds and maintains both a routing table and a signal stability (SST) table for each node. The SST classifies the states of neighbor connections based on the signal strength of periodic beacon messages.

- The *hybrid scheduling* combines the advantages of the proactive and reactive scheduling techniques. It normally divides nodes in sub-networks, and then adopts different configuration approaches depending on regions. An typical example is *Zone Routing Protocol* (ZRP) [9], which includes a proactive intra-zone routing scheduling, and a reactive inter-zone routing scheduling. The hybrid scheduling technique can reduce routing overhead and increase the scalability for static network, but the network division could create an additionally high overhead if the network topology is changed too frequently.

2.1.1.3 Efficiency-based and Stability-based Purposes

Depending on purposes, there are two subsets of shortest-path routing protocols for MANETs. The classical subset mainly considers the *efficiency* of a routing path, and protocols in this subset measure routing paths by the end-to-end metrics such as hop count or delay (DSDV [3], AODV [4] [5], and DSR [6]).

Another newer subset focuses on the *link stability*, it tries to extend the network lifetime and throughput, and they normally use the metrics such as signal strength (SSA [8], ABR [10]), route lifetime (RABR [11]), and link reversal (TORA [7], LMR [12]). These two subsets do not really have an opposing nature. The stability-based routing techniques can be an enhancement to the efficiency-based ones (BSR [13]).

2.1.2 Hierarchical Routing

Usually, a node in a hierarchical network has two basic roles: cluster head and cluster member; and an additional role: cluster gateway. The role of a cluster head or master represents for routing, management and aggregation. A cluster gateway can belong to more than one cluster, and this role normally only represents for routing data between clusters. Besides, all other nodes are all in the role of cluster members. The challenge of hierarchical routing techniques is the

election of cluster head and cluster gateways. These two types of nodes are easy to become the bottleneck in data transmission, because they afford the additional routing and/or managing tasks.

The common requirement for a hierarchical routing is to increase the scalability and reduce the routing overhead. However, depending on different application domains, some additional requirements could be conflicting. The center attention of this thesis is in the application domain of VANET in the next Section 2.1.3. This application domain mainly uses the hierarchy in localization services, and VANETs are normally divided based on physical positions.

2.1.2.1 Additional Requirements for WSNs

This subsection focuses on WSN, another application domain to which the embedded system can be adopted. The requirements of this application domain are for energy efficiency and therefore extend the network lifetime. Therefore, all WSN techniques must consider the maximum/limited energy-efficiency in its priority mechanism. The cluster division for WSNs is normally based on the calculation of node energy, and in some cases it is also based on positions (e.g. smart home and telemedicine). The process to divide a network into clusters can be done explicitly. Or, as an implicit process in a regular routing is also possible (e.g. each node is assigned a *Home Region* scope). The other requirements for hierarchical routing for WSNs are as follows:

- There are two major communication modes in WSNs. The more often one is from sensors to a sink, which is for aggregating monitoring data and it must be auto-configured by nodes. Another one is from a sink to sensors, and it is mainly for managing and querying purposes. Normally, the node-to-node communication in WSNs is not required. A hierarchical routing protocol for WSNs must at least provide the first mode.
- The routing driven modes in WSNs can be divided into three subsets, and at least one of them must be provided. A traditional mode is the event-driven, which normally used for gathering the event data from regular monitoring tasks. The second mode is called query-driven (or sink-driven), which normally requires both ways of communication modes. This mode is normally used for tracking or controlling the specific area data. Moreover, a timer-driven mode supports to run more complex real-time tasks (e.g. smart environment applications [14]).

The candidate protocols that fit the aforementioned requirements are *Low-Energy Adaptive Clustering Hierarchy (LEACH)* [15] (many-to-one, event-driven), *Threshold sensitive Energy Efficient sensor Network protocol (TEEN)* [16] (many-to-one, timer-driven), *Distributed Aggregate Management (DAM)* [17] (two ways, query-driven). Moreover, with the recent advances in localization technologies, to use the position information in WSNs is partially possible. *Two-Tier Data Dissemination (TTDD)* [18] (grid position based, event-driven) and *Clustering and Fuzzy Position based routing (CFP)* [14] (fuzzy position based, timer-driven) are considered for WSNs by this reason.

2.1.3 Geographic Routing

The additional geographic routing techniques can overcome some of the drawbacks of topological ones in highly dynamic MANETs, and they have unique advantages for VANETs:

- A more adaptable forwarding strategy can be built based on the connectionless principle. If the destination position is known, the forwarding strategy can simply use the neighbor positions as metrics to forward packets. Because the process is mainly based on the local geographic information, a node has no need to first discovery routing paths and to maintain routing tables. Therefore, it can adapt to the frequent changes of the network topology with a lower routing overhead and delay.
- Using geographic information enables a more accurate routing decision. For example, a new type of multicast techniques, called *geocast*, can be used to deliver packets to a specific location or region. A geocast can normally have a small routing overhead and a better scalability comparing with the regular multicast.
- The infrastructure of VANETs is suitable for using geographic routing techniques. Normally, The nodes in a VANET can obtain the additional supports from vehicle equidment (e.g. localization and energy). The other additional supports from roadside infrastructure can also help to get the localization information. Besides, it is easier to control the forwarding direction because the moving vehicles are generally limited along the road.

Although the geographic routing techniques have their advantages, they have not yet been become broadly practical in the related civil projects because some drawbacks have to be overcome firstly. The general geographic routing techniques and their problems in VANETs will be introduced in this overview section.

2.1.3.1 Localization

Geographic routing techniques normally assume that a node can get the three types of positions: its own position, the positions of neighbor nodes, and the positions of destination node (or region). The assumption is based on the development of localization techniques, which is one of the main challenges for using geographic routing techniques in VANETs.

Our PAVIN experiments in [19] demonstrate the difficulty in developing a localization technique in a relative ad-hoc mode: a localization service may require a geographic routing to transmit the reference data in the first place; but for transmitting the data, a geographic routing may need to firstly acquires the correct geographic information from a localization service. Even the exchange of reference data can be done by other routing techniques (e.g. at least the flooding-based techniques in the worst case), it increase the technical complexity. The situation may be improved if a node can gain the direct supports from the roadside localization service, but the roadside localization system could be expensive to be implemented, and therefore it is difficult to be broadly adopted especially for rural areas. Moreover, an additional localization overhead will be conducted along with the routing overhead in either ad-hoc or centralized method.

Many civil localization services and related techniques have developed to solve the problems. While some of these localization techniques have been relatively mature (e.g.

GPS/DGPS), it is reasonable to accept the previous assumption and to develop the VANET routing techniques based on geographic information. More details about localization techniques are explained in details in Section 2.2.

2.1.3.2 Greedy Forwarding and Its Limitation

Consequently, the node in an geographic routing can forward data in the following five steps. The process forwards data towards a destination position, and it finishes when reaching it. The first three steps are mainly the localization steps. The next-hop forwarding strategies in the fourth step (details in Section 2.3) is another main challenge for using geographic routing techniques.

1. Determining the position of its own
2. Determining the destination position
3. Determining the positions of neighbors
4. Selecting a next-hop forwarding node in neighbors
5. Forwarding data to the next-hop node.

The early unicast strategies started from the late 1980s are all based on the *greedy forwarding* strategy introduced in Section 2.3.1: the forwarding node choose the next forwarder from its neighbors that are located closer to the destination position. For example, the source node *S* chooses *R1* to forward packets to the destination position *D* instead of *a* in Figure 2.1. If using a *contention-based greedy forwarding* strategy (in Section 2.3.2), there is no need to get the neighbor positions in the third step in advance.

However, only using greedy forwarding will meet a *void area* situation: there is no other nodes, which the forwarding node can reach, closer to the destination position than the forwarding node itself (thus it is not from *b* to *e*), and therefore the greedy forwarding will fail even if there is an existing routing path (from *R1* to *F3*). The situation is more serious for VANETs because the nodes are not distributed arbitrarily and averagely, and the topology of VANETs follows the shapes of roads. For WSNs, some nodes are frequently put into a sleep mode to save energy, so they may not react to the other nodes temporarily. In this case, a recovery strategy need to be adopted instead of the greedy one, which is introduced in Section 2.3.3 and Section 2.3.4.

Section 2.3.3 describes a more advanced recovery solution by *perimeter routing* (from *R1* to *R3*). The main idea of the perimeter routing is to try finding a routing path that surrounds the border of a void area based on the *right-hand* rule (*R1* and *R2* instead of *c* and *d*). A series of techniques including *face routing* and *planarization* are developed based on the right-hand rule. After let a void area, the perimeter routing can be switched back to the greedy forwarding if the greedy condition matching again (from *F1* to *F3*).

2.1.3.3 Alternative Geographic Strategies

Implementing the full computing for the greedy and recovery strategy is sometimes not enough for the VANET for its high mobility, thus the techniques of *Geocast* in Section 2.4 and *Delay Tolerant Network (DTN)* in Section 2.5 are developed to overcome the drawback.

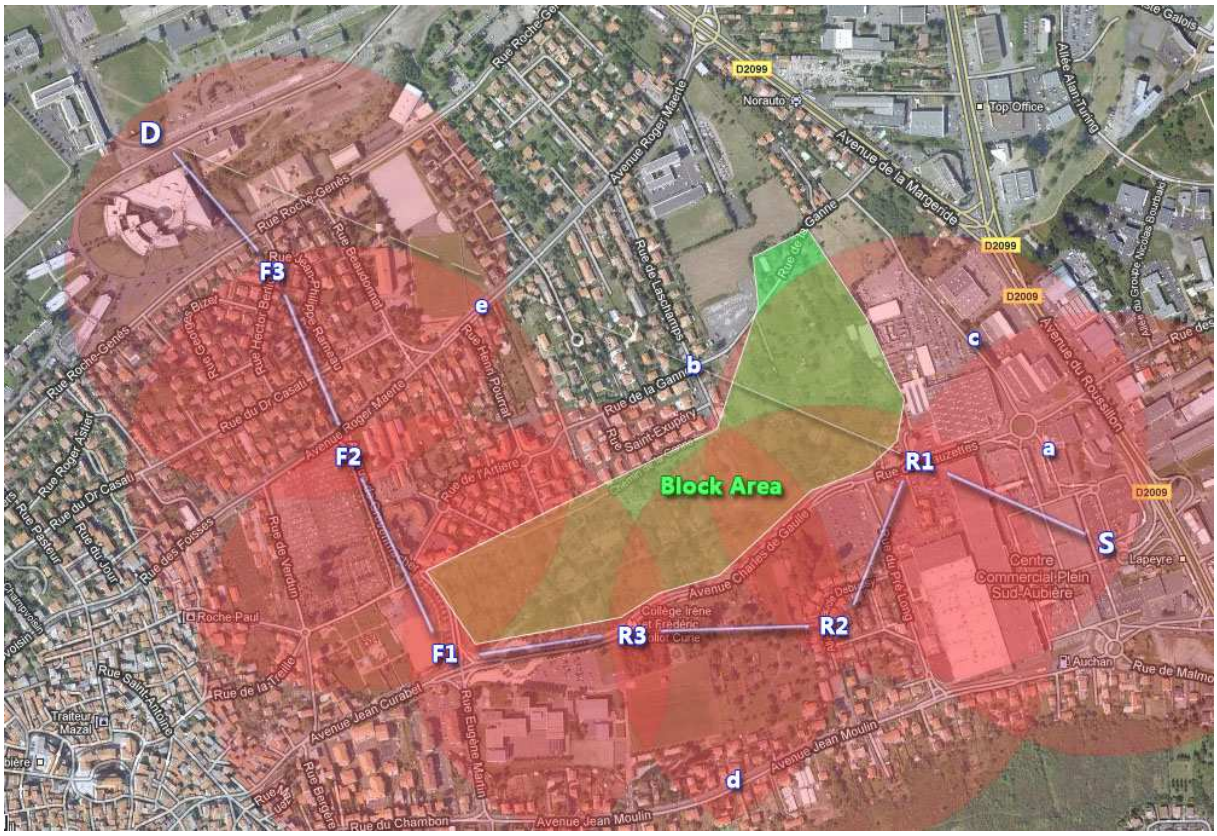


Figure 2.1: Local minimum problem in an example of VANET

The geocast is a multicast geographic routing strategy, which is used to deliver data to the nodes in a specific region. The flooding-based geocast technique such as the restricted directional flooding in Section 2.4.2 is more suitable for the highly dynamic network. With the supports from localization services, the geocast narrows the flooding area, and it does not require for getting a accurate destination position. Actually, the flooding is currently the foundation technique of the geocast protocols closely related to VANETs, and Section 2.4.3 focuses on these geocast protocols.

The aforementioned geographic routing techniques are more suitable for high dense VANETs, but it is not always the case due to the unique mobility features of VANETs. The DTN can be considered as an extreme case of MANETs characterized by the serious or complete lack of the end-to-end routing path because of low density and high mobility, which is exactly the case of VANETs. The recent researches consider using the *movement-based* and *carry-and-forward* strategies in the situation, and some of the related protocols will be introduced in Section 2.5.1 and Section 2.5.2.

2.2 Geographic Localization Services

The mostly used localization technique in current VANETs is the satellite-based navigation techniques such as GPS (United States), Beidou (China), and Galileo (European Union). However, it is not practical to assume that every node can be implemented such navigation device. Boukerche [20] summarize the civil localization services to overcome the GPS limitations:

- *DGPS*: correcting the positions from GPS based on the difference from the positions of reference stations
- *Map matching*: using the map knowledge to improve GPS positions
- *Cellular localization*: correcting positions by the mobile cellular infrastructure
- *Image/video processing*: providing positions through roadside security systems
- *Infrastructured indoor localization*: using the signal propagation characteristics for indoor environments
- *Dead reckoning*: calculating the current position based on the last known GPS position
- *Relative distributed ad hoc localization*: estimating the distance by the known GPS positions of other nodes

A bigger portion of them relies on the infrastructure supports, and solves the position of single node. The DGPS, video/cam localization and infrastructured indoor localization can provide a more accurate position than others, but they rely on centralized approaches to be realized. The dead reckoning can be independently completed by a node, but it is not accurate for a longer distance.

Most of the above localization services can help a node to get the position of its own, but not to get the position of neighbors and destination. The neighbor positions are normally learned through the periodical one-hop broadcast or the reactive neighbor knowledge querying, thus this step is relatively simple. The main issue here is how to discover the destination position. The destination position is normally specified in the forwarding packets from a source node (original sender). In the best case, the destination position is fixed, and the source node gets the position directly from the roadside infrastructure. In the worst cases, the source node uses the simple flooding to query the destination position. Between these two cases, the following localization services can be adopted.

There are two major processes for a localization service: *location update* and *destination query*. The former normally sends out the position-related information to a subset of nodes called *location server*, the latter searches the location servers to get a destination location. Here we divided the protocols based on the differences in the update and query strategies including *flooding-based* [21] [22], *hierarchical* [23], *home region* [24] [25] and *quorum-based* [26] localization.

Note that, the localization service for a destination node is an open issue. The flooding-based localization could generate a high localization overhead, and they are not scalable well, but they can have a low implementation complexity, and they are relatively robust in a small network section. The hierarchical, home region and quorum-based localizations can achieve the

network scalability, but these algorithms themselves may have too much impact on localization overhead, and they are easy to affected by node failures. Moreover, when taking into account the speed of nodes like vehicles, none of them can be said to be reliable.

2.2.1 Flooding-based Localization

Distance Routing Effect Algorithm for Mobility (DREAM) [21] represents a typical example of using the proactive flooding-based techniques: a node maintains a position table for the nodes that it can hear, and it tries to send its position information to the nodes that it can reach. In order to control the localization overhead in flooding, the DREAM protocol considers two effects between nodes: mobility and distance. The *mobility effect* is implemented as the flooding frequency. The node with a faster speed floods more frequently. The *distance effect* refers to the phenomena that if the distance between two nodes is greater, the relative movement to each other appears to be slower (e.g. for the node A in Figure 2.2, the node B seems moving slower than node C in the south direction). The packet to deliver the position information contains node id, position, direction and *age* (i.e. hop number). The *age* represents to the result of the distance effect. The receivers of such packet can then calculate their distance effect, and decide whether to discard the packet based on the *age* in the packet. Note that, DREAM only uses flooding in the destination discovery, not for the data delivery.

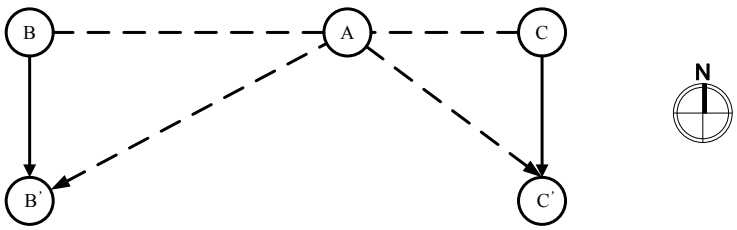


Figure 2.2: An example to show the distance effect

Another variation of flooding-based localization technique is used in *Location-Aided Routing (LAR)* [22]. When nodes do not have any knowledge about the network, LAR works similarly to DSR [6] and AODV [5]: reactive request process, avoiding redundant requests in a flooding, and the information about route and location is contained in the packets.

After the raw position of destination node is known, both DREAM and LAR then use the *restricted directional flooding* described in Section 2.4.2 to continue the localization services to get more accurate destination positions, or send data packets.

A common issue of flooding-based localization services is still the scheduling mechanism as in a topological route discovery: the choice of proactive or reactive. The advantages of flooding-based techniques are that the data distribution is mostly decentralized, and the implementation complexity is lower than the other localization techniques. However, they are not well scalable for large networks, thus only the subset of a VANET (e.g. the short ad-hoc sections between cities) may adopt the technique.

2.2.2 Hierarchical Localization

The hierarchical localization (or *hierarchical hashing-based quorum-based*) normally explicitly divides nodes into a hierarchical layer structure based on the node positions, and at least a node in each layer acts as a location server that responses for updates and queries for the nodes. The hierarchical localization services can help to reduce the localization overhead and achieve the network scalability, but whether it is robust enough to nodes mobility like VANETs will need more evaluations to prove. Here we only introduce a typical protocol named *Grid's Localization Service (GLS)* [23], which has some characters to be suitable for VANETs.

The GLS protocol provides a decentralized hierarchical algorithm, which can handle low-mobility nodes with a less localization overhead. If all nodes know their GPS positions and they agree on a global origin of the hierarchy as shown in Figure 2.3, the algorithm of GLS can be done by the nodes themselves. The layer in GLS is referred to as an order-*n* square. A number of order-*n* squares make up an order-*n+1* square as the next layer, and so on. The nodes in the same square must in each other's one-hop communication distance, and the maximum communication distance is assumed to be two hops. Note that, the location update and destination query service does not completely rely on the rules for geographic division.

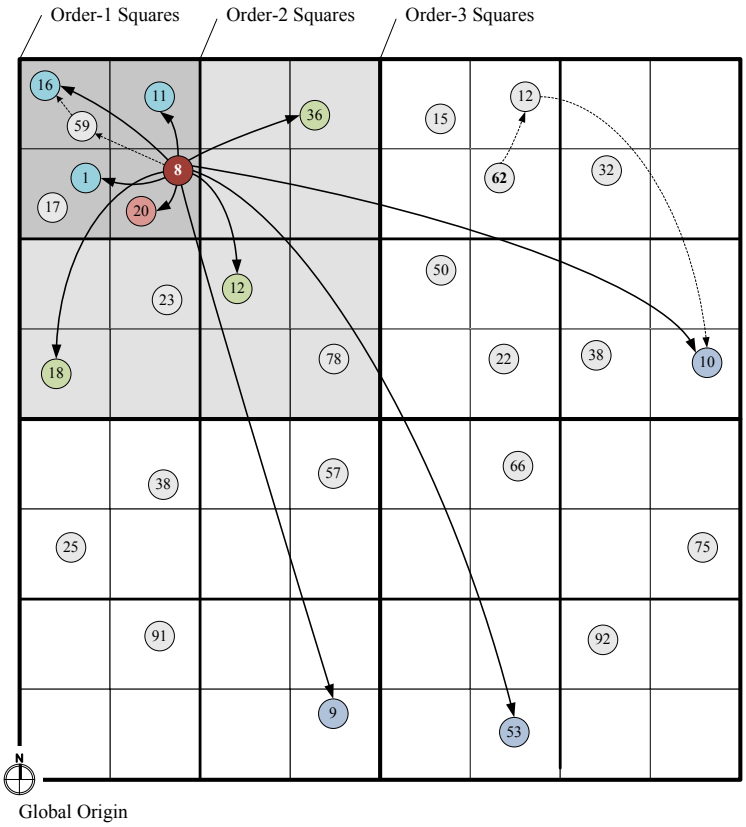


Figure 2.3: An example of GLS localization services

For the location update (e.g. the node 8), each node periodically deliver its ID to all one-hop neighbors in its first-order square (e.g. to node 20). Then the location is delivered to the assigned location servers in the next layer (e.g. node 1, 11, 16; maybe delivers from 59 to 16 but it is not important for the algorithm), and the process continues until the ID are delivered to the assigned location servers in all layers (e.g. node 12, 18, 36, then node 9, 10, 53). For each square in the

next layers, only one location server will be assigned. The assigned location server is the node with the least ID greater (or greatest ID less) than the ID of the source node; in the other word, the node with the closest ID is chosen. For the destination query, it uses the similar process, which tries to find the location server with the closest ID to the destination ID from its layer to the next ones (e.g. node 62 to 12, then 10), and a location server that has stored the ID of the destination will be found eventually.

To support larger networks, the IDs in GLS protocol are assigned by the hashing function intuited from the *consistent hashing* in [27] and similar to the ones in Landmark system [28]. These hashed IDs are assumed to be mapped from IP addresses, MAC addresses or other uniquely allocated names, thus a limited mobility for nodes is allowed. Moreover, it is possible to further introduce the fuzzy localization in to the hashing function, thus not all nodes need to know their accurate GPS positions.

The GLS protocol balances the localization overhead by evening the assigning of location servers. Moreover, because the GLS protocol delivers the location update and destination query based on layers, the localization overhead can be greatly reduced and it is predictable: if the height of the hierarchy is $O(log(N))$, effectively the location update and destination query is delivered to $O(log(N))$ location servers, where N is the number of nodes.

2.2.3 Home Region Localization

The home region localization (or *flat hashing-based quorum-based*, e.g. Stojmenovic's [24], *GHT* [25] and *GHLS* [29]) is more often used in *data* localization services instead of *position* localization services [30]. These two types of localization services are similar in general, but the data localization services generally have less sensitivity to mobility. For example, they may depend on the nodes' arrivals in certain regions and departures from them, or just the new data advertisements and disposals, thus their algorithms are not necessarily to be responsive to nodes' accurate position changes. Although some concept of home region localization may be used in the infrastructure design of VANETs, it is normally used in regular MANETs and WSNs.

Similar to the aforementioned GLS protocol, location servers are used in the home region localization, and a hash function is used by all nodes to produce the ID. One or more fixed nodes act as location servers and inform other nodes about their existing. They together designate as the home region of the network, and other nodes store their IDs to the location servers. For example, each node in GHT [25] hashes keys into geographic positions, and it stores a key-value pair at the node geographically closest to the hash of its key.

Because the number of location servers in the home region is independent of the total number of nodes, thus effectively the location update and destination query are delivered to $O(1)$ location servers.

2.2.4 Quorum-based Localization

The *quorum-based approach* meaning is that all nodes in the network agree upon a *mapping* that maps their unique identifier to one or more quorums. The quorums respond for the specified functions of other nodes. By these definition, the hierarchical and home-zone localizations in the two last sections can also be considered as the *hierarchical* and *flat hashing-based*

quorum-based localization [29] [30]. The hashing-based means that the quorums are chosen by a hashing function to build a distinct hierarchy.

For *quorum-based localization*, it normally means that nodes sends location updates to a subset of nodes (i.e. location servers), and location destination query to another. These two subsets of nodes must have the intersection nodes to assure a virtual connection backbone. In other cases, if two subsets of nodes are identical, they can also be called as *rendezvous-based* [30].

Here we only introduce about the classical quorum-based localization called *column-row* localization such as in DS-quorum (Dominating Set quorum) [26] or XYLS [29]. The DS-quorum protocol proposes an algorithm that divides a network into connected dominating sets as shown in Figure 2.4. The dominating set of a graph $G = (V, E)$ is the subset D of V where the set of vertices in G is either in D or adjacent to a vertex in D. The nodes representing the location servers are arranged in a form of columns and rows, for example, the location servers in rows may respond for the location update, and the ones in columns may respond destination query. Then, the location update is delivered from the current location of sender to north and south, until reaching the location servers in rows. The destination query is delivered from the current location of sender to east and west, until crossing the location servers in columns, and then pass to the intersection nodes with the queried location updates. Because the DS-quorum network deliver in the column-row form, effectively the location update and destination query are delivered to $O(\sqrt{N})$ location servers.

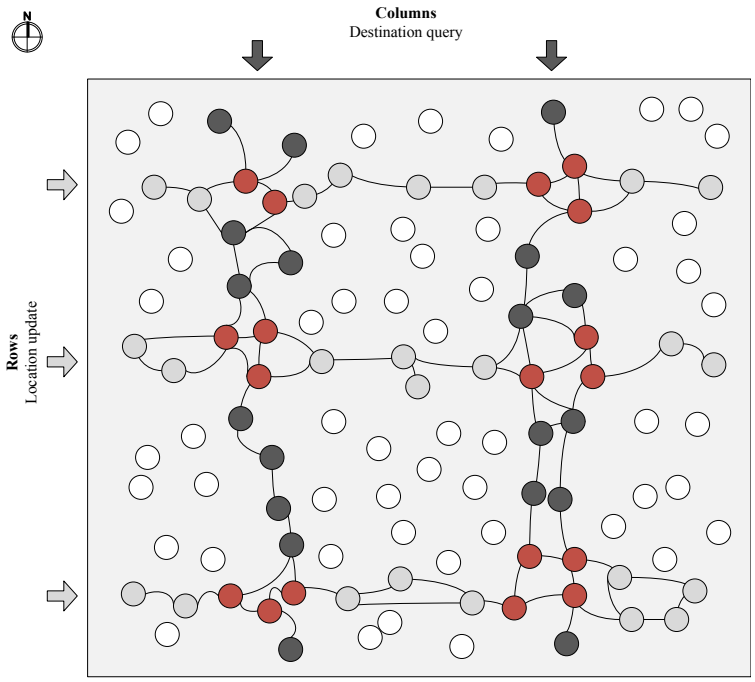


Figure 2.4: An example of DS-quorum localization services

As for used in VANETs, there are three advantages of the column-row quorum-based localizations: Firstly, they adapt well to synchronous vehicle movements on roads; Secondly, they can be used to form a network backbone for mixing ad hoc and infrastructure communications; Thirdly, they are able to better utilize the GPS information about longitudes (columns) and latitudes (rows).

2.3 Greedy Forwarding and Recovery Mode

2.3.1 Next-hop Candidates

When the positions of a node itself and destination are known from the localization service, a geographic greedy routing will forward a packet to one or more next-hop nodes with the maximum forwarding progress. The next-hop node can be selected based on the strategies following the Figure 2.5.

A geographic next-hop selection algorithm is normally defined in a *Cartesian* coordinate plane in two dimensions. The network model is assumed to be the unit disk graph where nodes can communicate within radio range R. In the Figure 2.5, the node at s is the last sender and the node at d is the destination. From point s to d, it is called *progress direction*. The area within the radio range and from y-axis toward the progress direction is called *progress area*. A algorithm can also select the next hop in a smaller progress area, e.g., the *maximum forwarding area* with a margin in the form of an arc having the center at d.

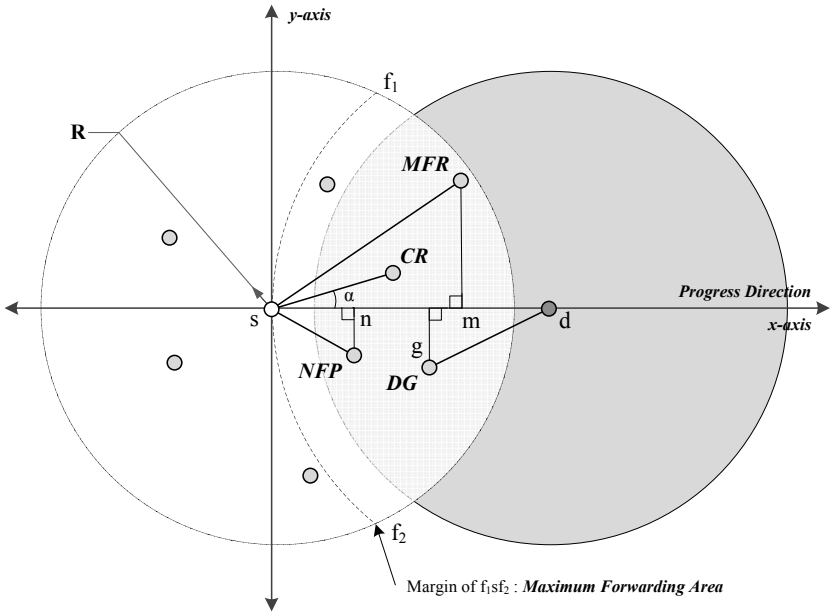


Figure 2.5: Next-hop selections in greedy forwarding

- *Most Forward progress within Radius (MFR) [31]*: This strategy select the node with the longest projection distance in progress direction (e.g. the distance of *sm*). The longer-range transmission is favorable because it may takes a packet to more hops ahead. If there is no other node as the next hop to forward a packet, MFR sends the packet back to the previous node.
- *Nearest with Forward Progress (NFP) [32]*: The node with the shortest projection distance in the progress area is selected (e.g. the distance of *sn*). The strategy favors shorter-range transmission because it may minimize transmission energy consumption (it depends on the underlying layer functions) and it has a lower probability of packet collisions in the contention-based forwarding (Section 2.3.2).

- *Distance-based Greedy (DG) [33]*: The strategy is originally proposed for wire networks. It select a node that minimizes the distance to the destination (e.g. the distance of gd). Its advantage is similar with MFR.
- *Compass Routing (CR) [34]*: It is the first proposal to using the minimum angle in the next-hop selection. It selects the node with the minimum angle between the node and destination (e.g. the angle of α). The nodes closer to the y-axis in the progress direction will consume more energy under this strategy.

The original NFP and CR have the problem of routing loop, but MFR and DG are loop-free [35]. The routing loop causes a packet circulate among certain nodes.

2.3.2 Beacon-based or Contention-based

A *beacon-based* forwarding requires knowing the positions of one-hop neighbor nodes, which can be achieved by neighbor knowledge exchanges (or called beacon exchanges). After the neighbor positions have been achieved, the selection for the next-hop node can be done by the sender itself. The beacon-based forwarding has less implementation complexity, but it relies on the underlying layer to provide a one-hop unicast mechanism. The neighbor knowledge exchange causes additional routing overhead, but the probability of packet collisions can be reduced if the frequency of exchanges is well controlled.

A *contention-based* forwarding does not rely on neighbor knowledge exchanges. A sender may blindly broadcast a packet, then the nodes that receive the packet auto-configure if they can be the next-hop forwarders. To minimize the packet collision, the number of forwarders needs to be limited by three restrictions.

- The first restriction is that only the nodes in a progress area are selected. An implementation for this restriction is relatively simple. A sender can add its position and the destination position in a forwarding packet. Plus each node already knows its own position. The nodes that receive the packet can calculate whether they are the required forwarders independently.
- The second restriction is optional, and it is to limit nodes in the area that the nodes can hear from each other. If a node has forwarded the packet, the other nodes can then know that and stop the redundant forwarding. If the RTS/CTS (Request to Send/Clear to Send) mechanism is not implemented, the restriction is required. There are three optional forwarding areas proposed in *Beacon-less Routing (BLR)* [36]: a circle with the diameter equaling to the radio range R, a Reuleaux triangle with the maximum apex angle of $60°$ on the sender position, or a sector with the same condition of the Reuleaux triangle as shown in Figure 2.6. Comparing the size of a circle with the radius of radio range R, the circle, Reuleaux triangle and sector limit the forwarding area to $\frac{1}{4} \approx 0.25$, $\frac{1}{2} - \frac{\sqrt{3}}{2\pi} \approx 0.22$ and $\frac{1}{6} \approx 0.17$.
- If only with the previous two restrictions, the node closest to the sender will normally receive and forward the packet firstly. In order to enable more geographic forwarding options and to increase the time lag for other nodes to react to the first forwarding, an additional timer delay function based on the geographic options in Section 2.3.1 is

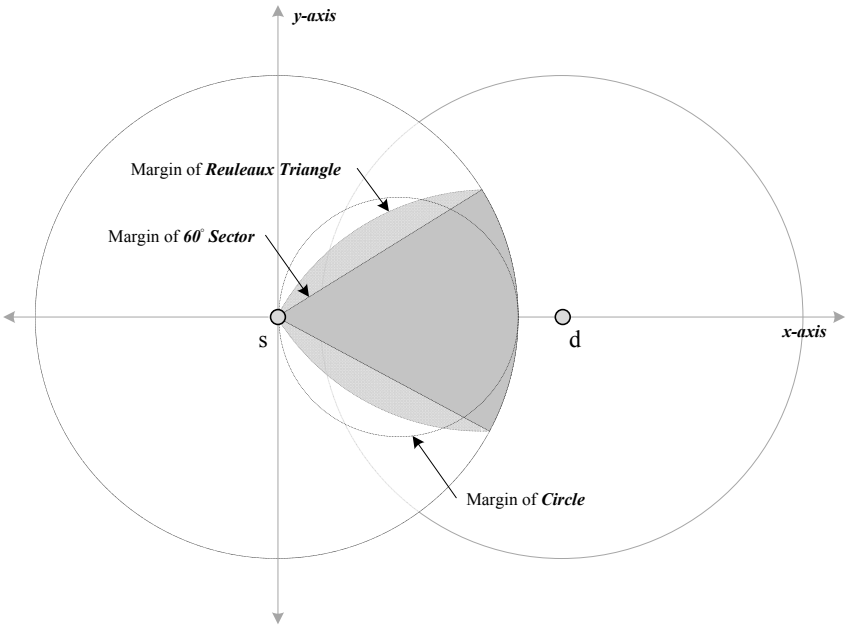


Figure 2.6: Optional areas in contention-based forwarding

implemented to nodes. For example, if the direct distance d is used, the delay t can be calculated as $t = \frac{a}{d} \cdot Delay_{MAX}$, where a is the parameter to adjust the advance progress. If this restriction is adopted to the MAC layer protocol, the $Delay_{MAX}$ is the maximum delay to keep a packet before dropping it.

The typical protocols in the contention-based category use the restrictions similar to previously mentioned ones. More of the protocols are proposed for the IEEE 802.11 MAC layer protocols, but with different implementations and extensions of the timer delay function. *BLR* only select one node with the maximum projection distance in a 60° sector toward the progress direction. *Implicit Geographic Forwarding* (IGF) [37] is similar to BLR, but it is implemented as the optional RTS/CTS mechanism in IEEE 802.11 standard. Both BLR and IGF have a drawback that they do not exploit all possible forwarders in the progress area.

If a RTS/CTS mechanism is adopted, the second restrictions on forwarding areas can be omitted, and a bigger forwarding area such as the *maximum forwarding area* in Section 2.3.1 can be exploited [38] [39]. In this case, it is better to use an unicast mechanism in the actual data packet transmission after the RTS/CTS finish.

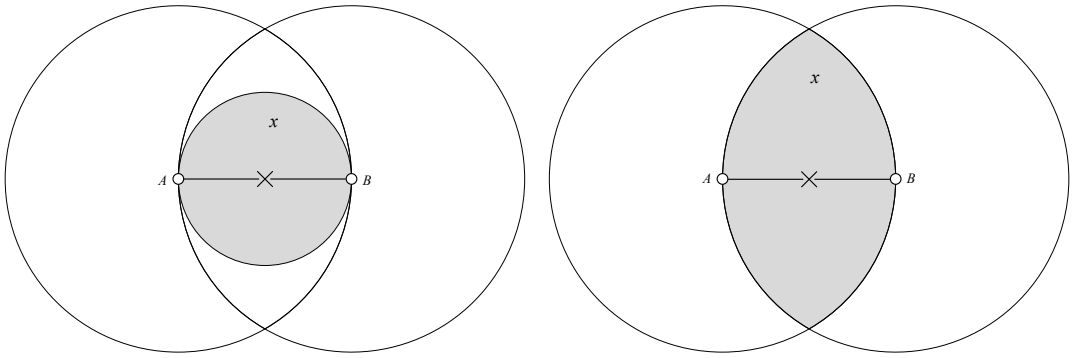
2.3.3 Perimeter Routing for Void Area

The major challenge for a greedy forwarding is an obvious one: the progress area of last sender could be a void (the *local minimum* problem of graph theory), thus the forwarding packet is blocked. The recovery solutions for this situation have been proposed to work with the greedy forwarding.

The recovery solution of *perimeter routing* (also named *face routing* or *face traversal*) is an advanced recovery solution in the state of art of geographic routing. A greedy forwarding plus a perimeter-based recovery represents the main direction of current researches about geographic

routing. Although the performance of perimeter routing could rely too much on ideal static network conditions, but it is the only resource-aware recovery solution that can guarantee the packet delivery (if a routing path does exist) by requiring just the neighbor information. Besides, it can work on both beacon-based and contention-based networks.

Perimeter routing is a recovery solution based on *planar graph*, which is a type of the graph with its edges that intersect only at their endpoints. A graph representing a wireless network does not naturally form as a planar graph, thus the graph need to be simplified by a *planarization* process. A non-planar graph reduces the performance of a perimeter routing, and it may cause the routing-loop problem [40] [41]. The challenge for the planarization process in geographic routing is that the nodes can only know the neighborhood information, thus a full planarization for the whole graph is not practical.



(a) GG (b) RNG

Figure 2.7: Planarization areas of GG and RNG (in gray color)

Two notable planarization algorithms which require only the neighborhood information are *Gabriel Graph (GG)* [42] and *Relative Neighborhood Graph (RNG)* [43]. For both algorithms, if any node x exists within the neighborhood ranges of both A and B (the areas with gray color as shown in Figure 2.7), the edge of (A, B) is removed to avoid the possible crossing edge. The remaining edges after the planarization are (A, x) and (x, B). *GG* defines the neighborhood range as a circle with a diameter as the line segment (A, B). *RNG* defines the neighborhood range as the intersection of two circles with radius as R and the circles are centered at A and B. *GG* and *RNG* offer different densities of remaining edges (wireless links). *RNG* produces the planar subgraph with fewer edges thus it reduces the routing overhead; on the other hand, *GG* produces the planar subgraph with a better connectivity thus it may reduce the hop number to a destination.

After the localized planarization process, the nodes get a local view of a planar subgraph without edges crossing each other. The next strategy of perimeter routing is to adopt the *right-hand rule (or left-hand rule)* on traversing on the borders of the faces in the planar subgraph. The packets are forwarded face by face, and progressively get closer to the destination position.

The first version of the recovery solution using perimeter routing is proposed in [44], which includes two routing algorithms named *FACE-1* and *FACE-2*. Figure 2.8 and Figure 2.9 demonstrate them as the stand-alone routing process without returning to greedy forwarding. The packet in both figures is assumed to be sent from the source node S to the destination node

D by a sequence of faces (e.g. from $F1$ to $F3$).

The key rules for *FACE-1* in Figure 2.8 is to find the edges that intersects with the line segment from the source to the destination (e.g. $\overline{SD}$), and the founds edges (e.g. (A, B) and (E, F)) should be closer to the destination gradually (e.g. from $F1$ to $F3$, the distances $dist(S, D) > dist(p_1, D) > dist(p_2, D)$). Before a packet is passed to the next face in *FACE-1*, the packet must do a complete traversal thought the border of a face and then return to the initial point (e.g. S, A or F).

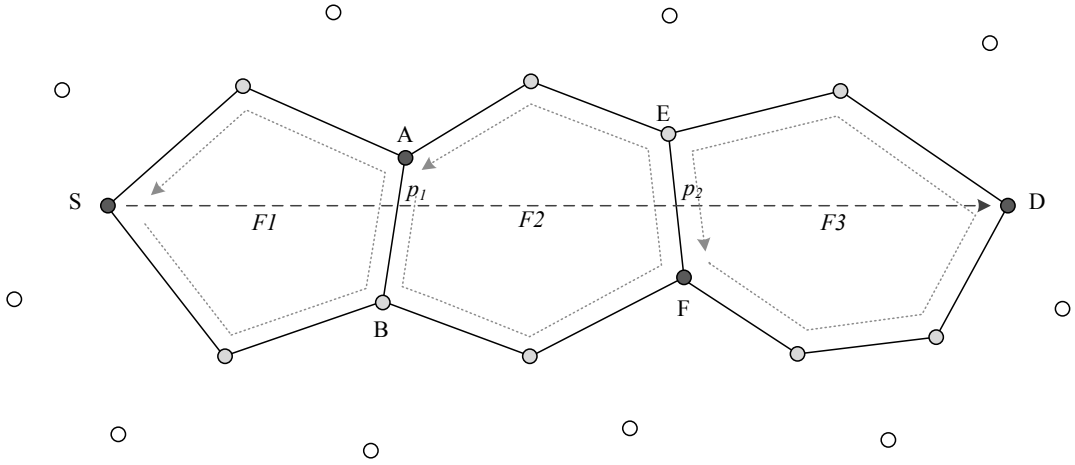


Figure 2.8: An example of the routing path by FACE I

FACE-2 in Figure 2.9 is a modified version of *FACE-1*. When a packet is passed to the node with an edge intersecting with the line segment $\overline{SD}$, the packet is delivered directly to the adjacent face instead of returning to the initial point (e.g. from B to F, instead of back to S).

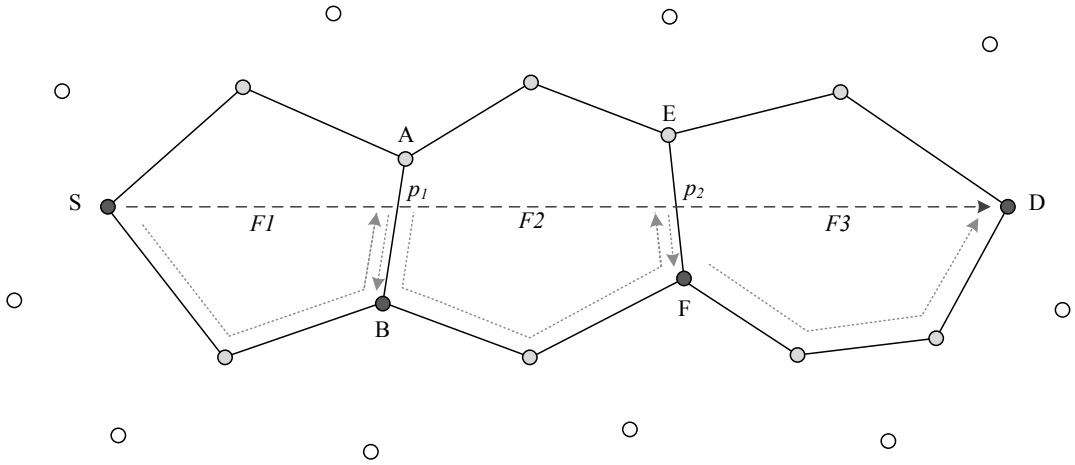


Figure 2.9: An example of the routing path by FACE II

Both *FACE-1* and *FACE-2* algorithms are not very efficient on their own, but they can guarantee the packet delivery without requiring the duplication of packets or memory. Normally, they are used as the recovery solutions to incorporate with the greedy forwarding. The first geographic routing algorithm named *Greedy-Forward-Greedy (GFG)* is proposed in

[44]. The algorithm adopts *GG* [42] for planarization, and it incorporates the *FACE-2* and a distance-based greedy forwarding strategy named *GEographic DIstance Routing (GEDIR)* [35].

A well-known beacon-based geographic routing protocol, named *Greedy Perimeter Stateless Routing (GPSR)* [45], implements a recovery solution similar to *FACE-2* (requiring one-hop neighbor positions). GPSR proposes the protocol-level details for face routing and an alternative planarization algorithm (*RNG* [43]). When switching faces by GPSR, the packet is always delivered through the first edge of the next face by adopting the right hand rule. Such first edge must be recorded in the transmitting packet until it reaches the next face in order to avoid the routing-loop problem. Then, the next edge is searched by the counterclockwise direction from the last edge.

Adaptive Face Routing (AFR) [46] is a variant algorithm of *FACE-1*. The source node in AFR initially estimates a boundary of *FACE-1* as an ellipse with foci on source and destination. When a packet reaches the boarder of the ellipse, the packet is delivered back to the last initial point. The packet is then sent to the initial point of next face. If the routing path is blocked because the ellipse is too small, the packet is sent back to the source node, and the size of the ellipse is increased. If c is the cost of the best path in *FACE-1*, AFR can achieve a worst case cost of $O(c^2)$. Besides, GOAFR+ [47] combines the greedy forwarding and AFR.

The open issue of such hybrid solutions is that they rely too much on ideal wireless network conditions, more precisely, the radio range of these solutions is assumed to be uniform as R in an unit disk graph. However, the realistic radio range is more often to be irregular (i.e. *quasi unit disk graphs*) because of the different densities of wireless medium, link errors and inaccurate positions from localization service. Some solution is proposed for the non-ideal network conditions, for example, *CLDP* [40] uses an additional proactive message for planarization, and *GDSTR* [48] use the traversal of a *hull spanning tree* (an alternative technique of planarization). The former increases the routing overhead significantly, and the latter loses the localizable advantage in geographic routing. Moreover, none of these perimeter routing protocols fully considers the mobility of nodes.

2.3.4 Other Recovery Techniques

The other solutions or suggestions for dealing with the void area problem are introduced in the following, but note that, a part of them only consider a low mobility of nodes, thus they are only given as the further references and they are not be completely practical for current VANETs.

- *Dropping*: Of course, the simplest solution is not to recovery the current packets if meeting a void area. Dropping the blocked packets can be an option if the nodes are generally moving and a resend mechanism is available, or a multi-path routing is adopted. A further improvement to be suitable for static networks is by GeRaF [38]. GeRaF is a contention-based forwarding which suggests to mark the void areas or directions if a packet is blocked, and the marking results is reflected in a RTS/CTS mechanism. SPEED [49] on the other hand is a beacon-based solution, which considers dropping the blacked packet for reducing the traffic congestion, and it deals with a void area in the similar way as a congestion area. Each node in SPEED records the average delays to destinations in its neighbor table. When meeting a void area, the delay is marked as ∞. The neighbors then get the notice for the void area by the so-called *backpressure beacon*.

- *Backtrack-and-marking*: Another suggestion is to pass the blocked packets back to the last forwarder (MFR [31]). The failing routing path must also be marked, thus the new greedy forwarding will look for another path and avoid routing-loop [35]. If the mobility of nodes is considered, any node in the similar position of the last forwarder can be used as a backtracking node. Furthermore, *GDSTR* [48] maintain a spanning tree where each node has an associated *convex hull* that contains within it the locations of all its descendant nodes in the tree. When a void area is found, the block packets are routed upwards in the tree until finding a node whose convex hull contains the destination.
- *Reactive Flooding*: If the node density and mobility are low, an optional suggestion is the reactive flooding-based techniques, e.g., the topological protocols with reactive route discovery scheduling (AODV [5] and DSR [6]).
- *Exploiting Two-hop*: This recovery solution works better in a beacon-based network. If two-hop geographic information such as GEDIR, DIR, and MFR is available for each node, the void area can be predicted or avoided [35]. The trade-off for the two-hop geographic information is an additional routing overhead, but it is not addressed in details in [35].
- *Geocasting*: The geocast can not avoid the void area problem, but the multipath created by geocast can reduce the chance for packets to be blocked by void areas. More details about the geocast techniques are in Section 2.4.
- *Carry-and-forwarding*: For networks with high mobility nodes, the nodes can hold the packets and wait until a next candidate forwarder or the destination node to present (e.g. VADD [50], GeOpps [51], and GeoDTN+NAV [52]). More details about the carry-and-forward techniques are in Section 2.5.

Actually, if the mobility and failure of nodes is taken into account, none of the previous algorithms and protocols can be considered to be absolutely reliable. A part of them just assume that the speed of packet delivery is the same as light and node failure is zero in their simulations, thus it is difficult to say that such algorithms or protocols are really reliable in dynamic networks like VANETs.

2.4 Geocast Strategies

The geocast forwarding steps are similar to the unicast ones, but the destination in geocast is restricted as a geographic region. If the destination is only a single node, when packets reach the border of the destination region that contains the node, the transmitting mode can be switched back to the unicast mode. A geocast is normally assisted with two other techniques: *hierarchy* and *flooding*.

The hierarchical geocast forward packets region by region, thus it can reduce routing overhead and increase network scalability. Besides, the region concept can be utilized to mark void areas more efficiently. However, the trade-off of these advantages is a clustering overhead. For highly dynamical networks like VANETs, considering the clusters could be too short-lived to worth creating, the hierarchy techniques may be only suitable for the localization service when there is the support from infrastructure Section 2.2. The current hierarchical geocast protocols (e.g. GeoTora [53] and GeoNode [54]) are more often used for the wide-range transmission of infrastructure networks and regular MANETs [55].

The following sections only describe the non-hierarchy flooding-based geocast techniques. Under this context, the geocast applications in the following sections is only for distributing emergency messages, for example delivering a collision warning to approaching vehicles and nearby junctions. In the following, we first introduce the basic flooding options, and then two well-known flooding-based geocast techniques. In the end of the section, the flooding-based geocast techniques close relating to VANETs are given.

2.4.1 Basic Methods in Flooding

Due to resource constraints of embedded sensor and negative effects from radio irregularity, flooding (i.e. global broadcasting) may be a suitable transmitting scheme for IVC routing algorithm. However, flooding in an MANET could cause serious redundancy, contention, and collision [56]. Therefore, it is important to determine a correct flooding technique for CIVIC protocol. Williams [57] classifies current flooding techniques of MANET to four categories:

- *Simple Flooding (SF)*: In SF, every node rebroadcasts a packet exactly once until all reachable nodes have received the packet. SF is adapted to low node density and/or high mobility networks.
- *Probability Based Method (PBM)*: PBM is similar to SF, but every node rebroadcasts a packet with predetermined or counter-based probability. When the probability is 100%, this method is identical with SF.
- *Area Based Method (ABM)*: Instead of probability, every node decides whether to rebroadcast a packet depending on an estimation of distance or location.
- *Neighbors Knowledge Method (NKM)*: Every node makes a decision on rebroadcast by its one-hop or two-hop neighbor knowledge. The neighbor knowledge is achieved by the periodic "Hello" packets.

Tseng [56] proved that the adaptive counter-based and location-based scheme could resolve the dilemma between reachability and broadcast storm. The interval of sending "Hello" packets

is also important to achieve efficient broadcast. Moreover, the radio irregularity may seriously affect directional routing especially when a packet can only be sent to one direction. Zhou [58] proposes using the multi-round discovery technique to solve the problem.

2.4.2 Restricted Directional Flooding

DREAM [21] and LAR [22] are two broadly adopted geocast protocols. They both adopt the restricted directional flooding in their data transmission, but their restricted areas are different.

By the steps introduced in Section 2.2.1, assuming a source node S in DREAM or LAR has known that the destination node D is in the position of (x_d, y_d) at time t_0, and that the current time is t_1, the node can then restricts the direction and area of the next flooding as shown in Figure 2.10. The key scheme for both protocols is to assure that a packet is sent to an expected region that the destination node will be there when the packet reaches the expected region.

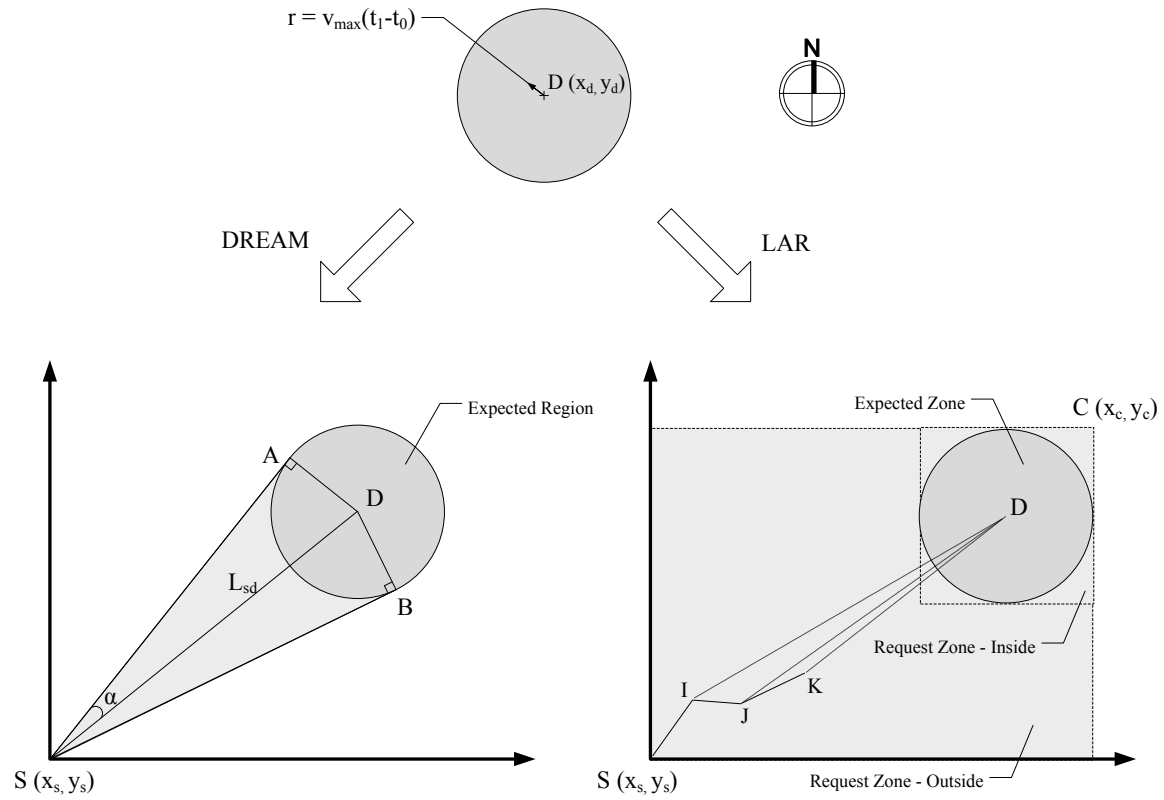


Figure 2.10: Flooding areas in DREAM and LAR

Both DREAM and LAR expect the node D is in the circle area centered at (x_d, y_d) with the radius of $r = v_{max}(t_1 - t_0)$ (e.g. *the expected region (zone)* are the same circle area in the north-east from node S), but the next steps are different:

- For DREAM, the nodes involved in the flooding process are the ones within the forwarding angle α defined as $\alpha = \arcsin \frac{r}{L_{sd}}$, where L_{sd} is the distance between nodes S and D.

- The LAR (scheme I) defines a *request zone* as a rectangular, where only the nodes in the rectangular floods the data packets. If a nodes S is outside the expected zone, the rectangular is within (x_s, y_s) and (x_c, y_c). If a nodes S is inside the expected zone, the minimum boundary of the request zone can not be smaller than the expected zone. The LAR (scheme II) further defines that only the nodes with the shorter distances to destination node can be involved in the next-hop flooding process (e.g. I, J, K).

2.4.3 Flooding-based Geocast for VANETs

The early examples of flooding-based geocast protocols for VANETs are a geocast scheme based on the IEEE 802.11 standard [59] and a protocol named *Inter-Vehicle Geocast (IVG)* [60]. The basic strategies of them are similar.

Firstly, when an accidence is happening, an alarm need to be sent out to all the vehicles that will be affected by the accidence. For example, if the accidence is in a roundabout, only the vehicles driven toward the roundabout will receive the alarm but not the vehicles that are leaving. The destination area that contains the alarmed vehicles is called *critical area*. If vehicles know their GPS information, the critical area can be defined easily. Secondly, when an alarm is spread in the critical areas, not all the nodes need to be involved as relay nodes. The method to limit the number of relay nodes is the same method in Section 2.3.2. The alarm will not be sent out immediately. A distance-based timer hold the alarm in waiting, and a node only rebroadcast the alarm when the node does not receive the same alarm from other nodes.

Besides [59] and [60], some other geocast algorithms and protocols are interesting enough to be introduced. Their basic techniques are similar to or have been introduced in the above sections, thus only their unique feature will be listed out as follows:

- *Cached Geocast* in [61] propose to include caching at the routing layer to deal with the situation of high velocities in VANETs. The small cache can help to improve the problems of neighbor selection and void area in the geocast forwarding.
- *Urban Multi-hop Broadcast (UMB)* protocol [62] redefines the RTS/CTS mechanism in IEEE 802.11 standard to address the problems of broadcast storm, hidden node, and reliability problems of multi-hop broadcast in urban areas. The UMB divides the road into segments in the direction of dissemination, and only one vehicle in each segment is on duty of forwarding and acknowledging the packets.
- *Abiding Geocast* is a specific geocast service considered in [63]. The abiding geocast is a geocast to be sent to a fixed geographical area (e.g. the warning of an icy road in winter). Besides the regular approach such as the periodical delivery, [63] provides three more options to realize an abiding geocast: the server approach, the node election in the destination region, and the neighbor exchange based solution.

2.5 Geographic DTN-based Strategies

DTN is an extreme case of MANET, and VANET can be treated as a form of DTN. Compared with the regular MANETs, the distinguished feature of DTN is that the end-to-end connectivity between source and destination in DTN is assumed to be broken due to frequent or constant network partitioning. The earliest works about DTN routing mostly use the flooding-based techniques. A direction of the recent works about DTN tries to utilize the movement feature of nodes instead of adapting to it. The techniques in such direction are very suitable for VANETs.

There are two main options of using the movement feature in VANETs: *Last Encounter Routing (LER)* and *Carry-and-forward Routing*. Some literatures further suggest controlling the mobility of the mobile nodes to help message forwarding. These suggestions are not practical for current VANETs thus it will not be mentioned in the latter sections.

2.5.1 Last Encounter Routing

A pioneering example of LER is a routing algorithm called Exponential Age SEarch (EASE) [64] [65]. The paper [64] first proposed a movement-based localization service, and it shows that it is possible to only use the node mobility to disseminate destination location information without using any flooding-based method. In other word, only "free" information about the local connectivity to neighboring nodes is adopted. Then, a simple routing algorithm named EASE was proposed to evaluate such localization service. The interesting conclusion about EASE is that the collections of last encounter histories at network nodes contain enough information for a geographic routing protocol to route packets.

For the part of localization service, each node in EASE maintains a *Last Encounter Table (LET)*, which contains three fields including *Node ID*, *Location* and *Time*. If a node i meet a node j at position P_{ij}, node i records an entry as *Node ID* equaling j and *Location* equaling P_{ij}. *Time* for the entry is the time elapsed since the encounter at P_{ij}.

As for the routing part, the principle is simple: when a source node tries to send a packet, the source node search its neighbors until finding a neighbor who meets the destination in the latest time based on the information of LET. Then the packet is routed toward the latest encounter location. The process is continuing until the packet reaches the destination node. For example, the vehicle S tries to send a packet to vehicle A as shown in Figure 2.11. In its current radio range, the vehicle B uses to meet the vehicle A at the location of B_1. If the location B_1 available on B is newer than any other locations information that the vehicle S can get, the packet is sent to the location B_1. The EASE made no assumptions about how to route the packet toward a latest encounter location, and any geographic routing protocol can be used here. The disadvantage of EASE is the delivery is easy to fail in a practical network when the network just starts up, or where there is a limited radio range thus the number of neighbors is too small.

The recent application of LER is FleaNet [66], which is a virtual flea market over VANET. The customers express their demands/offers by smart phones, PDAs and laptops within a VANET, thus the flooding-based techniques are not practical. The FleaNet uses the similar LER methods as in EASE.

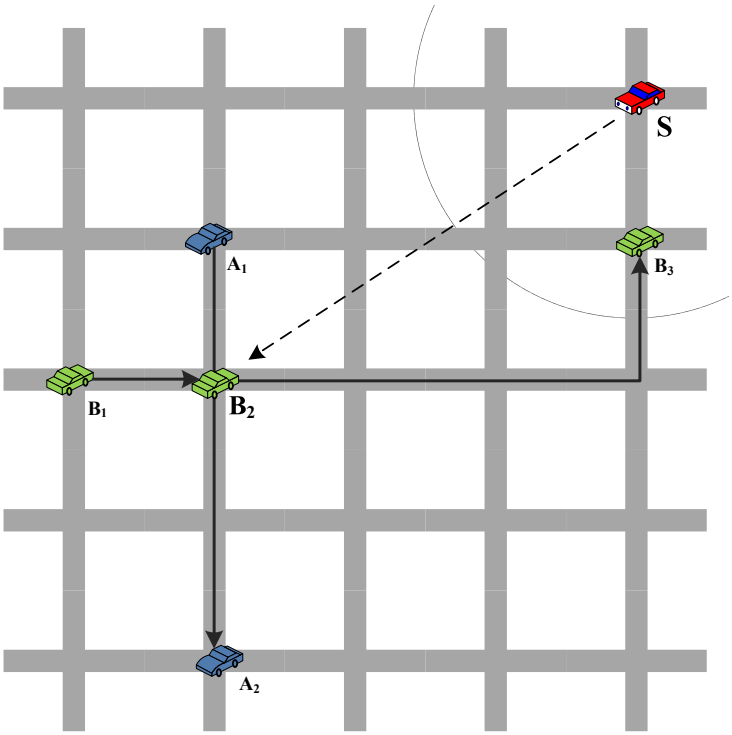


Figure 2.11: An example of the routing path by EASE

2.5.2 Carry-and-forward Routing

Carry-and-forward is a new concept proposed in [67]. The idea is as the name suggests: when a routing path does not exist for a packet, the last receiver can carry the packet, and forward the packet to the new receiver until some conditions meet.

The first vehicular protocol adopting the concept is *Vehicle-Assisted Data Delivery (VADD)* [50]. A moving vehicle in VADD carries a packet and forwards it to the next vehicle in the intersection of roads. In the order word, the routing paths in VADD are the exact shape of the roads. Moreover, VADD predicts the mobility of other vehicles, which follows the traffic pattern and road layout. A routing decision is based on the result of such prediction. The experimented routing decisions are based on location (*L-VADD*), direction (*D-VADD*), multipath direction (*MD-VADD*) and hybrid (*H-VADD*). The H-VADD protocol has much better performance and it can avoid the routing-loop problem.

Geographical Opportunistic Routing (GeOpps) [51] is another carry-and-forward, where requires navigation information of other vehicles to predicts the mobility of other vehicles. By knowing the navigation information, the node in GeOpps knows the paths of other vehicles when it tries to forward a packet, then a decision can be made by comparing the nearest point of these path to the destination. For example, the vehicle S in Figure 2.12 tries to find a routing path to the gas station at D. Two vehicles, A and B, are in the radio range of S, and they will be driven from A_1 to A_3 and from B_1 to B_3, respectively. The nearest point of these two routing path is A_2, thus A becomes the new relay in the routing path. The GeOpps in theory may get a better result than VADD, but the navigation information may be mostly private in current or future VANETs.

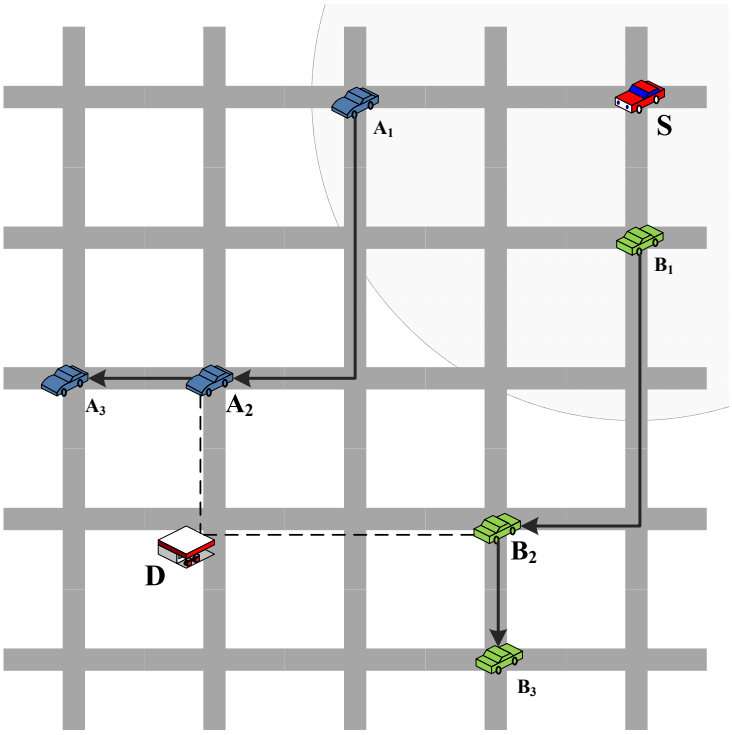


Figure 2.12: An example of the routing path by GeOpps

2.6 Geographic Routing in VANET Projects

The VANET relating projects concerning inter-vehicle routing protocols have been launched in *FleetNet* and *CarTALK2000* (Europe), *Cooperative Driving* (Japan) and *Vehicle Infrastructure Integration* (USA).

The IVC of FleetNet (2000-2003) [68] and CarTALK2000 (2001-2004) [69] [70] are based on UTRA-TDD (UMTS Terrestrial Radio Access Time Division Duplex). The UTRA-TDD is a third generation mobile telephone technology. It has about 1 km radio range, and 384 Kbps to 2 Mbps bandwidth according to the vehicle speed. It operates in the free frequency band from 2.010 GHz to 2.020 GHz. The IVC based on the development of UMTS technology can minimize the cost of access medium, and guaranty the full compatibility with the 3G mobile phone.

In Japan, the cooperative driving project (1993-2000) [71] [72] is started by JSK (Association of Electronic Technology for Automobile Traffic and Driving). It utilizes 5.8 GHz DSRC (Dedicated Short-Range Communication) for transmitting data, and it employs DGPS for measuring vehicle location [73]. The DSRC is adapted to the applications of highway infrastructures management such as ETC (Electronic Toll Collection) and vehicle counting, but it may not be appropriate for general IVC applications such as security and Internet access. This project develops a short-range cooperative communication protocol named DOLPHIN (Dedicated Omni-purpose inter-vehicle communication Linkage Protocol for HIghway automatioN) [72].

In USA, the leading project is called *Vehicle Infrastructure Integration* (VII, 2004-2010) [74] [75]. This project is focused on improving safety and roadway management. Its communications involve vehicle-to-infrastructure and vehicle-to-vehicle by using 5.9 GHz DSRC. In addition to VII project, there are researches at VTTI (Virginia Tech Transportation Institute, USA) trying to provide communication solution in high mobility scenarios by using the low-cost WLAN (Wireless Local Area Network) technologies such as IEEE802.11b [76].

The previous projects provide cooperative IVC solutions either on the limited application domains, or with the fixed wireless techniques. Their research efforts are more focused on the general-purpose Internet access with lower real-time constraint, mobility and reliability. The CIVIC protocol gives emphasis to adaptability, and it puts more effort to design robust auto-configured routing mechanisms.

Chapter 3

Communication System: Designs and Evaluations

3.1 Overview

The communication system is based on four layers and a task center. The stack architecture is as shown in Figure 3.1.

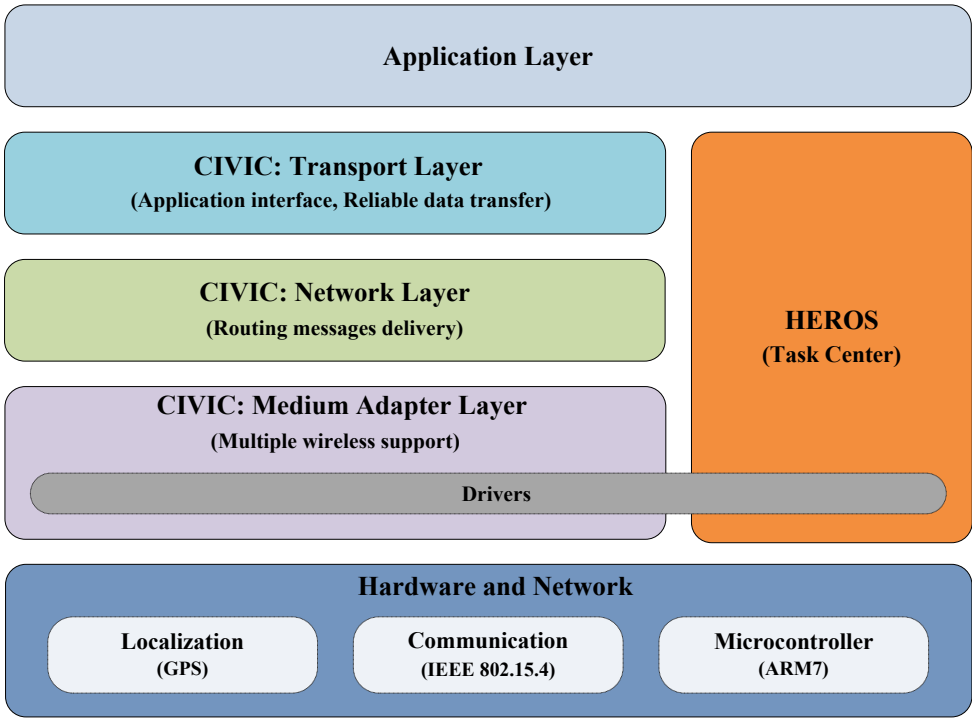


Figure 3.1: Stack Architecture of the Communication System

The works of this thesis involve two major components in the communication system: the CIVIC routing protocol and the HEROS embedded operating system. The former is a quick reaction and low resource consumption protocol for inter-node message transmissions; and the latter controls the system and assures intra-node resource aware IPC (Inter-Process Communication). The functions of the CIVIC protocol handles the three communication layers

includes the transport layer, network layer, and medium adaptation layer. The HEROS locates at the task center, and it controls the communication tasks.

The CIVIC protocol is the focus of the thesis. The motivation for designing CIVIC protocol has been introduced in Chapter 1. This chapter will describe the concepts and features, then the message delivery mechanisms and general implementations. The related works and real-world implementations and experiments on IVC fields are in Chapter 4. Besides, although the CIVIC protocol is originally designed for VANETs, its applications are not limited to it. The WSN applications of the CIVIC protocol are introduced in Chapter 5.

Next to the section of the CIVIC protocol (Section 3.2), it will be the HEROS section (Section 3.3). It is the only section about embedded operating system in this thesis, thus the related works, operating mechanisms, and evaluation result will all be given inside the section.

Then the hardware aspect (Section 3.4) and the network aspect (Section 3.5) of the communication system follow respectively. In the end of the chapter, all the major factors that cause the message delay and loss rate of the communication system will be evaluated. There are two groups of factors: the factors relating to the communication system itself (Section 3.6), and the factors from the outside disturbances (Section 3.7).

In the end, the simulation results on the *Shawn* simulator is given (Section 3.8).

3.2 CIVIC Protocol

3.2.1 Concepts and Features

3.2.1.1 Infrastructure Supports

The design of the CIVIC protocol is based on the scenarios of vehicular networks with dramatic changes of topologies according to location and time. In some scenarios, for example at night and on bad weather, the network density could get very low. In such scenarios, a communication system purely in client/server mode or in mobile ad-hoc mode may not be appropriate.

Since the distribution of vehicular network is generally along roads. The CIVIC assumes the roadside infrastructure *MMRS (Multi-support, Multi-service Routers and Servers)* can be deployed to support network access and QoS. The main functions of the MMRS are:

- Ensure the network connection to the nearby nodes
- Send a private message to a given node
- Send an alarm message to all nodes
- Forward message(s) to other MMRS

The MMRS should be connected with wired networks; otherwise the network connectivity cannot be assured. Each MMRS maintains at least two message queues. The first one stores alarm messages, and the second one holds private messages. When a mobile node reaches an MMRS, this node broadcasts a request message including its VID (Vehicle IDentifier). If an alarm and/or private messages exist, the MMRS sends these messages back to the mobile node. The private message for this mobile node will then be deleted from the private message queue.

The Figure 3.2 shows how a message is forwarded from one node to another through mixed networking of ad-hoc and infrastructure.

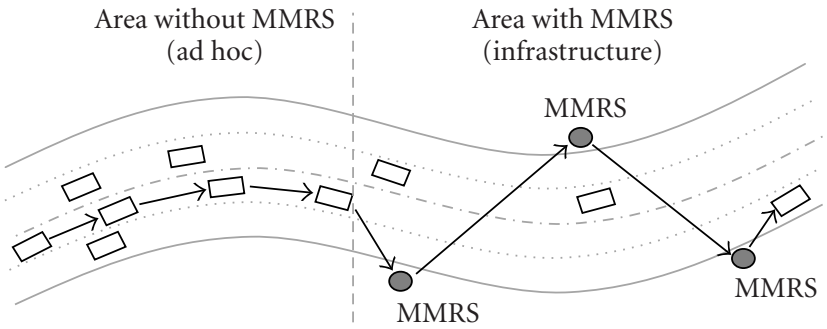


Figure 3.2: Mixed ad-hoc and infrastructure networks

3.2.1.2 Context Based Communications

The CIVIC protocol considers the communication contexts in a vehicular network, which are quite different from the ones in general-purpose MANETs. The nodes in a vehicular network are generally distributed along roads with directional movements, but in the MANETs with general purposes, nodes are grouped around an access point with random movements. By considering the contexts of distribution and direction, the CIVIC protocol can determine how to transmit messages (ad-hoc or infrastructure, interval of sending messages, etc.). There are two major communication contexts of a IVC:

The first one is the distribution. It can be used to estimate the bandwidth utilization and the capacity of a vehicular network. As previously mentioned, when a mobile node passes an MMRS, it sends a request message including its VID. The message may also contain its position, so the MMRS can estimate the distribution of vehicular network, and then a better knowledge concerning real-time traffic state will be obtained.

The direction of nodes is another major context of IVC. By introducing the direction, the PDR (Packet Delivery Ratio) and delay can be improved significantly [77]. Although the Euclidean direction is not appropriate for defining the direction of mobile node when roads are too winding, it may be applied for a short segment of a road. The CIVIC protocol assumes that all mobile nodes have an itinerary, and they move in a known environment. Thus, the direction of a mobile node is updated between two MMRS based on three factors: the itinerary and position of the mobile node, and the road map.

Moreover, since CIVIC protocol is based on the location-related context, a low-cost localization solution is a key element of our communication system. It is well known that the GPS is inappropriate in big city, particularly in EU where the roads are narrow and crossroads are much closed to each other. To obtain the correct position of a node for CIVIC protocol, mobile nodes and/or MMRS will use the LCD-GPS (Low Cost Differential GPS) implemented road maps. The LCD-GPS is a localization solution proposed by Kara [78]. This solution is able to improve the standard civil GPS accuracy even in dense urban area and it can be used for mobile tracking. More details about the LCD-GPS system are given in Section 3.4.4.

3.2.1.3 One-hop Link Stability

A common way to ensure quick routing response is to keep stable connections. In a high mobility scenario like vehicular network, the survival time of stable connections has great impact on QoS.

The stability of connection in CIVIC protocol is maintained by the neighbor knowledge exploration. The exploration is proactive, it is implemented by the exchange of "Hello" messages, and it must be performed only when the link stability is out of date. The dynamic interval of neighbor knowledge exploration is evaluated by $\Delta t = Min\{\Delta t_r\}$ with Eqs. 3.1:

$$\begin{cases} \Delta t_r = \infty, \ if \ v_r^{max} = v_s \\ \Delta t_r = \dfrac{R + x_s - x_r^{max}}{v_r^{max} - v_s}, \ if \ x_r^{max} > x_s \ and \ v_r^{max} > v_s \\ \Delta t_r = \dfrac{R + x_r^{max} - x_s}{v_s - v_r^{max}}, \ if \ x_r^{max} < x_s \ and \ v_r^{max} < v_s \\ \Delta t_r = \dfrac{x_r^{max} - x_s}{v_s - v_r^{max}}, \ otherwise. \end{cases} \tag{3.1}$$

where

R	:	the radio range in the worst case
x_s	:	the location of source node
v_s	:	the average speed
x_r^{max}	:	the location of one of its neighbor nodes
v_r^{max}	:	the speed of this neighbor node

Both x_r^{max} and v_r^{max} in Eqs. 3.1 are adjusted by the worst case of GPS error. The Eqs. 3.1 means that the interval of sending "Hello" messages depends on the distances and the relative speeds between the source node and its neighbor nodes.

After neighbor knowledge explorations, each node stores its neighbor information for the further multi-hop routing algorithm.

3.2.1.4 Multi-hop DANKAB

Because the CIVIC communication is based on broadcast, it is important to determine a correct broadcasting technique. The DANKAB (Directional Area Neighbour Knowledge Adaptive Broadcast) is therefore proposed when transmitting message by multi-hop. To use DANKAB, every node must obtain the location knowledge of a destination node and the neighbor nodes in the omnidirectional radio range (or, in the direction to the destination node if using an uni-directional antenna). In case of one-hop message sending, one-hop broadcast is used.

When the destination node is not in one-hop distance, DANKAB is used in the routing requests to find the next hop of source node. Figure 6 illustrates this process with source node S, destination node D, and routing node R. We define the direction area as an angle α with a default value of $\pm 30^{\circ}$. In order to reduce the number of messages in the network, only the nodes within the direction area can broadcast the message. If there is no node within the direction area, the angle α will be gradually increased (e.g. 45˚, 90˚ and 180˚) until the next hop is found. A

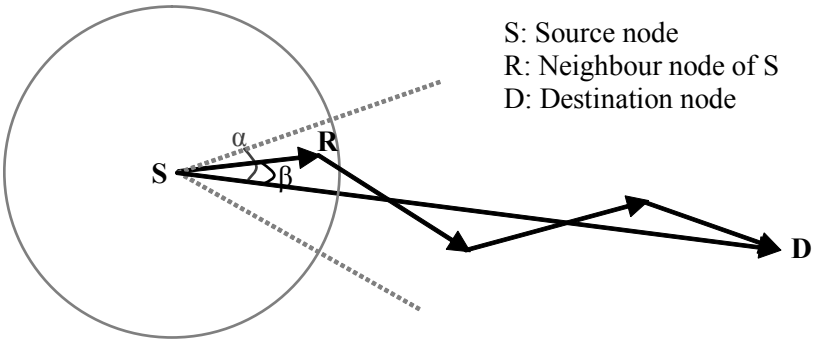


Figure 3.3: DANKAB routing concept

node can be a candidate in the next hop if $\cos\alpha \leqslant \cos\beta$. The $\cos\beta$ is calculated by law of cosines in Eq. 3.2:

$$\cos\beta = \frac{Dis_{sd}^2 + Dis_{sr}^2 - Dis_{rd}^2}{2Dis_{sd}Dis_{sr}} \tag{3.2}$$

where

Dis_{sd} : the Euclidian distances between nodes S and D
Dis_{sr} : the Euclidian distances between nodes S and R
Dis_{rd} : the Euclidian distances between nodes R and D

The destination location is a key element of CIVIC multi-hop routing approaches based on DANKAB. There are three ways that a source node can obtain the destination location:

- In an infrastructure network, the roadside MMRS can provide the location of destination node D.
- In an ad-hoc network, a location request will be performed by simple flooding to all directions.
- Because CIVIC protocol uses the directional broadcast by DANKAB, the nodes in/nearby a routing path will also receive the routing message even it is not sent to them or forwarded by them. Thus, the nodes can record this type of routing paths.

The location of destination node may change during this process, but the DANKAB is based on broadcast, so there is no need for a very accurate location of destination node.

When there is more than one node in the direction area, two energy-aware methods can be adopted for selecting the next candidate node:

- The first method is competitive broadcast. When a node in area α forwards (rebroadcasts) a routing message, it sends with a delay based on the remaining energy, thus the node with more energy will forward a message more quickly. Other nodes with less energy will discard the same routing message when they receive the first forward one.
- The second method is to let the source node S selecting a node from neighbor nodes to send/forward a routing message, and the other neighbor nodes except the chosen one will discard the routing message. It requires the additional information about the remaining resource (e.g. energy and memory) in neighbor knowledge explorations, but it generates much less routing data. We use the second approach for the implementation in this thesis.

After defining the next hop of source node S, the processes of DANKAB repeat hop-by-hop until the routing message attains the destination node or reaches the preset limitation of hop number.

If the routing path has been obtained, the data from application layer will be transmitted. If the data rate is low, DANKAB can also be integrated to the data sending, and the routing request can be ignored. For the implementation in this thesis, the two mechanisms are separated.

3.2.2 Layer-based Message Delivery

This section focuses on the layer-based message delivery mechanisms of the CIVIC protocol. Some issues about implementations will also be briefly addressed. The Table 3.1 shows three groups of essential messages in CIVIC protocol.

Message Types	**Message Names**	**Network Type**	**Size(Min.)**	**Size(Max.)**
Application	*DATA_SEND*	Unicast	15	70 *
	DATA_ACK	Unicast	14	20
Hello	*HELLO_REQ*	Broadcast	27	33
	HELLO_RPY	Multicast	39	53
Routing	*ROUTE_REQ_SF*	Broadcast	12	37
	ROUTE_REQ_DNB	Unicast	31	37
	ROUTE_RPY_DNB	Unicast	31	56
	ROUTE_RPY_PATH	Unicast	11	36

* The maximum size of *DATA_SEND* is configurable.

Table 3.1: CIVIC message groups, network types, and message sizes (bytes)

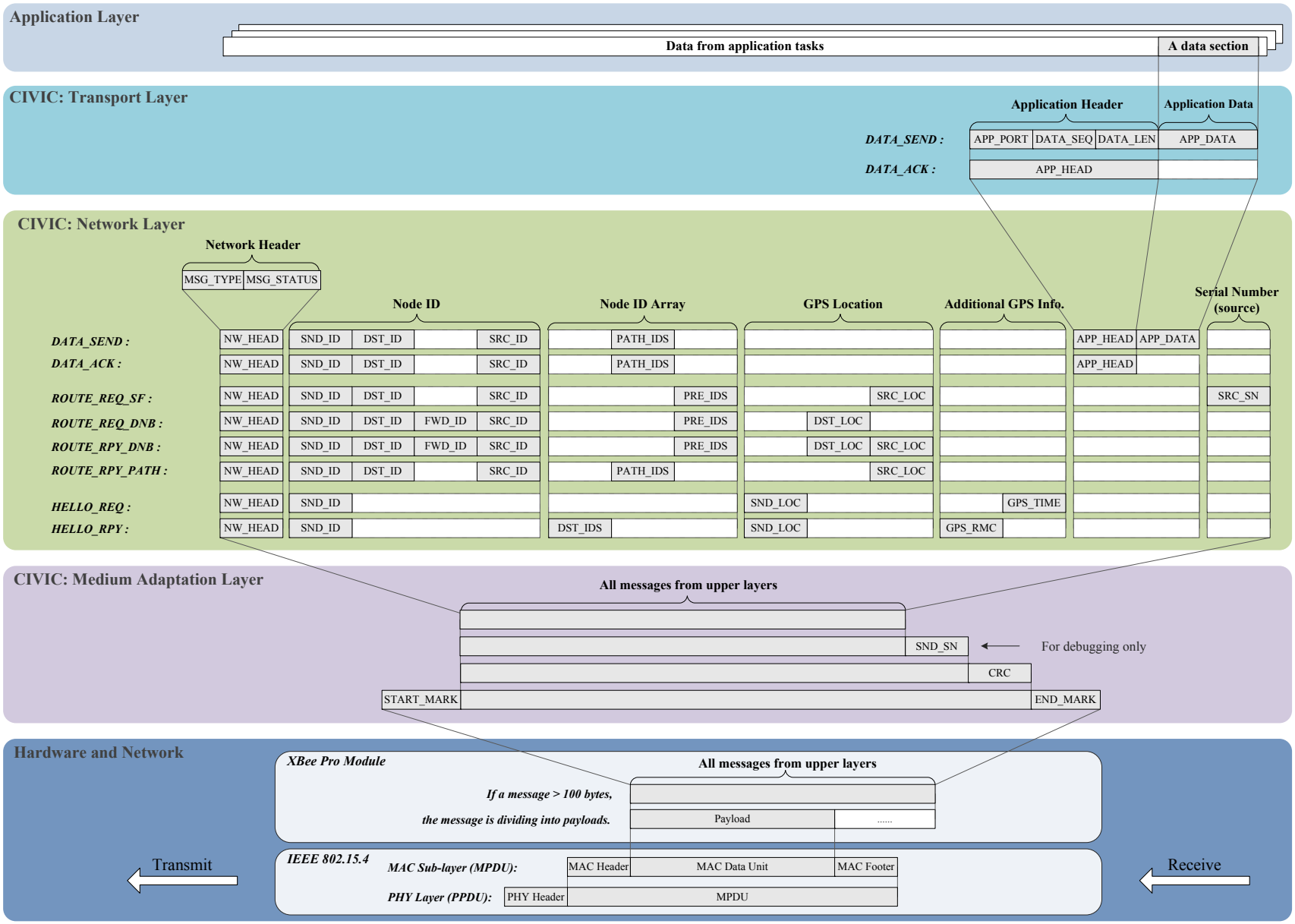


Figure 3.4: CIVIC messages on layer-based delivery

The Figure 3.4 shows how the messages pack/unpack between layers, and the main description is in the next subsections, and the description about the XBee Pro module and the IEEE 802.15.4 standard is in Section 3.6. Besides, another view in table format is in Appendix A.

The sender and receiver in CIVIC protocol talk with each other based on the preset message formats. The network header and application header are for identifying message types, and talking the existence and size of some dynamic contents such as application data, node ID arrays, and GPS location. The design in the message format has only one principle: the message sizes is as short as possible.

3.2.2.1 Transport Layer: Application Messages

The data from application layer is packed into the *DATA_SEND* messages by the transport layer. After at least one routing path has been found by the network layer, the data will be sending to the destination node. The transport layer in current CIVIC protocol is a simple one, and it performs three major tasks:

1. It *transports data* to and from the correct application tasks, as the basic function of a transport layer.
2. It is implemented with the *acknowledgement mechanism* for minimizing the influence of network congestion and other errors.
3. It provides an *routing interface* for controlling the networking actions in runtime. The controllable actions include defining the routing direction, switching between proactive and reactive request, setting the routing task intervals, disabling/enabling acknowledgement mechanisms, and disabling/enabling application tasks.

The task of *transporting data* is done by add/identify the application header for the data of application tasks. The application header contain three message fields as shown in Figure 3.4, and each takes one octet to transmit. These message fields are *APP_PORT*, *APP_SEQ* and *DATA_LEN*, which indicate the identity of an application task, the sequence number of a application data from the task, and the size of the application data, respectively. The field after the application header is the *APP_DATA*, which is normally a section of data content. The *APP_DATA* has a changeable size from one byte to a predefined maximum size. If the data from application tasks is larger than the maximum size, the data will be divided into multiple messages and then sent. The predefined maximum size is configurable by users, and the maximum size of *DATA_ACK* in Table 3.1 is for our experiment implementations only.

The design of the *acknowledgement mechanism* is for reliable communication. It is well known that a broadcast-based communication can cause collisions, thus the CIVIC protocol has the additional mechanism in transport layer to minimize such influences. To assure a *DATA_SEND* reach the destination node, a source node can ask the destination node to send back an acknowledgement message, named *DATA_ACK*, which only contains the application header from a *DATA_SEND*. If the source node does not receive this acknowledgement in a limit of time, it can thus choose further actions.

For the implementation of the *routing interface* of transport layer, before the sending each *DATA_SEND*, the transport layer checks several key parameters representing the sub-tasks

of application interface. Here we only explain the switching of proactive/reactive routing request as an example. In the network layer of the CIVIC protocol, a system real-time task keeps on sending proactive routing request for all application tasks based on the configuration of application layer and the status of destination nodes. The reactive routing request is implemented as a proactive routing request that is sent on-demand immediately. If no routing path is found, the routing request continues at intervals, and the repeat period is set by application tasks through the application interface. If at least a routing path is found, the sending of *DATA_SEND* follows.

3.2.2.2 Network Layer: Hello Messages

The network layer is the major component layer of the CIVIC protocol for routing messages. There are two groups of messages implemented in the CIVIC network layer as shown in Table 3.1.

The first group of messages is the implementation of neighbor knowledge exploration for the one-hop link stability in Section 3.2.1.3. It is a proactive routing approach based on the communication contexts including locations, directions and speeds. This type includes *HELLO_REQ* (hello request) and *HELLO_RPY* (hello reply) messages. The *HELLO_REQ* is a one-hop broadcast message to neighbor nodes asking to build stable links. The calculation of sending intervals is by Eq. 3.1. The *HELLO_RPY* is a one-hop multicast reply to the *HELLO_REQ*, which contains the information of location, direction and speed of the replier node.

The *HELLO_RPY* in our experiment implementations is the only message sent by one-hop multicast. If the neighbor nodes react to the *HELLO_REQ* from a source node one-to-one immediately, all replies will reach the source node approximately at the same time, and a message jam happens. There are some methods that can be used to prevent such issue (e.g. random delay, priority-based reply), and our implementation has been using a real-time task to control the *HELLO_RPY*. Because the replies are actually controlled by neighbors themselves instead of the sending action of source node, the message jam is minimized. Because a *HELLO_RPY* is sent out to a group of neighbors that require for building one-hop stable links, the *HELLO_RPY* is multicast, and therefore the sending frequency of *HELLO_RPY* is reduced.

3.2.2.3 Network Layer: Routing Messages

If a destination node is in one-hop distance, there is no need to send a routing request; if not, multi-hop routing request and reply performed by the SF or DANKAB as the second group of messages in Section 3.2.1.4. This group contains four types of messages: *ROUTE_REQ_SF* (the routing request by the simple flooding approach), *ROUTE_REQ_DNB* (the routing request by the DANKAB approach), *ROUTE_RPY_DNB* (the routing reply by the DANKAB approach), *ROUTE_RPY_PATH* (the routing reply through a found path):

A routing request must be echoed by a routing reply if the destination node receives the requirement. The availabilities of the destination location (*DST_LOC*) and the location of source node (*SRC_LOC*) decide which pair of request/reply will be chosen. Figure 3.5 shows a routing request process. The routing request process is normally driven by a real-time task at

intervals (proactive), but they can also be reactive if the application layer asks for it.

The reaction to a routing request will be: a routing reply, or a routing forward, or/and table updates. A routing reply process is shown in Figure 3.6. It is normally driven by an event task for the message arriving. A routing reply follows by two reactions in the next hop: a routing forward, or/and table updates.

Because CIVIC protocol also works in ad-hoc mode, the SF technique is integrated into the routing approaches of CIVIC. Therefore, the protocol can still perform the application tasks when there is no location obtained by GPS. A request/reply pair of *ROUTE_REQ_SF* and *ROUTE_RPY_PATH* is designed for this purpose, but it is called the worst request/reply pair because a *ROUTE_REQ_SF* messages request all nodes to be involved into a routing request.

The *ROUTE_REQ_SF* message is sent when the destination location is unknown. If a *ROUTE_REQ_SF* message does not contain the location of a source node, a *ROUTE_RPY_PATH* message is sent back by a destination node to the source node; otherwise, the routing reply will be performed by DANKAB approaches to the source location (*ROUTE_RPY_DNB*). Both routing replies can contain the location of the destination node if it is available, so that the next routing request from the source node can use DANKAB (*ROUTE_REQ_DNB*). The *ROUTE_REQ_DNB* message is sent when a destination location is known, and it is normally replied by a *ROUTE_RPY_PATH* message.

If an ideal network (zero delay and error rate) is working in a pure ad-hoc mode without any location system, the worst request/reply pair will be continuing; but if the location system is available for all network nodes, the situation is different: 1) If destination nodes are fixed locations and source nodes are mobile, most of the routing messages in a network are sent as the best request/reply pair of *ROUTE_REQ_DNB* and *ROUTE_RPY_PATH*, except the routing messages in the first round. It is the best request/reply pair because both *ROUTE_REQ_DNB* and *ROUTE_RPY_PATH* are unicast, and the latter one does not need the amount of calculations like using *ROUTE_RPY_DNB*. 2) If both destination nodes and source nodes are mobile, the best request/reply pair can still perform the routing messages if it is an ideal zero-error network. However, if it is in a practical real-world network, the selection of request/reply pairs is depending on the comparison of the mobility level of network nodes and the message delay (along with loss rate). If the message delay and loss rate are lower, the network nodes can still get a broadcasted message even they are not in the original locations; and therefore, the best request/reply pair can be performing in most routing messages. Or, an implementation can be fixed to use *ROUTE_RPY_DNB* instead of *ROUTE_RPY_PATH* in high-mobility networks. However, even in the latter method, to lower the message delay and loss rate is still a key target. Therefore, we will details these two parameters of our communication system in the Section 3.6 and 3.7.

In the following, the data structures in the Figure 3.5 and Figure 3.6 will be briefed, along with the C modules containing the data structures. More details about the implementation for experiments are in Section 4.1.

- *NeiTable*: The major module for the routing request and reply is *table_neighbor.c*, which contains the *NeiTable* table to store the information from one-hop neighbor knowledge exchanges. This module is also implemented with the functions to calculate the next hop by DANKAB, and the next interval of sending a Hello request, because these calculations are based on the information in *NeiTable*.

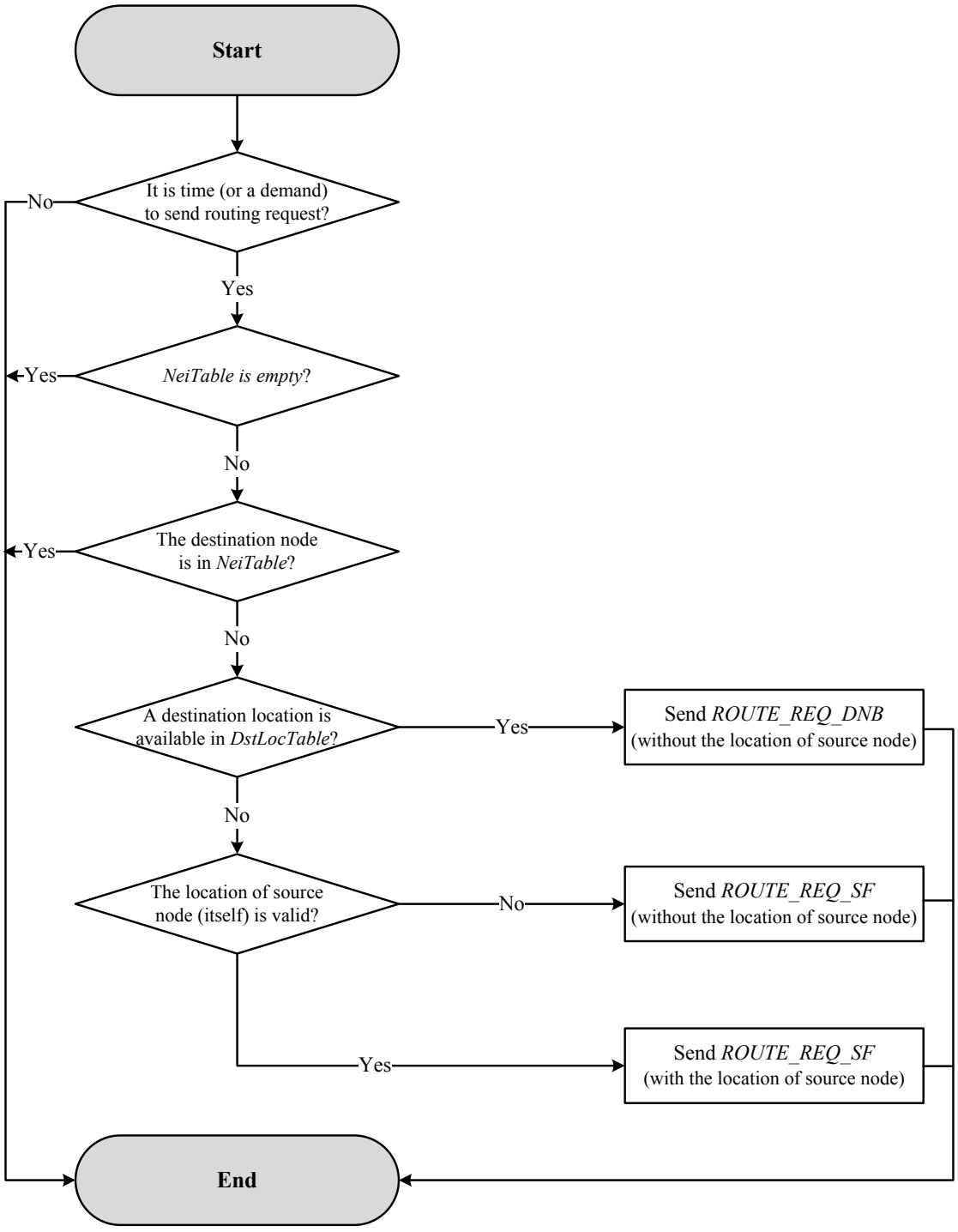


Figure 3.5: The proactive tasks related to the first sending of routing requests

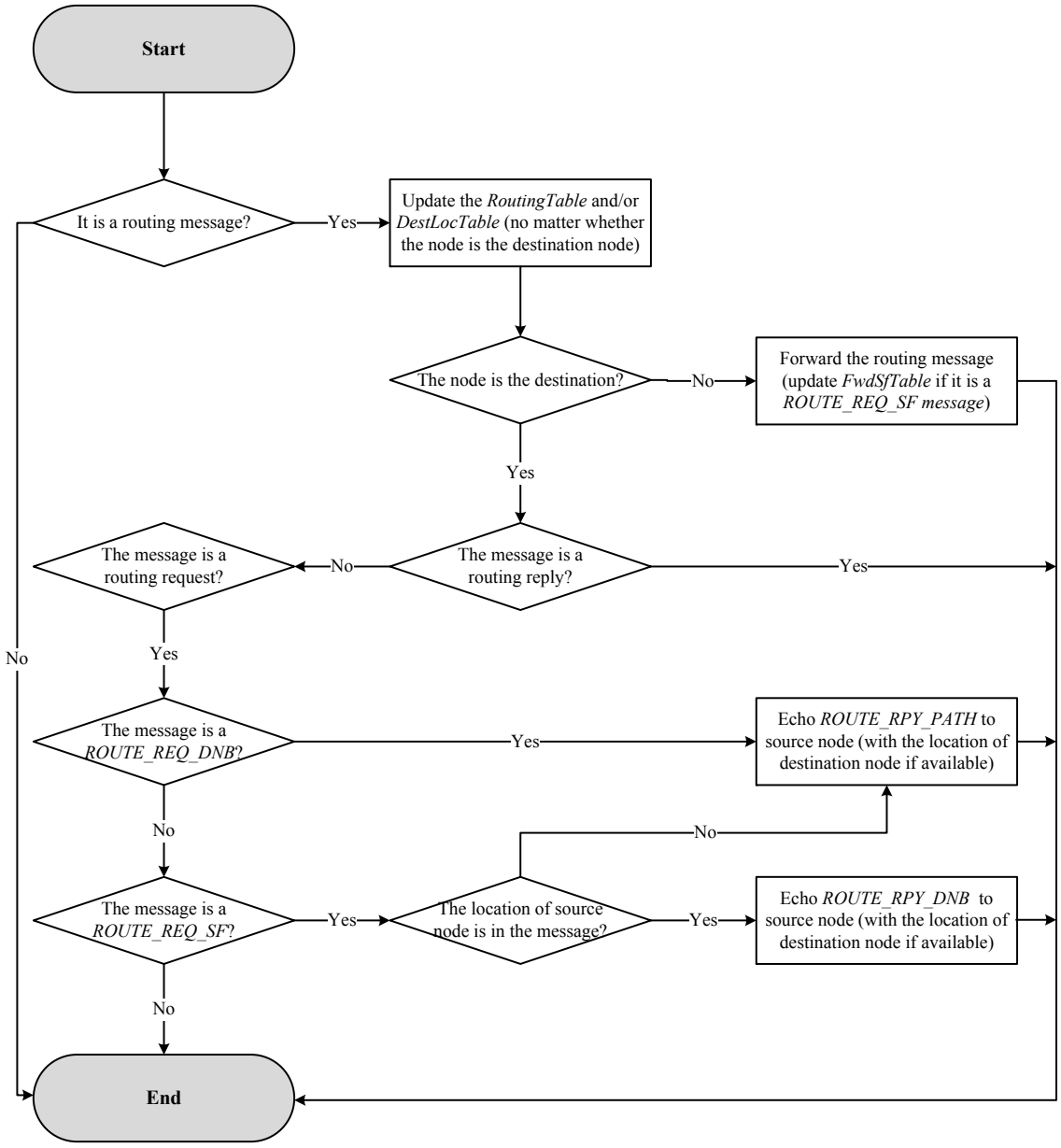


Figure 3.6: The reactive tasks related to routing requests and replies

- *FwdSfTable*: The *table_sf.c* with the *FwdSfTable* table is for avoiding the redundant *ROUTE_REQ_SF* messages, because the broadcast of *ROUTE_REQ_SF* messages will not select a node from neighbors to forward a message like DANKAB. The *FwdSfTable* records the serial number from source nodes (*SRC_SN*). If a redundant *ROUTE_REQ_SF* message has the same *SRC_SN*, the message will be discarded.
- *RoutingTable*: The *RoutingTable* is in *table_routing.c*. It keeps the routing paths from routing replies for application tasks to use. The interval to clear an outdated routing path is depending on the level of mobility of the network nodes.
- *DstLocTable*: The *table_dst_loc.c* with *DstLocTable* is for two usages: storing the IDs of destination nodes (*DST_ID*) if there are requests from application layer; storing the captured locations of destination nodes (*DST_LOC*) from routing messages if their IDs are in the table.

3.2.2.4 Network Layer: Message Fields

This subsection introduces the message fields which are added in the network layer as shown in Figure 3.4. The first two fields added in the network layer are the network header. The *MSG_TYPE* (one octet) indicates the type of a message. It works together with *MSG_STATUS* therefore the receiver nodes can recognize a message and the contents in messages. The *MSG_STATUS* (one octet) is the status of a message (e.g. whether the source location is ready, hop number, and path length) as shown in Table 3.2. Besides, the *MSG_STATUS* could also contain the debug information and the activation of testing functions.

After adding *MSG_STATUS*, the message fields will be attached by the network layer depending on the network header, which include: 1) ones of four node IDs as shown in Table 3.3; 2) one array of node IDs as shown in Table 3.4, which have dynamic sizes and the size is indicate in *MSG_STATUS*; 3) one type of the node locations from GPS to calculate the DANKAB angle and indicate the replying location (source location); 4) the application header and data; 5) the Table 3.6 shows the message fields that are not in the previous tables.

3.2.2.5 Medium Adaptation Layer: Software/Hardware Interface

The medium adaptation layer (including hardware drivers) works between the hardware of a sensor node (i.e. the LiveNode if it is for our experiments) and the network layer of the CIVIC protocol (or HEROS in task center), thus this layer will also be introduced in the hardware section (Section 3.4.2). This subsection only focuses on software aspect: the software interface and the message delivery.

Comparing with the regular hardware drivers, the layer has two additional functions:

- From the sensor hardware to the higher layers, a major software task of the medium adaptation layer is to convert the meaningless characters from various hardware drivers to the meaningful information (e.g. GPS location and time, sensor data) or messages (e.g. the messages in previous section) with unified *Get()-like* interfaces.
- From the higher layers to the sensor hardware, this layer provides unified *Set()-like* interfaces, and take care of the rest (e.g. enabling the maximum low-level sending intervals, handling the receiving/sending if overrunning, and maintaining the unsent data).

Masks	Values	Descriptions
10000000	0 or 1	For *HELLO_RPY* message, it equals to one if the location of sender node (*SND_LOC*) is valid.
10000000	0 or 1	For all routing messages, it equals to one if the location of source node (*SRC_LOC*) is valid.
10000000	0 or 1	For a *DATA_SEND* message, it equals to one if a (*DATA_ACK*) is required. The network layer of a receiver can directly reply the *DATA_ACK* message, and there is no need to really implement the acknowledgment mechanism in transport or application layer.
01000000	0 or 1	For all routing messages, it equals to one if a routing path is discovered by DANKAB, it equals to zero if by SF.
00001111	1 to 15	For *HELLO_RPY* message, it indicates the number of multicast destination nodes in the field of *DST_IDS*.
00111000	1 to 7	For routing and application messages with *PATH_IDS*, it represents the number of IDs in a found routing path recorded in *PATH_IDS*.
00000111	1 to 7	For routing messages with *PRE_IDS* ((*ROUTE_REQ_SF*, *ROUTE_REQ_DNB* and *ROUTE_RPY_DNB*)), it records the number of IDs in a previous passing path recorded in *PRE_IDS*.

Table 3.2: Binary masks, values and descriptions for *MSG_STATUS*

Field Names	Size (octet)	Descriptions
SND_ID	1	The ID of the last sender (forwarder) node.
DST_ID	1	The ID of a destination node.
FWD_ID	1	The ID of the next node that is selected by DANKAB to forward an unicast routing message.
SRC_ID	1	The ID of source node (the first sender node).

Table 3.3: Message fields relating to the node ID in a network message

Field Names	Size (octet)	Descriptions
DST_IDS	1 to 15	This field is for the multicasted *HELLO_RPY* only. It contains the IDs of one-hop destination nodes.
PATH_IDS	1 to 7	When a message is sent by a found routing path, it must contain this field. It indicates the path will be used to forward the message.
PRE_IDS	1 to 7	It is for a routing searching message (*ROUTE_REQ_SF*, *ROUTE_REQ_DNB* or *ROUTE_RPY_DNB*) to attached a found node. When this type of message is passed to a network node, it is added with the ID of this node to the end of the ID array.

Table 3.4: Message fields relating to the array of node IDs in a network message

Field Names	Size (octet)	Descriptions
SND_LOC	19	The location of sender node for one-hop *HELLO_RPY* only
DST_LOC	19	The location of the destination node with *DST_ID*.
SRC_LOC	19	The location of source node with *SRC_ID*.

Table 3.5: Message fields relating to the location from GPS in a network message

Field Names	Size (octet)	Descriptions
GPS_TIME	6	The UTC time in a six-octet HHMMSS format (hour:minute:second) from GPS. It is only deployed in *HELLO_REQ* messages, and it is for the monitoring software to use as a time reference in debugging mode.
GPS_RMC	11	The direction and speed of a neighbor node in *HELLO_RPY*. This field is from the "RMC-Recommended Minimum Specific GNSS Data" from GPS [79].
SRC_SN	1	The message serial number of a source node. It is for *ROUTE_REQ_SF* only.

Table 3.6: Message fields for only one type of network messages

For the message delivery in a sender node, when sending messages from network layer, an additional field, named *SND_SN*, is attached to all messages. It is a serial number (one octet) to indicate the place of a message in the sender's output sequence. It is different from the *SRC_SN* in the last section. The error rate in the following sections or chapters is calculated by the missing serial number.

After attaching *SND_SN* (one octet), the layer will attach a CRC (Cyclic Redundancy Check) for all messages to assure the messages sent and received correctly. In the end, this layer adds starting mark (one octet) and ending mark (one octet), named *START_MARK* and *END_MARK* to all messages. If a character in a message is as same as *START_MARK* and *END_MARK*, the character will be converted to other one (by adding *0x7D* like the methods in TCP/IP protocol).

In a receiver node, these mechanisms are done backwardly. Moreover, if there are the further functions to compress/decompress and encode/decode data, they should be implemented in the medium adaptation layer.

Besides, except the three message groups in Table 3.1, a group of management message is implemented to carry information between a station and network nodes for managing the network and feedback experiment results. This message group is not an essential part of the CIVIC protocol, thus only a brief is given: This group includes two types of messages: 1) a *MANAGE_REQ* message is sent from a station to network nodes; 2) a *MANAGE_ACK* message works backwardly to a station, and it will be discarded by network nodes if receive it. The mechanism of management messages is directly implemented on the medium adaptation layer, thus it can use the *START_MARK*, *CRC*, and *END_MARK* to assure the message correction. It also has a identity field *MSG_TYPE* after the *START_MARK* as all CIVIC messages, but there is no other limitation to the contents between *MSG_TYPE* and *END_MARK*. In our

implementation, these contents are mostly readable ASCII texts.

3.3 Integrating with HEROS

3.3.1 Overview

The simplest implementation of the CIVIC protocol is to schedule the CIVIC tasks as an infinite loop, and activate them non-preemptively by timer interrupts. Such implementation is suitable for some applications required to satisfy a strict memory limitation, but it cannot assure a higher priority real-time task to be run when the system is busy, and it is difficult to achieve the intra-node resource-awareness. Besides, the program codes implemented by this method are hard to be maintained and updated. To overcome these shortcomings but still meeting the low memory footprint requirement, the communication system in the thesis is integrated with an Embedded Operating System (EOS), named HEROS (Hybrid Event-driven and Real-time multitasking Operating System) [80]. The evaluation results of this section show that HEROS is suitable for our targeting low-aware high-dynamic networks in terms of memory consumption and system latency.

The design of HEROS is aimed towards a reliable lightweight EOS with good resource-awareness, maintainability and adaptability; more importantly, these features should be able to perform in low-cost mobile devices. To achieve these goals, the architecture of HEROS introduces the concept of a coordination language named Linda [81] [82] to design a two-level component-based microkernel. It is the key to assure the simplicity for implementing resource-aware embedded applications, and to enable a reliable concurrent/parallel processing mechanism. Based on the two-level components, the configurable hybrid microkernel of HEROS merges the advantages from event-driven and real-time multitasking operation mechanisms, thus HEROS requires a less resource consumption while supporting more variant embedded applications.

The following subsections first introduce the concept of Linda coordination language, then related works on existing event-driven and real-time multitasking EOSs. Next, we describe the key features of HEROS including the Linda-based component design and the hybrid microkernel. In the end, the evaluation result will be given.

3.3.2 Linda Mechanisms

This section will first brief the evolution of Linda in industry and research, then the Linda mechanisms relate to the design of HEROS will be introduced.

Linda language was proposed by David Gelernter [81] in 1985, and the terminology "coordination language" was introduced to Linda by Gelernter and Carriero [83] in 1992. In a not accurate description, Linda can also be seen as a coordination extension that can be added to nearly any programming language and platform. The Linda implementations can be found for many major programming languages, e.g. C, Java, Smalltalk and Lisp. The main commercial developments involve the Linda concept including Sun's JavaSpaces [84] and IBM's TSpaces [85]. A list of recent Linda-related research projects can be found in [86], and most of the projects relate to the middleware design in distributed computing, especially for web services applications. A current trend is to introduce Linda to the middleware design for mobile ad hoc networks, e.g. LIME [87] and TOTAM [88], which relate to HEROS distantly. Besides, some previous works introduce Linda to the design of a parallel computing interface

for Unix or Unix-like operating system [89]. To the best of our knowledge, HEROS is the only EOS that uses the Linda mechanisms in its design.

In a more accurate description, Linda is a machine model to coordinate the computations in parallel and distributed systems. The mechanism separates coordination from computation by using a logically shared memory called "*tuple space*" or *TS*. The tuple-based communication is asynchronous and anonymous. A message cannot be sent or received between two processes directly. Instead, a process sending a message outputs the message to TS anonymously, and a process wanting such message seeks and inputs it from TS. A sender process and a receiver process do not need to know the existence of each other. The design uncouples the spatial aspect in programming because a communication is not based on the identity of processes. The design also uncouples the temporal aspect because processes do not have to have overlapping lifetimes. However, a full time uncoupling is added with an assumption that a tuple will remain in TS forever until a receiver process get it or the program that generates the tuple needs to be terminated. This assumption may be reasonable for Internet web service applications, but it is questionable for current embedded applications.

A TS contains tuples produced by processes, and there are two types of tuples in a TS: the passive tuple contains data values; the executable tuple contains, incorporates, or activates program codes. The original Linda [81] defines three operation primitives to access a TS, including *out()*, *in()* and *read()*. The out() operation produces a tuple, writes it into TS, and the executing process continues immediately; in() reads and removes a tuple from TS, then the executing process continues; read() reads a tuple but not removes it. Both in() and read() in original Linda will be suspended until the required tuple is available.

Because tuples in a TS can be physically distributed on separate computers, the accessing to a tuple is not refereed by any physically memory address, but by a *structured name*. The structured name is a subset of the combination of contents in a tuple. Figure 3.7 demonstrates a successful out() and in() operations using the structured name {"Temperature", "2010-08-01"} to assign 30 to "min" and 33 to "max". The "Temperature" and "2010-08-01" are called actual parameters in this statement, and 30 and 33 are formal parameters. After the executing, the tuple {"Temperature", "2010-08-01", 30, 33} will be withdrawn by in().

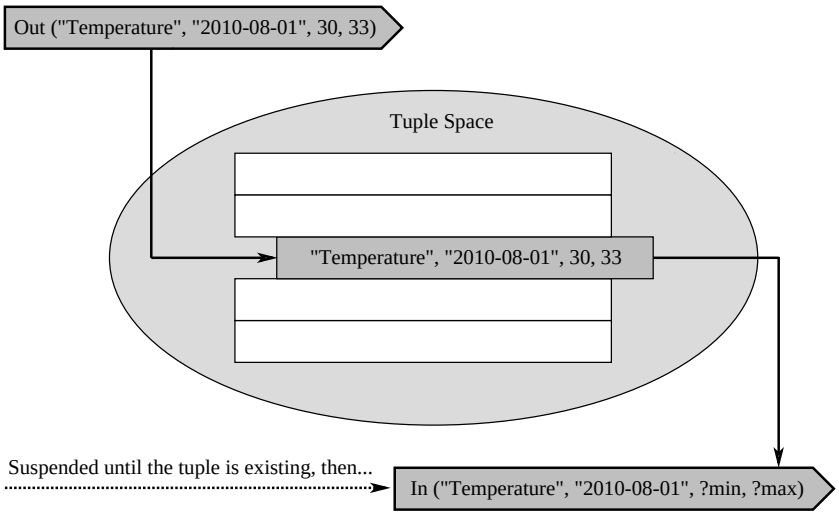


Figure 3.7: A tuple-based out() and in() operation by structured name

The advantages of using Linda are to allow a more orthogonal separation between coordination and computation, and a more general subsumption of various levels of processes. Moreover, the Linda mechanisms suit highly dynamic networks. It is not likely to use a traditional server/client model or a complex mobile ad-hoc model to coordinate the distributed computing tasks in a highly dynamic network but still maintain low resource consumption. Linda could offer the foundation of a better and simpler solution.

However, the original Linda mechanisms cannot be directly applied to HEROS, because:

- Some operations in original Linda concepts may require too much computation resources for embedded applications (e.g. structured name, undirected in() and read()).
- A full time-uncoupling is not adequate for real-time computing processes.
- If all processes and tuples and are anonymous, it is difficult to direct signal/message and build a true real-time multitasking mechanism.

Therefore, HEROS absorbs the essence of Linda, but develops its own mechanisms that are more suitable for the mobile embedded applications (the description starting from Section 3.3.4).

3.3.3 Related Works on EOSs

In the existing EOSs, there are two common operation mechanisms: real-time multitasking and event-driven.

The real-time multitasking mechanism provides a solution for rapidly developing the time-sensitive applications and it gives the full control over real-time tasks [90]. However, this mechanism consumes high resources in terms of energy, CPU and memory. The existing embedded OSs such as SDREAM [91], μC/OS-II, VxWorks, QNX, pSOS, WinCE.NET, RTLinux, Lynxos, RTX, and HyperKernel are not suitable for the resource-constraint mobile networks (e.g. WSNs) because they can only operate as this mechanism. Comparing with HEROS, they consume more resource in terms of CPU and memory.

To minimize resource consuming, many EOSs were developed for WSN fields (called WSNOS: WSN Operating System) such as TinyOS [92], Contiki [93], MagnetOS, MantisOS, EYEOS and SOS (Sensor Operating System) [94]. These WSNOSs meet the resource-aware requirement, e.g. TinyOS can perform an event-driven component-based operation with tiny memory footprint. The rest of WSNOSs (except Contiki) are based on multitasking mechanisms. Contiki is based on event-driven, which is similar with TinyOS, but it can be configured to run in a multitasking mode. Note that, on one hand, a single task event-driven system does not fit for hard real-time requirements. On the other hand, in an event-driven mechanism (e.g. TinyOS), the task switches is normally based on a non-preemptive event-loop. The event-driven mechanism is suitable for WSNs because of low resource consumption, but the existing event-driven embedded WSNOSs are essentially implemented by a single processing mechanism; therefore, they may not be suitable for the embedded applications requiring complex hard real-time operations.

HEROS evolves from *SDREAM (Super-small Distributed REAl-time Microkernel)*, a real-time EOS developed by our team, but the design of HEROS integrates event-driven mechanisms with real-time multitasking into a configurable hybrid microkernel. HEROS can

run in a pure hard real-time mode like SDREAM, or an event-driven mode like TinyOS but with a certain level of hard real-time supports. This design is able to adapt to more various embedded applications including intelligent transportation, health care, environment monitoring, etc.

3.3.4 Linda-based Component Designs

The architecture of HEROS introduces the essential concepts from Linda shown in Figure 3.8: the tuple-based communication with simpler in()/out() primitives. The component-based design and the concurrent/parallel processing mechanism build on these essential concepts.

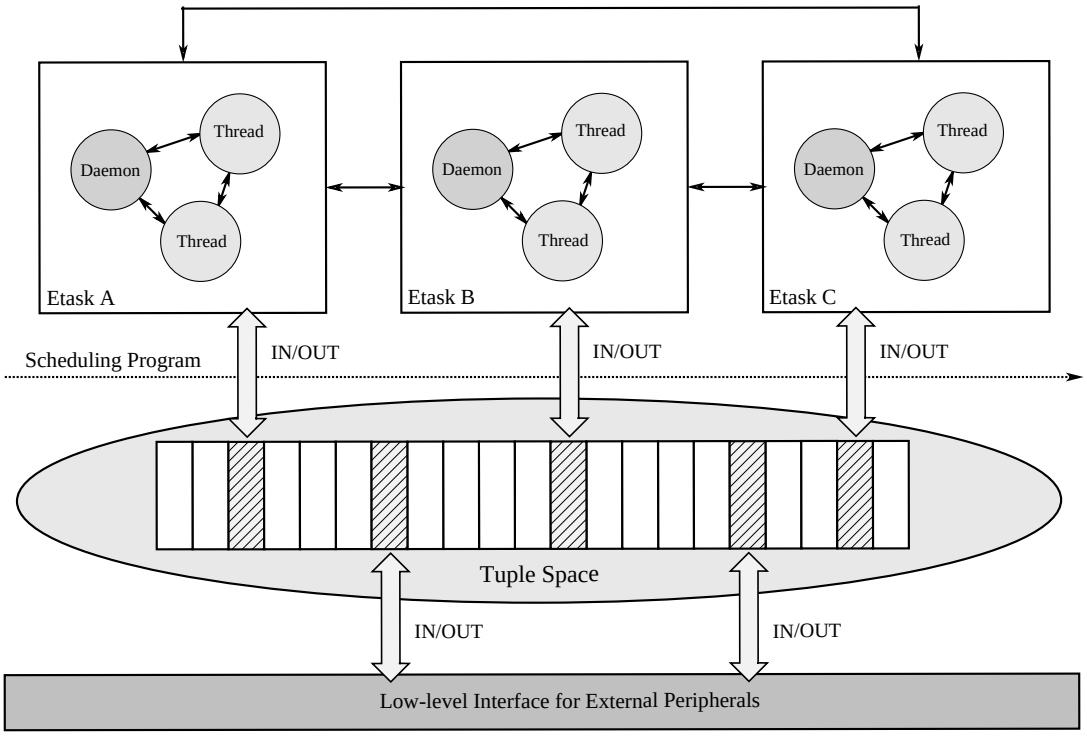


Figure 3.8: Tuple-based Component Architecture in HEROS

There are two sets of components in HEROS: *thread* and *etask*. *Tuple* along with *IN/OUT* primitives work as the component interface:

Thread is a low-level component that performs a single task in HEROS. A series of threads can be engaged in a complex real-time task under the control of an etask. Threads run in concurrent or in parallel, and they are preemptive based on the priority.

Etask is the high-level component that encapsulates a group of threads to complete a specific task, which is similar to the event concept of TinyOS. Etasks are performed in a sequence according to the priorities. Etask is not preemptive, but interruptible. It means that an etask with the highest priority will only be running after the current etask is finished, but the continuity of etask can be broken (e.g. an interrupt occurs) and resumed. When an etask finishes executing, it will enter a system thread named *daemon*, which enables hardware to be switched to a low-power mode (e.g. switch the ZigBee Pro module to the sleep mode for our experiments).

Tuples are contained in a mutual static TS, which will not be released until the program is terminated. Tuples are the only pipes for communicating messages and signals to/between

threads in HEROS. A thread must be ported to at least one static tuple. The static tuple is named *thd_tuple*. Besides, an optional type of tuple, named *tsk_tuple*, works for general etasks, e.g. the passive tuple in Linda concept, the temporary private resource for an etask. The *thd_tuple* and *tsk_tuple* have the same data structure. The *IN/OUT* primitives work similar to the in()/out() primitives in the original Linda concept. Besides the tuple-based advantages have been mentioned in Section 3.3.2, the unified TS in a single memory embedded system can help to prevent the memory fragmentation.

The advantages of the two-level component-based designs are to adopt more various applications with less computation overhead, memory footprint and implementation difficulty:

- For the computation overhead, considering an instance of HEROS as $\Re$, it contains a set of threads $T = \sum_{i=1}^{n} \tau_i = \{\tau_1 \parallel \tau_2 \parallel \tau_3 \parallel ... \parallel \tau_n\}$ where "$\parallel$" represents the concurrent or parallel operation. If $\Re$ is implemented by one etask with multiple threads, and the set of n threads are given with $\tau_i = (c_i, t_i)$ where c_i and t_i are the worst-case computation starting time and working period of thread τ_i, respectively. Let the utilization of τ_i be $u_i = c_i/t_i$, then the total utilization U will be given as Eq. 3.3 [95]:

$$U = \sum_{i=1}^{n} (c_i/t_i) \tag{3.3}$$

  If U is greater than the feasibly utilization bound of any scheduling algorithm, there is no guarantee for a hard real-time multitasking mode. The usage of event-driven etasks is to divide the threads into groups as $E = \sum_{i=1}^{m} \varepsilon_i = \{\varepsilon_1 \rightarrowtail \varepsilon_2 \rightarrowtail \varepsilon_3 \rightarrowtail ... \rightarrowtail \varepsilon_m\}$ where "$\rightarrowtail$" denotes the non-preemptive sequential operation. Because the utilization as Eq. 3.3 is limited to a subset of T within ε_i, the computation overhead is reduced.

- The memory footprint aspect refers to the collecting/releasing of private recourses by etask. When an etask is activated, it allocates private resources for sub-threads including context stacks, and optionally, additional *tsk_tuples*. After an etask is completed, these private resources will be free. This design allows embedded applications to be scheduled for more tasks with less memory footprint.

- For the implementation aspect, the two-level hybrid scheduling program can perform as an underlying cluster mechanism. The etask in HEROS is related to the event-driven implementation: it could be a packing widget that encapsulates a group of threads to complete the reactions to an event like TinyOS. But there is no limitation for the user program to do that, thus etask can just be used as a task group. Tasks can be divided into relatively independent task groups. It easies the application developer for implementing level-based tasks and managing program codes.

Besides, HEROS makes a trade-off between system latency and flexibility. To lower the system latency, the application tasks in HEROS are pre-configured. The components are created during the software initial stage, and there is no support for generating components in a runtime.

Before we start to get into the details of component designs, an overview for the activation process is given as follows: Tuples accept the signals and messages coming from threads or external peripherals by OUT operations. If the object of an OUT operation is a *thd_tuple*, the OUT operation activates the related thread. An etask is activated if one of sub-threads

is activated, then the etask starts to collect private resources. A two-level hybrid scheduling program executes or terminates etasks/threads, and maybe suspends threads, based on their priority conditions (in Section 3.3.5). An executed thread uses an IN operation to withdraw data from its relating and interesting tuples. The words "activate" and "trigger" in the following sections implicate the previous tuple-based activation processes.

The following subsections show the structures and behaviors of component described in the grammar of C language (by IAR C/C++ Compiler for ARM 4.40A [96]). The field names in capital letters means the constants assigned at the initialization stage of a program. The field names and their sequences are not the same as the ones in the actual codes of HEROS. The changes are only for easing the descriptions. The briefs for the functions of fields are listed as in the tables, and further descriptions are followed if necessary.

3.3.4.1 Etask

The group of etask components is called ECB (Etask Control Block). The data structure for etasks is as shown in Table 3.7.

Field Name	Data Type	Description
state	char	*Terminated*, *Sleep*, *Ready* or *Executing*
rest_time	unsigned long	Residual lifetime of the etask
next_etask	struct Etask *	Pointer to the next etask to be run after finishing the etask
thd_rdy	struct Thread *	Pointer to the next thread to be run
ETASK_ID	char	Etask ID
ORG_PRIORITY	unsigned short	Original priority of the etask
MAX_LIFETIME	unsigned long	Original lifetime of the etask

Table 3.7: The data structure for ECB nodes (struct Etask{...})

The etask component has four states including *Terminated*, *Sleep*, *Ready* or *Executing*. The explanation is in Figure 3.9 and the following list.

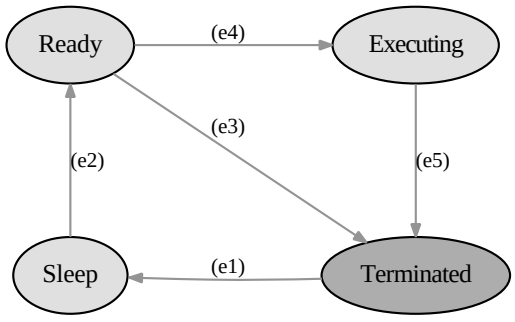


Figure 3.9: Transform of etask states

- *Terminated*: It is the initial stage of an etask. If an etask is triggered, but its sub-threads have not yet been given enough resources (e.g. stacks and private *tsk_tuples*) to run, the etask is keeping on this state. If an etask is called by the scheduling program in this state, the processes will seek and collect private resources, instead of executing the sub-thread.

- *Sleep*: It is a "transitional" state. Before an etask is *Ready*, it will be labeled to *Sleep* when the resource collections have been finished (as the edge "e1" in Figure 3.9). For the implementations with less etask, *Terminated* and *Sleep* can be combined to an *Idle* state where no resource will be released when an etask is terminated.
- *Ready*: An etask is switched to *Ready* when any of its sub-thread is triggered by a signal/message and the resource collections are finished (edge "e2"). Etasks are non-preemptive and scheduled by their priorities; therefore, an etask could remain *Ready* if another etask is keeping on running or another etask gains a higher priority in etask sequence. If the previous situations happen and the etask finishes its *MAX_LIFETIME*, the etask will be back to *Terminated* (edge "e3") and wait for a new triggering action.
- *Executing*: This state indicates that an etask is actually run by the scheduling program (edge "e4"). It will back to *Terminated* until all of its sub-threads are terminated (edge "e5").

There is a one-way link list representing the priorities of etasks, called EPL (Etask Priority Link), as shown in Figure 3.10. The *current_etask* in the head is a global pointer to identify the executing etask. The rest of links are connected by the *next_etask* in ECB nodes. The initial sequence of the link list is based on *ORG_PRIORITY*. The etask scheduling in the following sections implicate the resorting action to this link list.

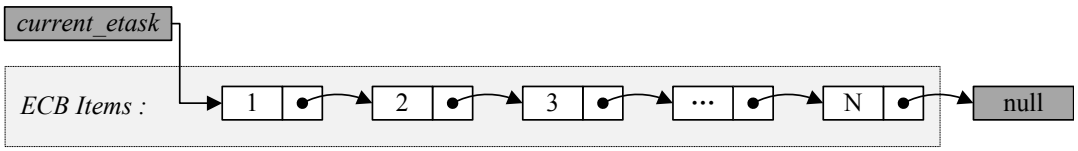


Figure 3.10: EPL: Indicating the priorities of etasks

The variables *thd_rdy* has the similar usage as *current_etask*. It provides an etask with the first thread to run (described in the next section). The variable *rest_time*, along with the constant *MAX_LIFETIME*, are used to calculate the priority of etasks (in Section 3.3.5). More details about the event-driven mechanism are in Section 3.3.5.1.

3.3.4.2 Thread

The group of thread components is called TCB (Thread Control Block). Table 3.8 shows the data structure for threads.

Thread has five states: four of them are similar to the ones in etask, including *Terminated*, *Sleep*, *Ready* and *Executing*, plus one more state of *Suspended*. Figure 3.11 demonstrates the possible transforms between states and the explanations follows:

- *Terminated*: A thread has this state when it is created. After the thread is triggered, this state is continuing until finishing the private resource collection.
- *Sleep*: When finishing the resource collection, a thread will enter the state of *Sleep* (the edge "t1" in Figure 3.11).

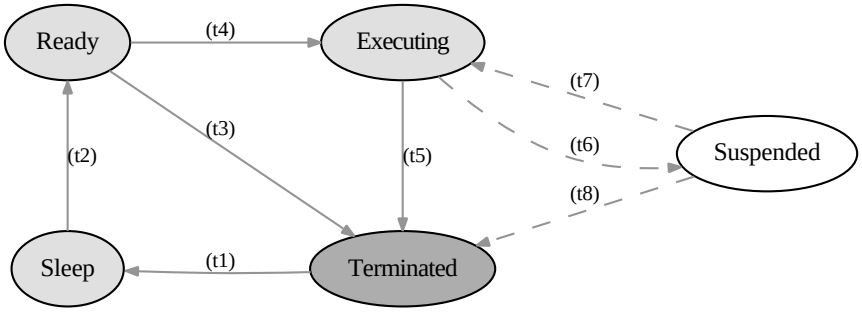


Figure 3.11: Transform of thread states

Field Name	Data Type	Description
state	char	*Terminated*, *Sleep*, *Ready*, *Executing* or *Suspended*
next_thd	struct Thread *	Pointer to the next thread to be run after finishing the thread
rest_time	unsigned long	Residual lifetime of the thread
func_addr	unsigned long	Memory address of a procedure function
master_etask	struct Etask *	Pointer to the etask to which the thread belongs
thd_stack	struct Stack *	Pointer to the private stack
etsk_tuple	struct Tuple *	Pointer to the *tsk_tuple* shared within etasks
THD_ID	char	Thread ID
THD_TUPLE	struct Tuple *	Pointer to the private *thd_tuple*
ORG_PRIORITY	unsigned short	Original priority of the thread
MAX_LIFETIME	unsigned long	Original lifetime of the thread

Table 3.8: The data structure for TCB nodes (struct Thread{...})

- *Ready*: After a thread is triggered and the resource collection is finished, it will enter the state of *Ready* (edge "t2") and wait for executing. If it expires its lifetime but not yet executed, is will be back to *Terminated* (edge "t3").
- *Executing*: When a thread is actually running by the scheduling program, it will be labeled *Executing* (edge "t4"). It will back to *Terminated* until its task is finished or it uses up the lifetime (edge "t5").
- *Suspended*: This label indicates a waiting state. A thread enters the state because of any of the following conditions:
  - A thread is preempted.
  - If a thread seeks for its interesting content from *tsk_tuples* but the content is not available.

  The difference between *Suspended* and *Ready* is the entering/leaving of former is with a context save/restore operation(edges "t6" and "t7"). If a thread uses up its lifetime, it will also enter the state of *Terminated* (edges "t8").

The variable *next_thd*, along with the *thd_rdy* in its master etask component, indicate the set of actions in the mater etask. The sequence of the one-way link list indicate the priorities of sub-threads in an etask, called TPL (Thread Priority Link), as shown in Figure 3.12. The *thd_rdy* is the head defined in an etask. The priority calculations are based on *rest_time* and *MAX_LIFETIME*.

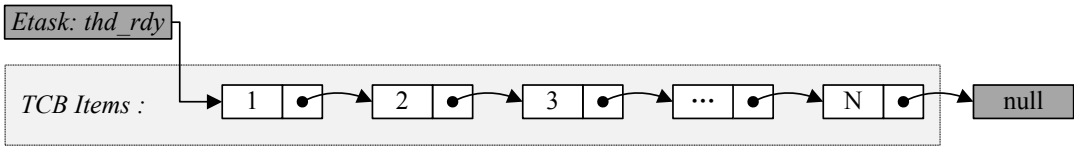


Figure 3.12: TPL: Indicating the priorities of sub-threads in an etask

The variables *thd_stack* and the *etsk_tuple* point to the private resources collected by etask. The structure of *thd_stack* is as shown in Table 3.9:

Field Name	Data Type	Description
cur_sp	unsigned long *	For calculating the stack pointer offset
start_add	unsigned long *	Starting address of stack
end_add	unsigned long *	Ending address of stack

Table 3.9: Stack Structure(struct Stack{...})

A private stack stores the contexts information for a thread when the thread enters the state of *Suspended*. The memory is collected dynamically after the thread is triggered, and the collection of stacks in an etask will be released when the etask is terminated.

The section only describes the component design of thread. More details about the hard real-time scheduling mechanism about thread are in Section 3.3.5.2.

3.3.4.3 Tuple

HEROS updates the tuple-based communication from the original Linda concept with the following two mechanisms:

- *Directional mechanism*: As previously mentioned in Section 3.3.4, a thread must be connected to at least one static tuple as a communication port. Instead of matching the structured name as the original LINDA concept, the implicit numeric constant (*KEY*) can be used to identify tuples and orient messages. The *KEY* works as a quick reference, but whether to continue running a thread can also be depending on the content in tuple.
- *Owner mechanism*: An OUT primitive can write data to any tuple unless its *TUPLE_TYPE* equals to true (means a *tsk_tuple*) and *owner* is not null. This situation happens if a *tsk_tuple* is a locked and working as a private resource shared within an etask; the *owner* here is pointed to an etask. Except this situation, there is no more owner limitation for IN/OUT primitives. The *owner* in a *thd_tuple* only works as a reference to the master thread of it. In addition, a passive tuple is an unoccupied *tsk_tuple* where *master_thd* is null. A passive tuple can work as a regular buffer.

Because of the resource constraints in embedded applications, some additional mechanisms are not designed to be implemented to the current HEROS, e.g. the read() primitive and the garbage collections for tuples. Besides, there is no underlying mechanism for the current HEROS to prevent the abuse of IN/OUT operations, it is depending on the user programs built on HEROS to operate the tuple-based communications correctly.

The TS in HEROS is a table containing the items of tuple with data structure as shown in Table 3.10:

Field Name	**Data Type**	**Description**
owner	void *	Pointer to a master thread or etask (default is null)
out_head	unsigned char*	Pointer for writing
in_tail	unsigned char*	Pointer for reading
msg_num	unsigned short	Message counter (Equals to 0 if the tuple is empty)
KEY	char	Tuple ID
TUPLE_TYPE	char	The type of tuple: *thd_tuple* or *tsk_tuple* (0 or 1)
START_ADD	unsigned char*	Starting address in memory
END_ADD	unsigned char*	Ending address in memory

Table 3.10: Tuple Structure(struct Tuple{...})

A tuple in memory is implemented as a ring buffer, and the ring buffer is a critical resource operated by IN/OUT primitives. An OUT operation inserts data into a tuple, and the data is written at the head of the ring buffer indicated by *out_head*. An IN operation withdraw the data from the tuple, and it is started from the tail indicated by *in_tail* then directed to the head. An OUT/IN operation causes a plus/minus one on *msg_num* until the tuple is full/empty, respectively.

The constants *START_ADD* and *END_ADD* can be easily updated to a dynamic collection, and they can be modified to indicate a local, shared or distributed memory. Because the

applications built on HEROS need only to communicate through the tuple-based component interface by IN/OUT primitives, the further modification on low-level memory details will not require rebuild for these higher-level applications.

3.3.4.4 IN/OUT Primitives

In the end of this section, we show the processes in the IN/OUT primitives. This section only describes the Linda-based component designs. For both IN and OUT primitives to a *thd_tuple*, the last step will be branched to the scheduling mechanism, which will be introduced in the next section. The italic font with brackets in the following figures indicates the major variables or constants used in the processes.

The *LOCKED* and *UNLOCKED* in the following figures represent the *binary semaphores* (with the values of 0 and 1) for protecting the critical sections. If it is just for slow and simple embedded applications, binary semaphores could be replaced by *DIS_ALL_IRQ* (disable all interrupts) and *ENA_ALL_IRQ* (enable all interrupts). however, in practical embedded applications with more complex interrupt services, to disable and enable all interrupt would at least cause the loss of data.

The description starts from the processes in the OUT primitive as shown in Figure 3.13. The major branch of an OUT primitive is based on the types of tuple:

- For the *tsk_tuple*, the major processes are identifying *owner*, writing, and return.
- For the *thd_tuple*, the states of related components will be updated from *Terminated* to *Sleep* after writing. The related components include: 1) the master thread refereed by *owner*; 2) the master etask referred by *master_etask* (in the thread component). Note that, after the state updating, the master etask will not start to collect resources immediately.

When receiving signals and messages from interrupt services or other threads, the data will be written to tuples by OUT primitives. There are two tasks need to be finished by an OUT primitive: 1) writing data; 2) collecting resources and scheduling. The processes of first task are as shown in the figure, the rest are in the Figure 3.16 of Section 3.3.5.1.

The processes of IN primitive are shown in Figure 3.14. The top branch of an IN primitive is based on the message number leaving in a tuple. If the tuple is empty after withdrawing data, the *return* happens immediately for a *tsk_tuple*. But for a *thd_tuple*, it changes the state of the thread and get into the scheduling processes as shown in Figure 3.17 of Section 3.3.5.2.

HEROS assume that when the data has been withdrawn from a *thd_tuple*, the related thread has finished the duty, and the thread should be suspended or terminated. Consequently, there are two tasks that need to be finished by an IN primitive:

- withdrawing data
- releasing private resources (if all sub-threads has been terminated) and calling for rescheduling

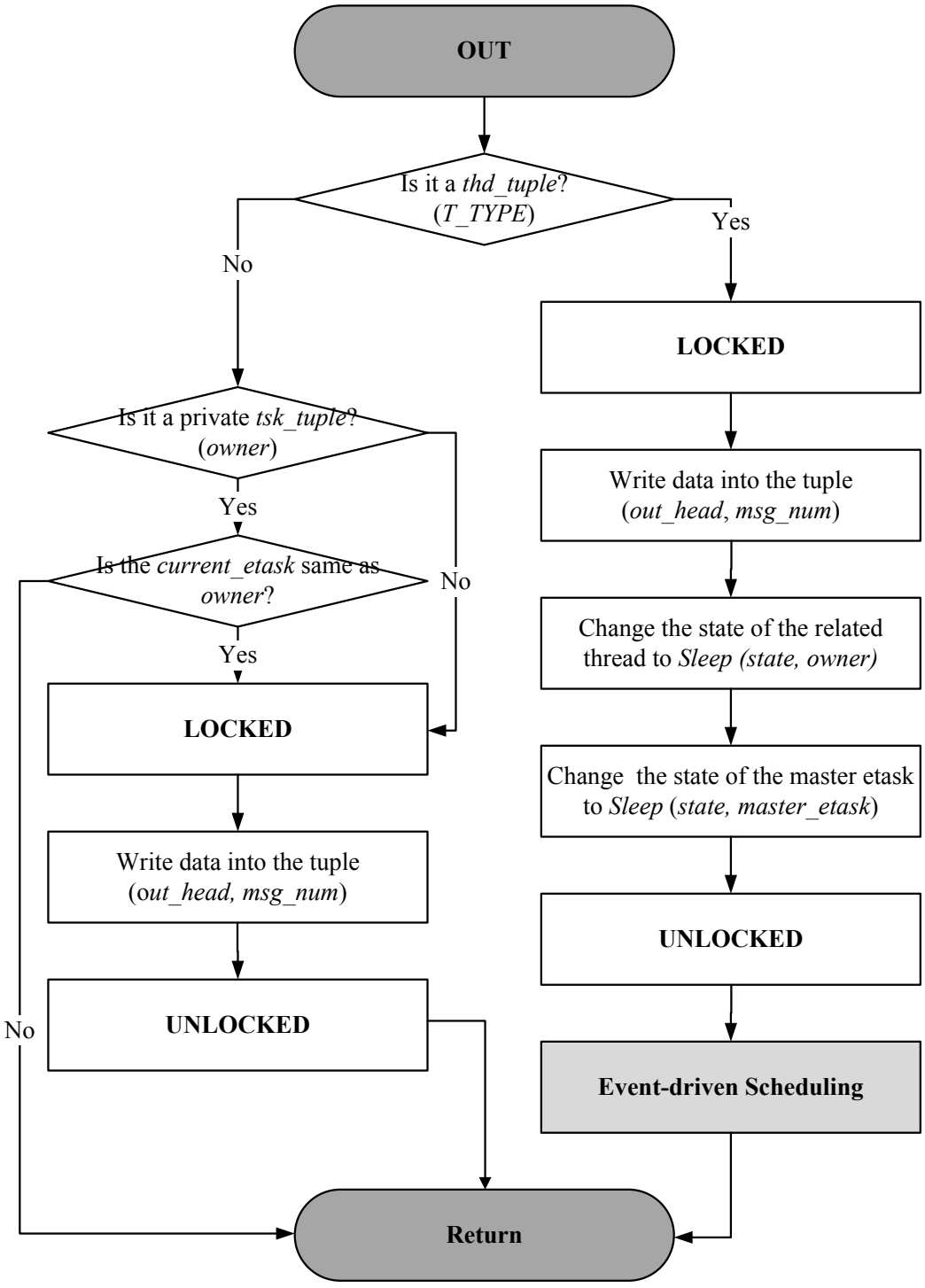


Figure 3.13: Processes in the OUT primitive

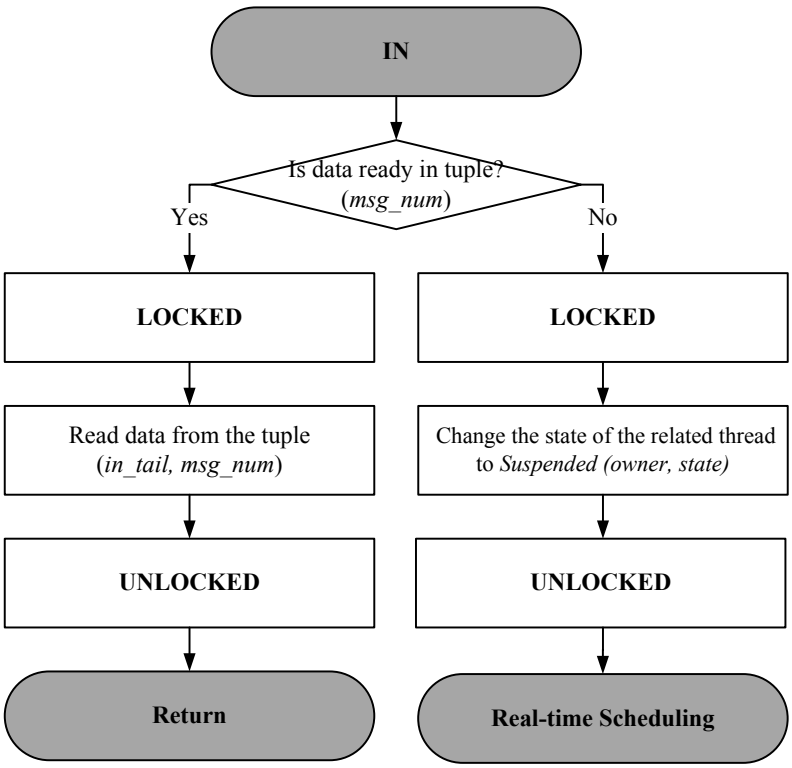


Figure 3.14: Processes in the IN primitive

3.3.5 Hybrid Priority-based Scheduling

Based on the two-level components, HEROS adopts a hybrid mechanism to merge both event-driven and real-time multitasking scheduling mechanism. The HEROS application can be configured to run in two modes:

- If the implementation contains only one etask, the preemptive threads are scheduled in a hard real-time multitasking mode.
- In the implementation with more than one etask, a two-level scheduling is used: etasks are run in sequence, and threads are scheduled in an event-driven soft real-time mode. Besides, to give the event-driven mode with a certain level of hard real-time support, there is an optional emergency mechanism in HEROS to allow the highest priority etask taking over the executing etask.

The priority-based scheduling mechanism of HEROS adopts the EDF (Earliest Deadline First) algorithm, which can assure the threads meet their hard real-time deadlines if $U \leq 1$ in Eq. 3.3. The main idea of EDF algorithm is to search the real-time tasks closest to its deadline and execute it. The EDF is one of the most common algorithms using in the real-time scheduling mechanism. It has the disadvantage on the fault tolerance if $U > 1$, but the two-level scheduling mechanism reduces such situation. The original EDF algorithm does not always give the optimal result for non-preemptive scheduling, thus it is better used for the preemptive thread scheduling. For the etask scheduling, HEROS uses static scheduling, or EDF as an optional choice.

3.3.5.1 Event-driven Scheduling

There are four logical layers in the event-driven mechanism as shown in Figure 3.15 including the layers of trigger, keeper, dispatcher, and handler.

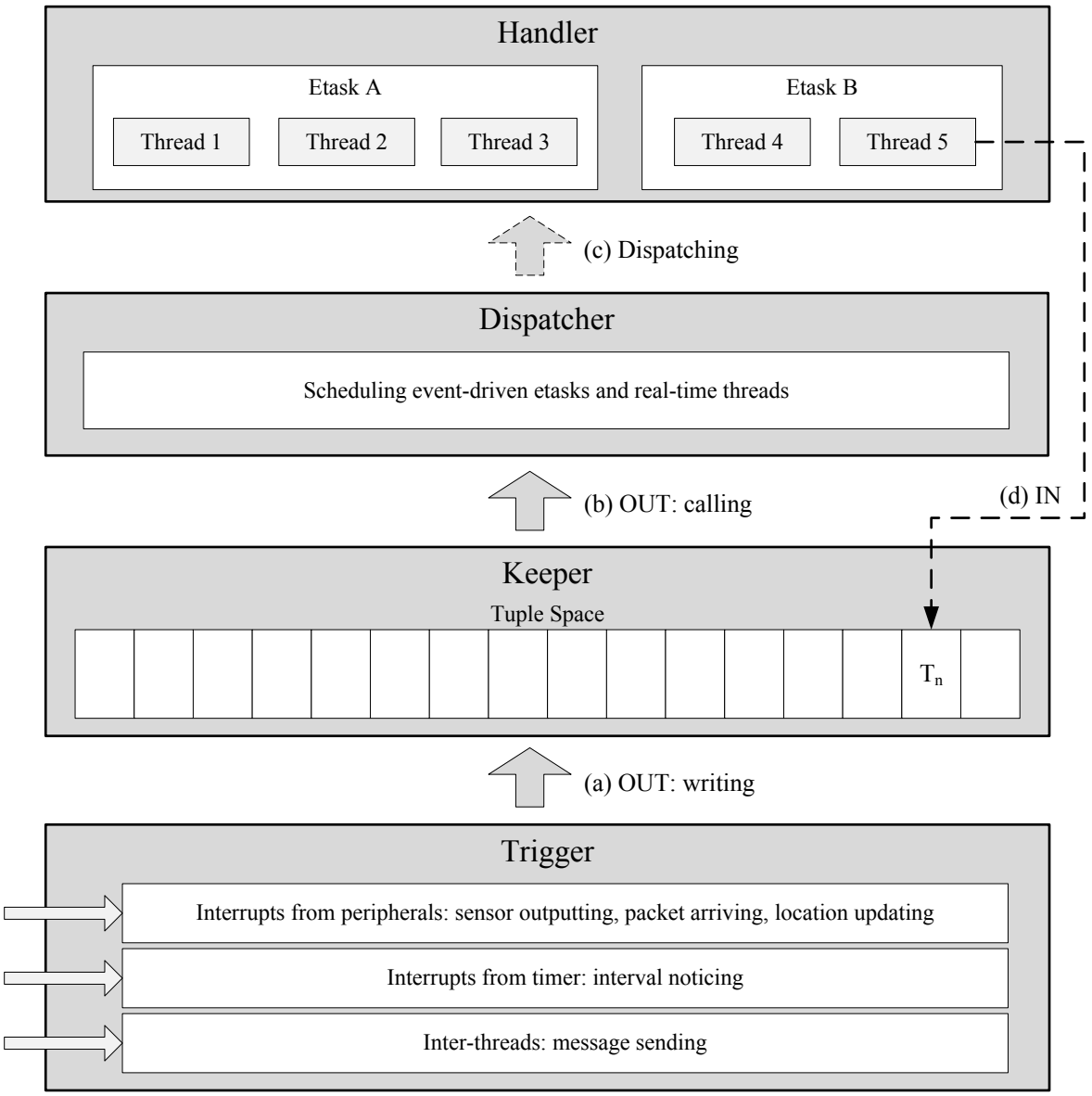


Figure 3.15: Logical layers for event-driven mechanism

- The *trigger* layer receives signals or messages from interrupt services or threads. A trigger action implicates that there will be a further handler action. The interface function to the next layer is the OUT primitive.
- The signals or messages are saved as the tuple-based format in the *keeper* layer. The Linda tuple-based mechanism loose couples between trigger (sender) and handler (receiver) in this layer. When the trigger layer continues sending data, the data processing in handler layer are working in parallel. Moreover, because an incoming data implicates

a further handler action, if tuples with incoming data is *thd_tuples*, the last part of an OUT primitive is to activate the scheduling mechanism in a dispatcher layer.

- The *dispatcher* layer has two duties: scheduling components and collecting resources. The scheduling duty is to calculate and resort the component sequence based on the static priorities or the EDF algorithm. After finished scheduling, only one thread in one etask will be executed. If the component has no resource to run, the second duty in dispatcher layer is to collect resources.
- The *handler* layer contains the actual event "customers": threads and etasks. For example, the *Thread 5* in *Etask B* in Figure 3.15 has the highest priority to be executed, thus it is selected by the dispatcher layer to run its data processing task on Tuple T_n. Note that, if the T_n has no incoming data, the thread will not be activated by the trigger layer.

The following detail more about the OUT primitive and the event-driven scheduling mechanism in/between the keeper layer and the dispatcher layer. After finishing writing data, the processes of OUT primitive is branched to scheduling processes. Moreover, if all sub-threads in an etask finish their duties, the real-time scheduling program will call event-driven scheduling program to reschedule the next etask.

The Figure 3.16 is the unfinished part of the Figure 3.13 for the OUT primitive.

The top branch divides the flowing to two directions. Both of them are to operate the EPL. Because the etask is non-preemptive, if there is already an etask executing, only the etask from the second to the last one will be resorted (if the EDF algorithm is applied). As previously mentioned, there is an optional emergency mechanism to allow the highest priority etask taking over the executing etask, and it happens here. The implementation is to disable the *current_etask* and *current_thread* anyway, then move the highest priority etask to the head of EPL and move the executing etask to the next.

If the global *current_etask* pointing to a terminated etask, the next etask will be move to the first place. The rest processes are divided based on the state of *current_etask*. If the *current_etask* is ready, execute it; if not, collect resources for it. In the end, if the all etask are terminated, call daemon thread after return from event-driven scheduling program. Because the state of an etask will be changed by an OUT primitive from *Terminated* to *Sleep*. In other word, if the calling is from an OUT primitive, at least one thread and its master etask should be activated. Normally, only the call from the real-time scheduling program in the next section will cause the calling of the daemon thread.

3.3.5.2 Real-time Scheduling

An event-driven scheduling program will be switched to the real-time scheduling program, if there is at least a thread in an etask that need to be executed. As previously mentioned, the etask is designed as a packing widget containing a group of threads to complete the reactions to an event, thus before entering the real-time scheduling program in an etask, all the sub-thread should already have been in the state of *Ready*. The real-time scheduling program switches the states between *Ready* and *Suspend*, until all threads finish their duties. After that, the real-time scheduling program will switch back to event-driven scheduling program, which will arrange the next etask to run.

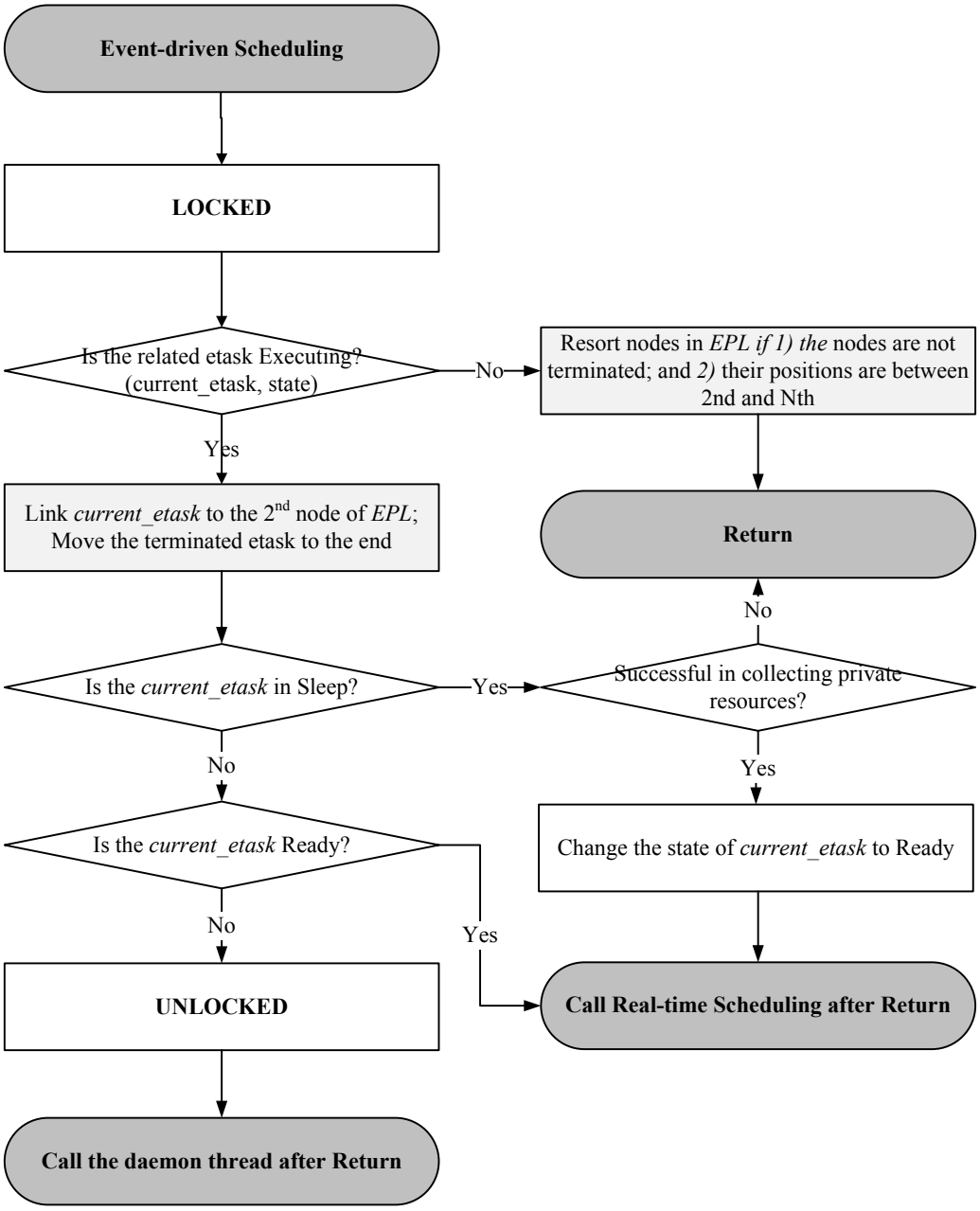


Figure 3.16: Processes in Event-driven Scheduling

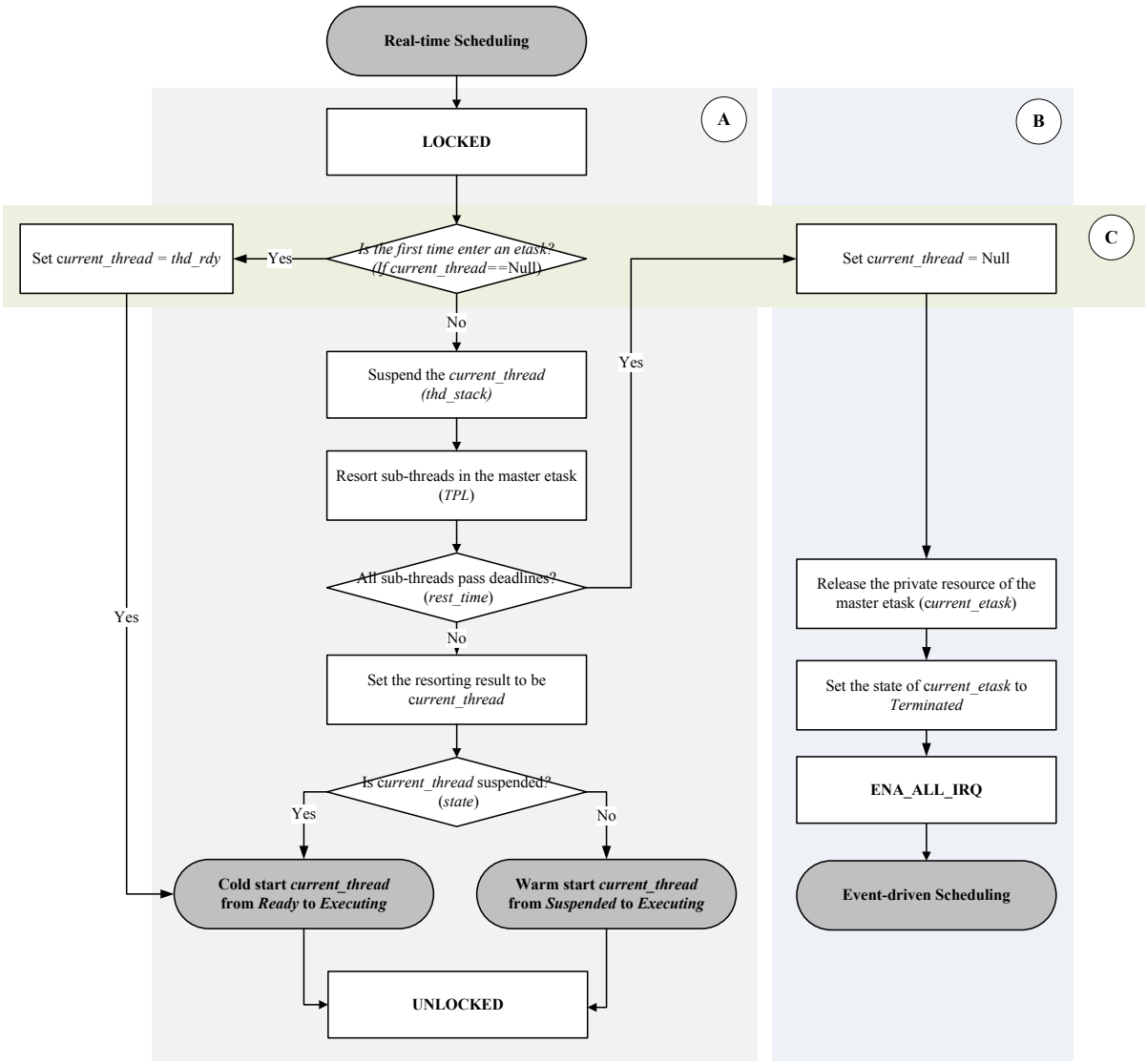


Figure 3.17: Processes in Real-time Scheduling

The real-time scheduling program is relatively simpler. The part A in the Figure 3.17 contains regular real-time preemptive process. In the end of part A, a *warm start* executes a suspend thread and resort the context stack from it private resource. For a *cold start*, this resorting action is not called. The part B tells how the real-time program switches back to event-driven program. There are two tasks need to be done here: releasing private resources, and changing the state of etask to *Terminated*. The part C is to indicate whether it is the first time to enter the etask. If it is, the *current_thread* must be referred to the first priority thread.

3.3.6 CIVIC with HEROS

The embedded communication system can provide adaptive task mechanisms for different IVC application requirements based on the hybrid mechanism in HEROS. This section gives an example of an event-driven software design that has been tested. Because the flow of computing process in this design is the event-driven multitasking mode, thus it can be well adapted to the message input flow of the CIVIC protocol and it can leave low memory footprint [97].

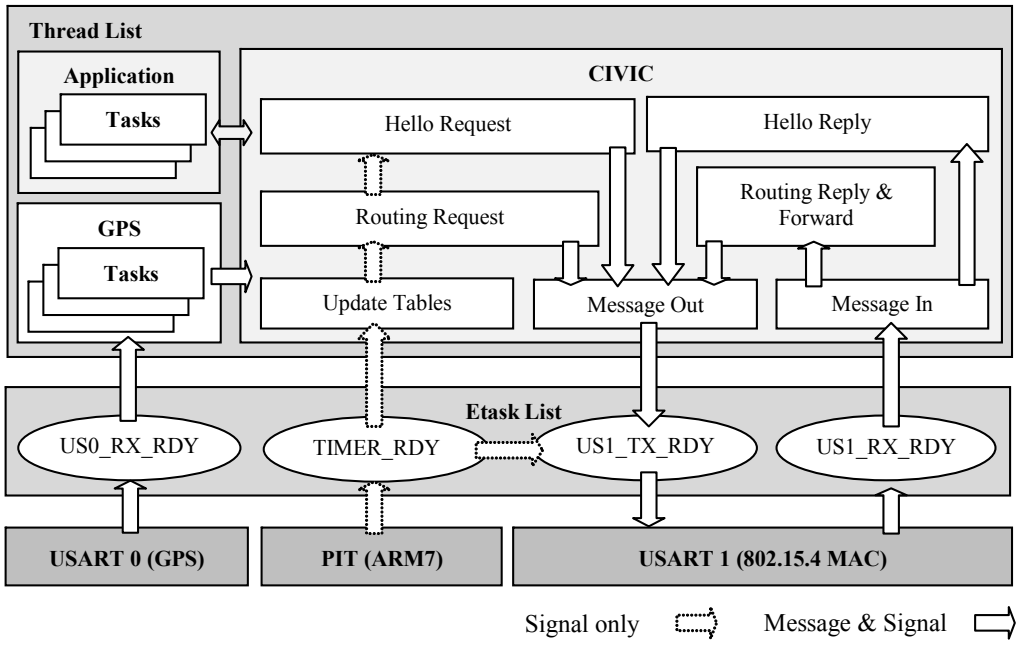


Figure 3.18: System stack and event-driven data flow

There are four major event-driven etasks in Figure 3.18. The *TIMER_RDY* etask is driven by interrupts from the PIT (Periodic Interval Timer). The rest of etasks are mainly driven by interrupts from the USART (Universal Synchronous/ Asynchronous Receiver/Transmitter) ports connected to GPS module (USART0) or XBee module (USART1).

Figure 3.19 shows an example of processing flow between etasks and threads. Only the etasks and threads relating to the major system process are shown in Figure 3.18 and Figure 3.19.

The *TIMER_RDY* etask runs the periodic tasks, e.g. sending "Hello" messages, activating proactive routing searches, and removing the out-date table items. The tables need to be cleared periodically are the neighbor table and the routing table.

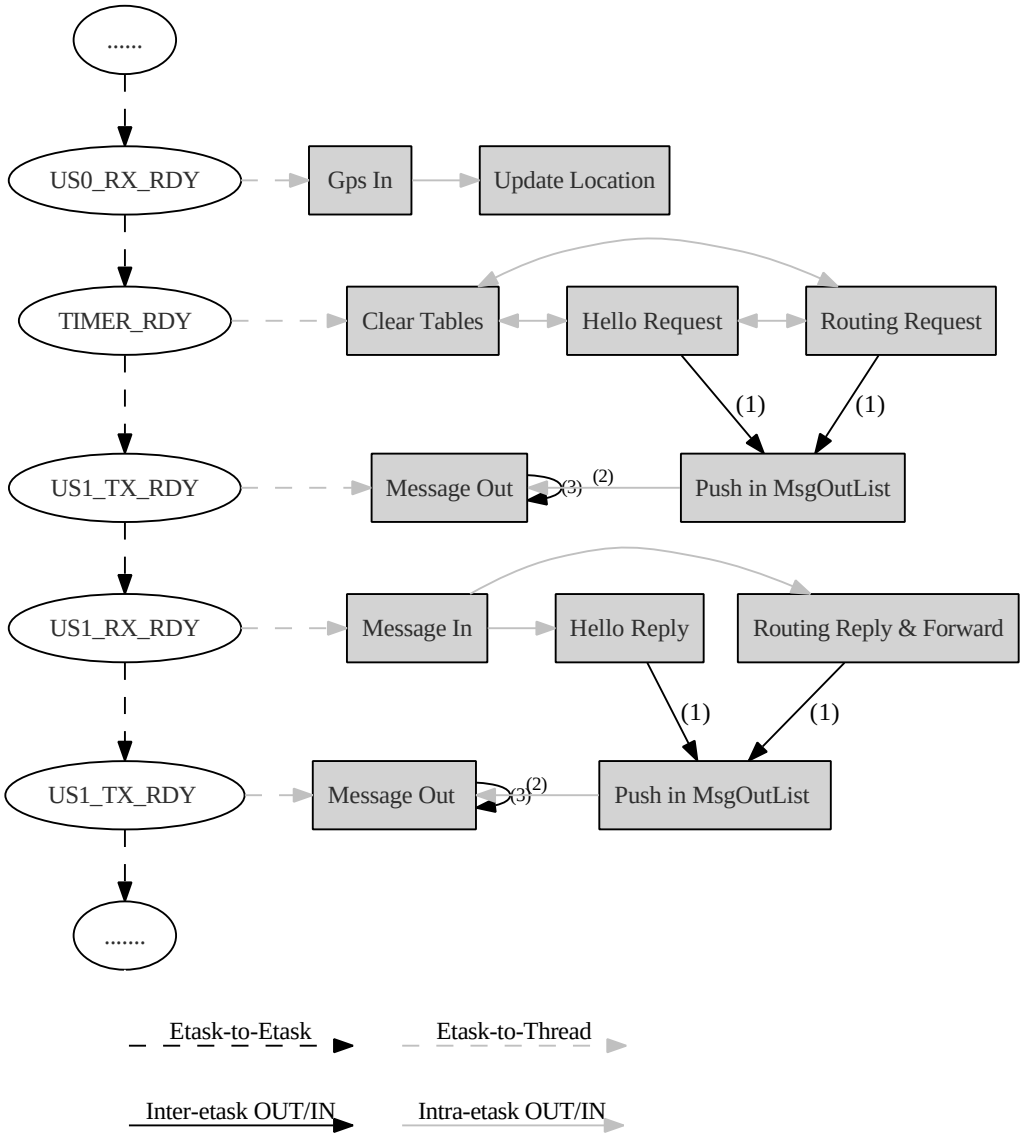


Figure 3.19: The interactions between etasks and threads

The *US1_TX_RDY* etask handles the message outputs. To avoid the sending intervals becoming too short, other etasks should not directly send out messages. Instead, they push messages into a buffer list called MsgOutList (Step 1 in Figure 3.19). It will activate the *US1_TX_RDY* etask to check whether the last transmission has been finished. If it has been finished, a message will be sent out by the "Message Out" thread (Step 2); if not, the etask is end, and a PIT timer will be activated to run the etask after a waiting period (Step 3). In addition, for the time-sensitive designs, the TIMER_RDY etask can take control of the output related to send message at a fix interval.

The etasks *US0_RX_RDY* and *US1_RX_RDY* contain threads to process incoming raw data. The former deals with the GPS data, the latter deals with the CIVIC data. The major routing for these two etask is similar: 1) when the input buffer is ready for data processing, a thread translates the raw data into meaningful messages; 2) based on the message types, the etask divide messages into the related threads for further actions.

Note that, this section only shows the overview of an event-driven design. The settings of treads (tasks) are normally different in applications, event for the event-driven design. More details about actual task implementations can be found from Section 4.1.

3.3.7 Performance Evaluation

HEROS has been implemented with underlying mechanisms for our targeting high-dynamic networks including WSNs and VANETs: On one hand, the WSNs focuses on information retrieval and the subsequent matching of the attributes of certain phenomenon, the event-driven mode in HERO can be adopted to these applications; on the other hand, the VANET application such as hazard alarming and cooperative driving require extra effort to deal with real-time operations, HEROS can be configured to work in hard real-time scheduling. This section provides the experiment results on system latency when operating even-driven and real-time mechanisms. For using in high-dynamic networks, the system latency in HEROS has been minimized. Moreover, HEROS is dedicated to low-cost mobile devices, thus it must have less requirement of memory resource. The result on memory aspect is also given in the section.

3.3.7.1 System Latency and Memory Consumption

The hard real-time scheduling mechanism should provide a predictable response time. In the mechanisms of HEROS, the major variable to affect the system latency is response time of IN/OUT primitives, which is directly related to the message size that need to be handled by the primitives. Because the message size is limited within the size of the pre-defined tuples, and the maximum number of tuples is limited within a static TS, thus the response time of IN/OUT primitives are also predictable. The following result is from the performance evaluation of IN/OUT primitives at 48MHz.

The first part of system latency is from IN/OUT primitives. As the OUT primitive shown in Figure 3.13, the major factors of response times are message writing and event-driven scheduling. The equations of the number of program cycles and the response time are $104 + 32n\ cycles$ and $2.164 + 0.666n\ \mu$s, respectively, where n is the size of data in the byte unit. For the IN primitive as shown in Figure 3.14, there is a difference between whether to

call the real-time scheduling. If calling for scheduling, the maximum equation in Table 3.11 is applied; or else, the minimum one.

	Cost(cycles)	Time (μs)
Maximum	$149 + 46n$	$3.101 + 0.957n$
Minimum	95	1.977

Table 3.11: Cycle and response time for IN Primitives

The second part of system latency is from the etask (event-driven) switch and the thread (real-time) switch as in Table 3.12. The former costs 90 cycles, and it responds in 1.873 μs. For the thread switch, there is a major difference from cold start and warm start:

	Cost(cycles)	Time (μs)
Warm start	99	1.873
Cold start	89	1.852

Table 3.12: Cycle and response time for Thread (real-time) switch

In the memory aspect, a minimum implementation of HEROS needs only about 5 KB memory (code: 3572 bytes; data: 1272 bytes). The required memory for HEROS is small enough to be available for most of the existing wireless sensor boards.

3.3.7.2 Comparison with TinyOS

Table 3.13 shows the comparison between HEROS and TinyOS. HEROS is tested on AT91SAM7S256 (48MHz), and the TinyOS results are from [98] with ATmega128 (4MHz). The table only compares three system operations (i.e. scheduling a task, context switch and hardware interrupt latency) and the major memory consumption between HEROS and TinyOS. Because the tasks in HEROS are pre-configured, when the number of etasks and threads increase, the sizes of components and TS will also increase.

	HEROS (AT91SAM7S256)		**TinyOS (ATmega128)**	
	Cost(cycles)	*Time (μs)*	*Cost(cycles)*	*Time (μs)*
Scheduling a task	43	0.895	46	11.5
Context Switch	56	1.165	51	12.75
Hardware Interrupt (hw)	5	0.104	9	2.25
Hardware Interrupt (sw)	61	1.269	71	17.75
	Size (bytes)		*Size (bytes)*	
Code	3572		1272	
Data	432		48	
Sum	4004		1320	

Table 3.13: Comparison between HEROS and TinyOS

Note that the operation of context switching happens between the two threads in the 'warm' mode. In order to support real-time multitasking operations, HEROS has more system overheads than TinyOS but has similar system cycles for the basic system operations.

3.4 Hardware Platform: LiveNode

3.4.1 LiveNode Components

The hardware platform used by our real-world experiments in the thesis is the LiveNode sensor developed by our team [99]. It is a versatile wireless sensor node, which enables to implement rapidly a prototype for different domains of applications such as telemedicine (wireless cardiac arrhythmias detection), inter-vehicle communication [100], and environmental data collection (FP6 EU project NeT-ADDED).

The LiveNode sensor is a small board (70x55mm) and is powered by a 9 V standard battery. It may be equipped with different types of components (GPS, Wi-Fi, ZigBee, GSM and different type of sensors) to meet the requirements of an application. The major components of LiveNode sensor used in our real-world experiments have three parts as shown as shown in Figure 3.22:

- The Atmel AT91SAM7S256 microcontroller [101] is used for data processing. It is an ARM7TDMI based high-performance 32-bit RISC microcontroller with Thumb extensions with USB Device Interface, 32 I/O pins, one Advanced Interrupt Controller, one Periodic Interval Timer, two USARTs, 256 K bytes Flash and 64 K bytes SRAM.
- The MaxStream XBee Pro module [102] is to ensure the wireless communication of IEEE 802.15.4 standard (ZigBee). The module operates within the ISM 2.4 GHz frequency band. It is low-cost, it requires low power (e.g. TX peak current is 45 mA at 3.3 V and power-down current is less than 10 μA), and it can reach a wide range (the outdoor line-of-sight transmission distance is up to 1600 m [102]). The ZigBee is chosen in our new experiments instead of Wi-Fi because ZigBee has an outdoor RF line-of-sight range up to 1.6 km and an indoor range of 100 m, which is equivalent to Wi-Fi indoor range one. Besides, the energy consumption for ZigBee module is less than the available Wi-Fi modules.
- The GlobalSat ET-301 GPS module [79] is for specific GPS signal processing. It is a 20-channel all-in-view tracking receiver. It communicates on the serial port with the micro-controller at the default baud rate of 4800 bps. The receiver automatically sends its complete message once every second. Then, the micro-controller detects and decodes the message.

If not specified, the "sensor" in the following sections or chapters means the LiveNode with above three components.

3.4.2 Medium Adaptation Layer: Multiple Wireless Supports

To adapt different roadside infrastructures and utilize more radio spectrum, our communication system is designed to support multi-radio and multi-channel on network nodes. The radio and channel should be auto-configured to minimize interference. The LiveNode sensor can be equipped with three types of wireless access medium: Wi-Fi (IEEE802.11b), ZigBee (IEEE802.15.4), or GSM (GPRS). Several LiveNode sensors can be connected together to enable the multiple wireless supports by using extension connectors (SPI, I2C, and I/O

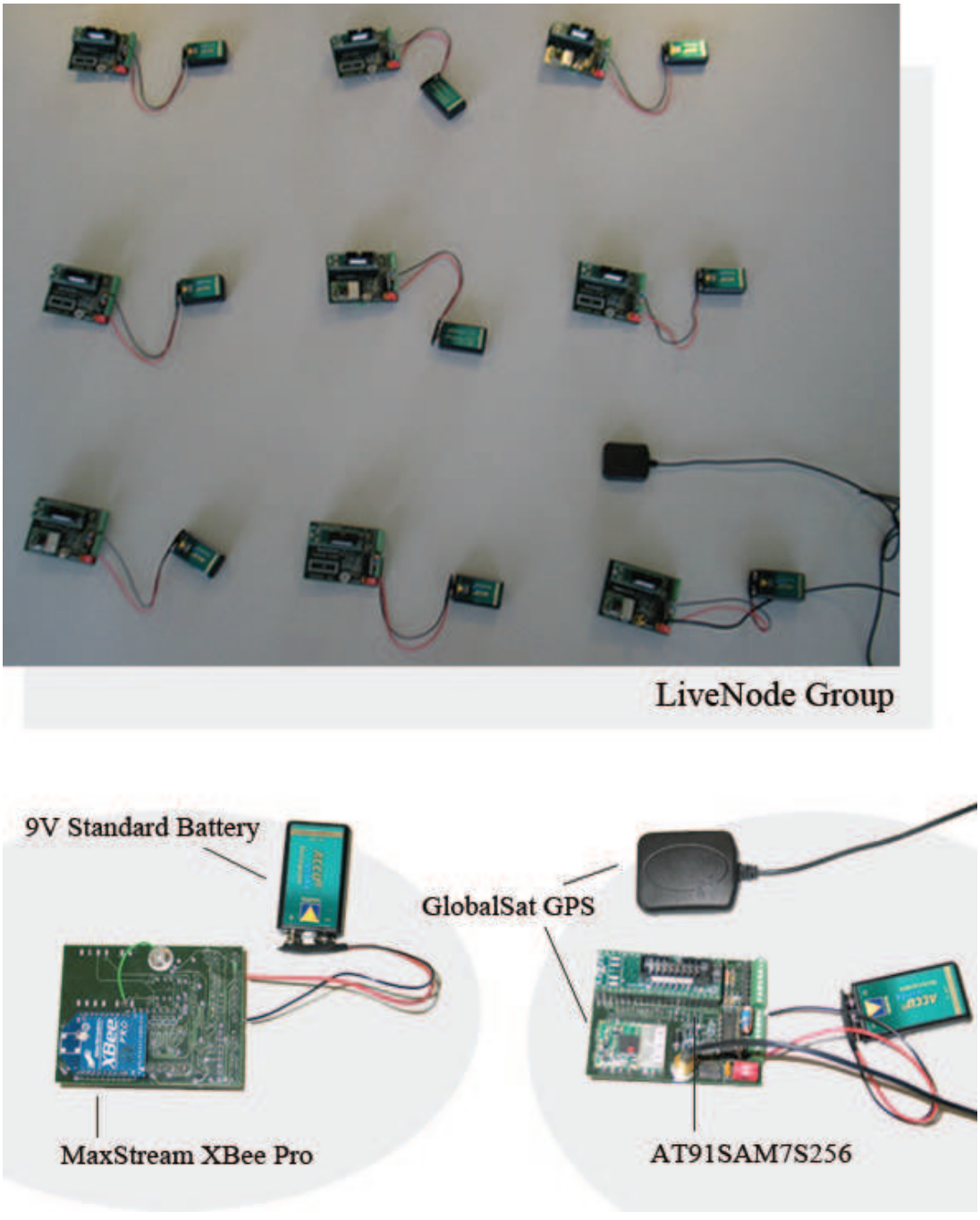


Figure 3.20: LiveNode platform

Figure 3.21: Two LiveNode sensors link together: ZigBee plus Wi-Fi

connectors). For example, two LiveNode sensors in Figure 3.21 are linked as a couple. The couple of LiveNode sensors can then provide both ZigBee and Wi-Fi supports.

The Wi-Fi and ZigBee have been adopted by the communication system. From inter-vehicle communication, smart home, to telemedicine, the Wi-Fi enables many applications to connect to the Internet. With an appropriate antenna, the radio range of Wi-Fi can reach 1 Km with the vehicle speed about 100 Km/h. A new trend of wireless standard is ZigBee, which requires lower cost and lower power. Its maximum outdoor line-of-sight transmission distance is 1600 m [102].

3.4.3 Designs in Hardware Driver

The programming of drivers seems trivial but it is important for a reliable quick-reaction communication system. Moreover, it must work closely along with the medium adaptation layer. An example of the USART driver (only the part of inputting data) is given to demonstrate the major design as shown in Figure 3.22.

As the lowest level of the communication system, it must provide simple IN/OUT interface for upper layer. In addition, it should have certain level of self-configure functions. In the figure Figure 3.22, *Us1IrqHandler*, *Us1EndInput* and *Us1BufSwitch* on duties of putting the network data into two switchable buffers, then the *Us1Prc* pass the data to the upper layer. In case of the upper layer does not have enough time to deal with the data, some interrupt services suspended, and the overrun data will be cleared.

The functions in the figure together can be consider as one of the "inputting triggers" in the trigger layer of HEROS logical layers (in Section 3.3.5.1). To keep the data sending and receiving in high speed, the driver programs are driven by interrupts. The major interrupt sources are on the left side of the figure. Each source has a standalone modular to handler, and the modular are normally protected by flags. The interrupt services are for noticing the

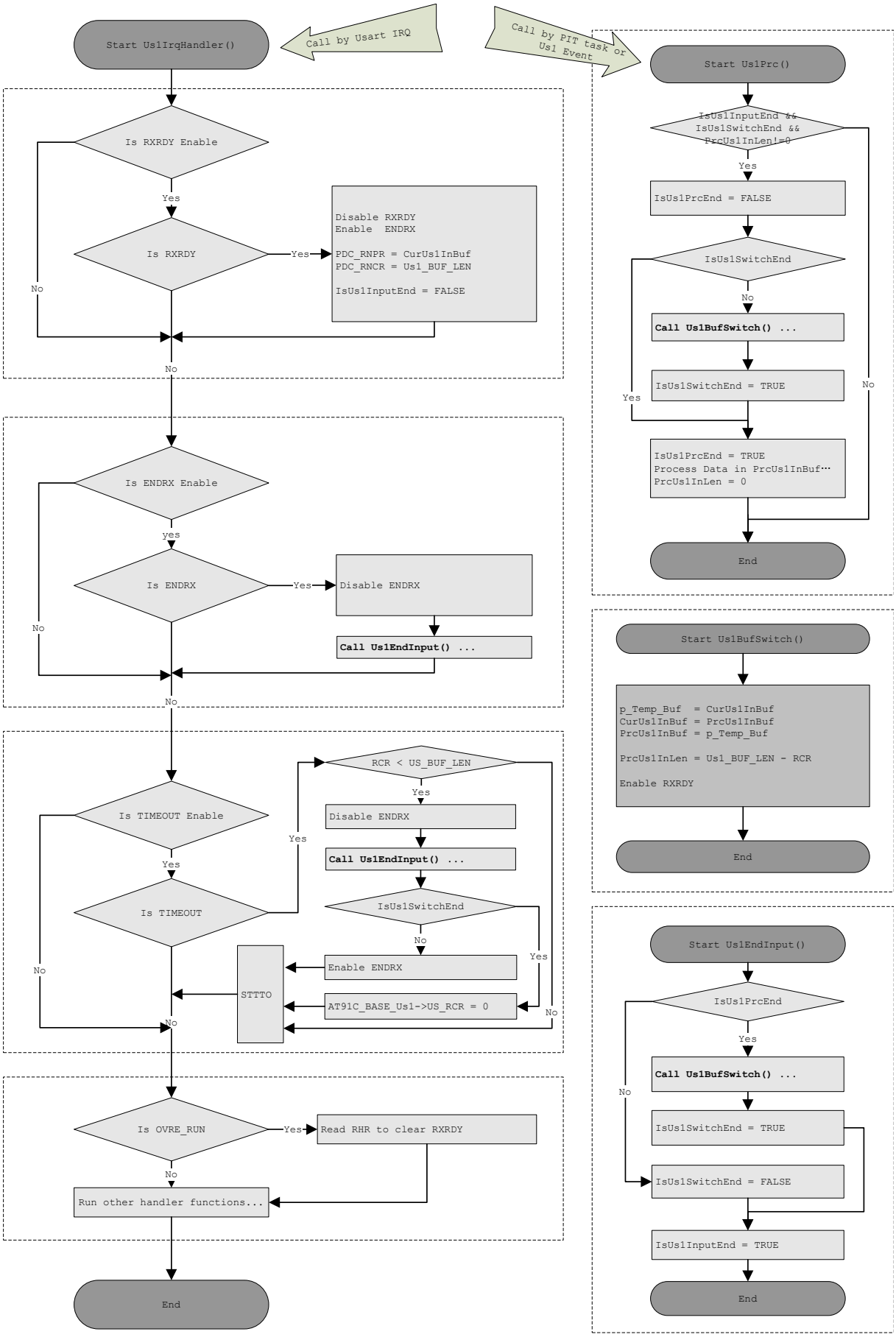


Figure 3.22: Driver for XBee Pro chip: An example for inputting data

happening of events. No actual data copy or move should be happen here.

3.4.4 Low-cost GPS module and LCD-GPS solution

As previously mentioned, the CIVIC protocol is based on location-related contexts, thus a localization solution is one of key elements of our communication system. This subsection introduce an evaluation of the low-cost GPS module that we used, and a brief of the LCD-GPS solution developed by our team. The evaluation is a part of the evaluations of the communication system, but since it is not directly relating to network communications, it is put in the hardware section.

A common solution to improve the GPS accuracy is to use DGPS (Differential GPS) [103]. However, installing a reference station or receiver for DGPS is complex and very expensive. A DGPS station costs about 30000 €, and a DGPS module (e.g. RTK-DGPS) is about 3000 €. The costs of DGPS products make them hard to be applied to most VANETs and WSNs. The communication system chooses a low-cost solution by using only the standard GPS (civil GPS) module (it is about 50 € for the GPS module on LiveNodes) along with the LCD-GPS solution to improve the accuracy of locations.

The Figure 3.23 shows an experiment for the GPS module on LiveNodes, and the result is normally common for other low-cost GPS modules. The experiment was conducted with three sensors in three days. The average distance of the sensors was 41 meters, and the experiment periods were approximately between 9 AM and 12 AM.

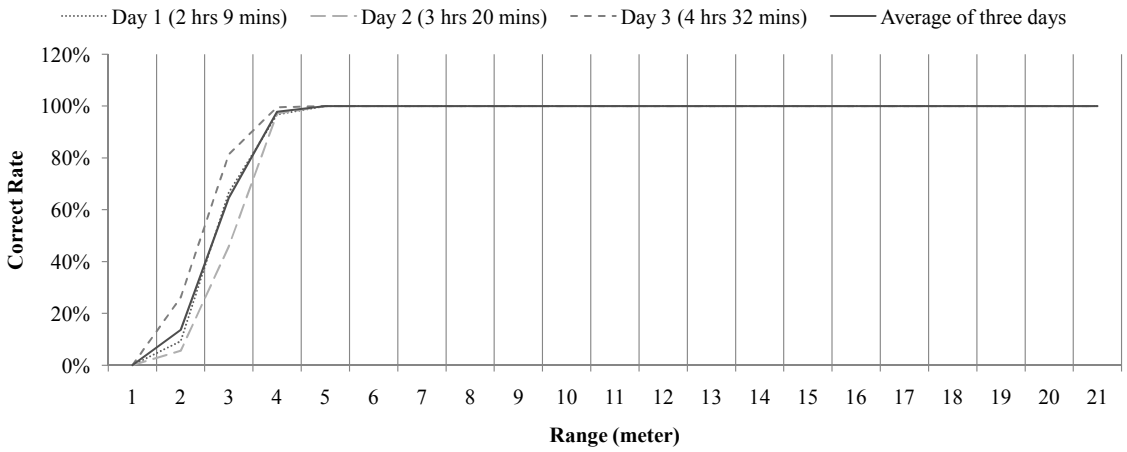


Figure 3.23: Correct rate for an experiment for GPS accuracy

The ranges in the figure are the radiuses from the accurate positions got from a RTK-DGPS module, which provides the position with a 10 cm deviation. For the GPS module on LiveNodes, 13.71% position is correct in one meter, 64.57% in two meters, and the maximum range is in four meters. In other word, the GPS module can get positions with a four-meter deviation in the worst case. For many WSNs, this accuracy may be enough; moreover, for most wireless sensors, they can normally rely on the sink node or master nodes to know their raw positions. For VANET applications like hazard warning, the minimum accuracy is one meter [104], thus the previous accuracy need to be improved.

The main idea of LCD-GPS solution is to use some fixed nodes as reference stations. The other nodes are called mobile nodes. Reference stations and mobile node have the same hardware equipments. The LCD-GPS solution assume all reference stations know their positions with a good accuracy. The reference stations analyze the instantaneous GPS errors, and they cooperate to deduce a global error correction. The mobile nodes apply the correction result and get more accurate positions. There are several degrees of calculations in LCD-GPS. The simple difference is the easiest to be applied because it can be run locally, and it does not require the global correction results from the network server.

Back to the previous experiment, after applying the simple difference between the sensors, we got the corrections of standard deviations as in Table 3.14 [78]. The average

		Sensor 1	**Sensor 2**	**Sensor 3**
Original standard deviation	*East*	0.7257	0.6317	0.6526
	North	1.3083	1.1173	1.2356
	Height	1.6904	1.6113	1.8555
When sensor 1 as the reference station	*East*	-	0.4216	0.4579
	North	-	0.6646	0.7772
	Height	-	1.1067	1.3123
When sensor 2 as the reference station	*East*	0.4216	-	0.4458
	North	0.6646	-	0.6289
	Height	1.1067	-	1.1627
When sensor 3 as the reference station	*East*	0.4579	0.4458	-
	North	0.7772	0.6289	-
	Height	1.3123	1.1627	-

Table 3.14: Corrections of standard deviations on the three directions (unit is meter)

By comparing with the original standard deviation, the average deviation gains 30

The integration of LCD-GPS and the communication system is still an ongoing work. some experiments involve CIVIC and LCD-GPS is in [19]. In the design, the GPS data are carried by CIVIC protocol between reference stations and mobile nodes. The LCD-GPS solution calculates corrections, and then both reference stations and mobile nodes can get more accurate positions.

3.5 Network Specification: 802.15.4

The communication system does not have a strict limitation on network specification. The CIVIC protocol can handle the addressing of network nodes, multi-hop peer-to-peer routing, and it provides the communication interfaces for upper layers. The only requirement of the CIVIC protocol for the underlying network medium is to provide a one-hop broadcast mechanism.

In this case, our communication system chooses the IEEE 802.15.4 standard [105] as its underlying layer. The standard is well known as a low-power consumption standard requiring low-cost devices. For example, the MaxStream XBee Pro chip costs only 32 U.S. dollars, and it requires just 63 mW transmit power [102], which is an ultra low power consumption. To keep CIVIC protocol adaptable to other network specification, we only enabled the broadcast mode with the unslotted CSMA/CA algorithm provided by the XBee Pro module. Actually, even if we enabled the slotted CMSA/CA with a beacon node to control the networking, there is still a high probability that two sensors would sense the same slot being free, and transmit data in the same slot [106].

The IEEE 802.15.4 standard in our communication system specifies the PHYsical layer (PHY), and the Media Access Control (MAC) portion of the Data Link Layer (DLL) as shown in Figure 3.24.

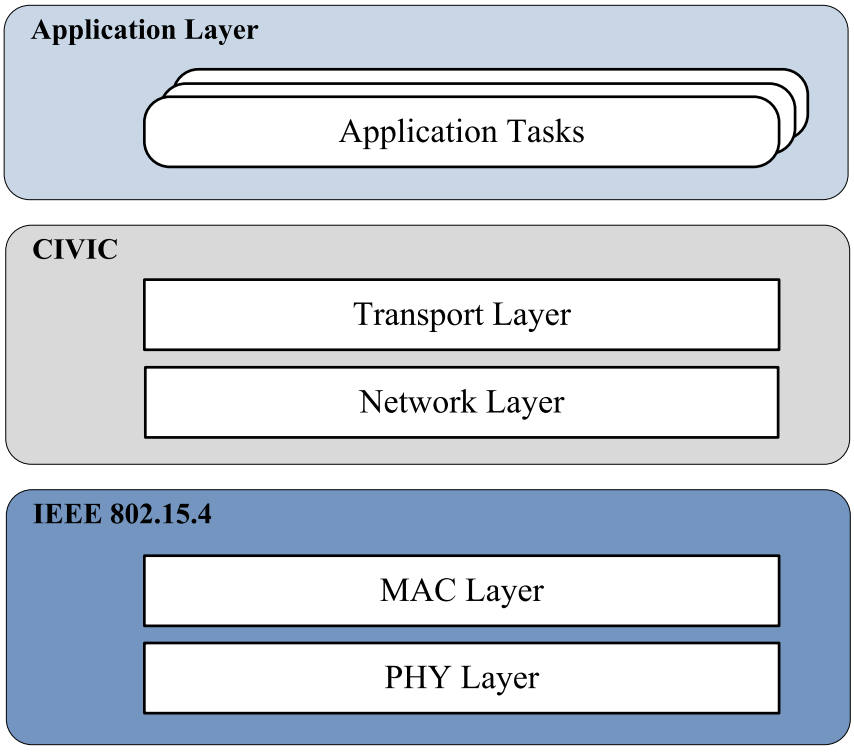


Figure 3.24: CIVIC and 802.15.4 standard

The original standard in 2003 [105] specifies two physical layers based on Direct Sequence Spread Spectrum (DSSS). The one we use is within the 2400 MHz frequency band. The standard raw data rate is 250 Kbps, which may be already enough for many low data rate network applications like WSNs. The MAC layer use CSMA/CA to avoid or reduce the

collision in the transmission process. More details and the evaluation results for 802.15.4 standard will be in Section 3.6.4, 3.6.5, and 3.6.6.

3.6 Theoretical Evaluations

3.6.1 Introduction

This section given a general evaluation for all the elements introduced in the previous sections, and see if these elements are suitable for our communication system. It introduces the factors that affect the message delays, gives the calculations for each delay element, and determines a theoretical result. The next sections will compare the theoretical result with the real-world experiment results. The factors and their notations are listed as follows:

- DF_{uart} : Inter-module serial communication by UARTs
- DF_{sw} : Software execution
- DF_{csma} : Channel access by CSMA/CA
- DF_{rf} : Over-the-air RF transmission
- DF_{ifs} : Inter-Frame Space (IFS)
- DF_{xbee} : XBee-PRO module operations

The calculations of DF_{uart} are based on [101] and [102]. The DF_{sw} is evaluated based on the software design in LRPC (Laboratoire Régional des Ponts et Chaussées) experiments, [107] and [108]. The DF_{csma}, DF_{rf} and DF_{ifs} are calculated according to IEEE 802.15.4 standard [105] used in XBee-PRO module. The DF_{xbee} is estimated via the in-lab results and [102].

The Figure 3.25 presents the periodic sequence of above factors in the experiments. From top to bottom, there are two general processes: the output and input of a message; the output and input of an acknowledgement. From left to right, there are three main hardware modules: MicroController Unit (MCU), XBee-PRO Module, and RF Antenna. The labels in figure are the notations of equations to calculate the delay factors, and they will be explains in the next sections.

3.6.2 Inter-module Serial Communication

The DF_{uart} from inter-module communication relates to the serial data rate and the number of characters needs to be transmitted.

The UART (Universal Asynchronous Receiver/Transmitter) enables the serial communications between microcontroller and XBee-PRO module. Although UARTs on both sides were registered to get 115200 baud, the actual bit rate on the XBee-PRO module is just 111111 bit/s [102], and the actual bit rate on the microcontroller is 111238 bit/s [101]. Therefore the former is used for the following calculations.

A character of ASCII text takes one octet (8 bits) to store, and it takes two extra bits (start and stop bits, no parity bit) to complete a serial transmission in the settings of our experiments. Thus the UART data rate, R_{uart}, is 111111/10/1000 = 11.1111 octet/ms. The transmitting period of one character, P_{char}, is $1/R_{uart}$ = 0.09 ms/octet.

Moreover, in the settings of a XBee-PRO module, the characters in a message from upper layer are not directly transmitted over-the-air. Instead, they are kept in a DI (Data In) buffer. The transmission will be started only if one of the following conditions is met [102]:

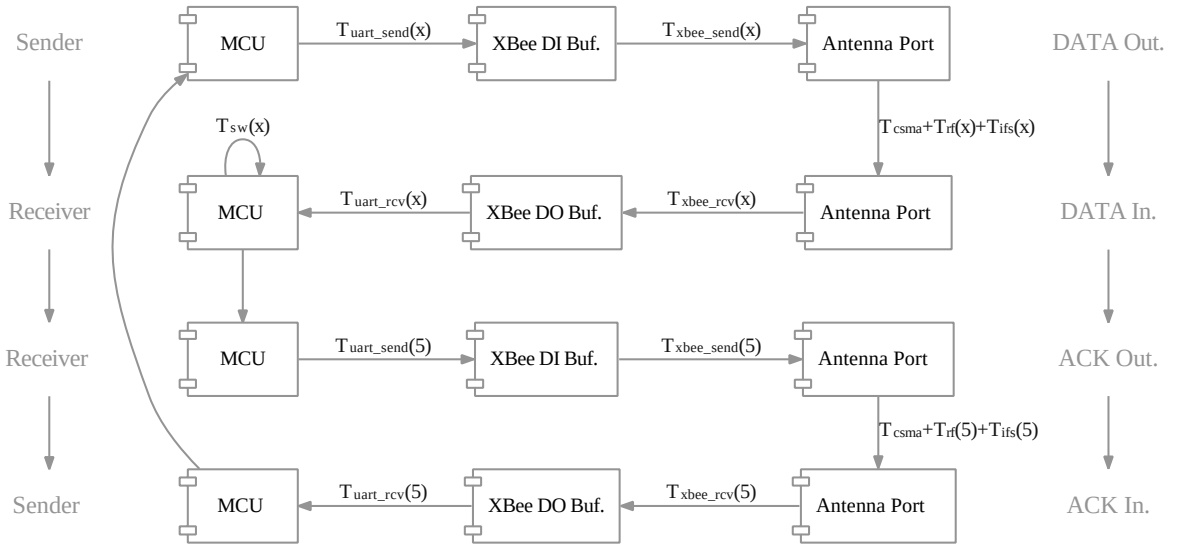


Figure 3.25: The periodic sequence of delay factors

1. The number of characters in DI buffer reaches the maximum payload of RF packets ($L_{payload}$).
2. There is no incoming character for a amount of time, which is determined by a RO (packetization timeout) parameter in XBee-PRO module. The unit of RO parameter is the transmitting period of one character. Our experiments set the RO parameter to be one. The period of RO delay, P_{ro}, before an over-the-air transmission is equal to P_{char}.
3. A command mode sequence is received. This mechanism was not used in the experiments.

Therefore, the delays of the receiver and the sender in millisecond unit are calculated as follows:

$$T_{uart\_send}(x) = \frac{x}{R_{uart}} + sgn(x \bmod L_{payload}) \times P_{ro} \tag{3.4}$$

$$T_{uart\_rcv}(x) = \frac{x}{R_{uart}} \tag{3.5}$$

where

x	:	the number of transmitted characters (octets)
$L_{payload}$	:	100 octets; the maximum payload of RF packet
R_{uart}	:	11.1111 octet/ms; the data rate of UART
P_{ro}	:	0.09 ms/msg; the RO parameter
$sgn(x)$	:	equals 1, if x>0; equals 0, if x=0; equals -1, if x<0

3.6.3 Software Execution

The CPU execution time is used to evaluate the elapsed time of software program. Since there are extra passing time in the accesses of memory and I/O, the DF_{sw} in the theoretical result should be slightly less than the actual elapsed time.

There are two parameters need to be known before given the equation of DF_{sw}, which are C_{cpi} and R_{mcu}. The C_{cpi} is the average CPI (Cycles Per Instruction) for ARM7 family. The R_{mcu} is the MCU clock frequency. The LiveNodes in the experiments are set to run in 48×10^6 Hz (cycles per second).

Where $Number_{instr}$ holds the instruction count in a program, the equation of DF_{sw} is given by:

$$\frac{Number_{instr} \times C_{cpi}}{R_{mcu}}$$

There are four major parts of instructions to be considered in a DATA/ACK software system: in a sender, the producing of DATAs and the processing for ACKs; in a receiver, the processing for DATAs and the producing of ACKs. But as previously mentioned, the starting point of calculating the delay in our sender program is after a DATA has been formed, so this part of instructions are not considered. Moreover, ACKs are fixed on the length in five octets, thus the software operations for ACKs are considered minor in their effects on delay (less than 100 μs).

Only the instructions relating the processing for DATAs on receiver are considered in calculations. They include the codes eliminating the 0x7D escape sequences in a DATA and calculating CRC. All together, they are counted to be C_{instr}. They are programmed in loops to processing characters in a DATA. Therefore, the effects for running these instructions are increased when the size of DATAs are grown.

As a result, we get the equation of $DF_{sw}(x)$ in millisecond unit:

$$T_{sw}(x) = \frac{C_{instr} \times C_{cpi} \times x}{R_{mcu}} \tag{3.6}$$

where

x	:	the number of characters in a message
C_{instr}	:	141 instructions/character; the number of instructions increased by per character
C_{cpi}	:	1.9 cycles/instruction; the average CPI for ARM7
R_{mcu}	:	48×10^3 cycles/ms; the MCU clock frequency

3.6.4 Channel Access by CSMA/CA

The unslotted CSMA/CA algorithm is shown in Figure 3.26, and it works as the following steps [1] :

1. If a device wishes to transmit a data frame, it first initializes the variables NB (Number of Backoff periods) to 0, and BE (Backoff Exponent) to macMinBE (the minimum value of the BE; range from 0 to 3). The macMinBE for the experiments was set to be one, because we assumed that there would not be much network interference. If macMinBE equals zero, the next step is disable.

[1]The acronyms and abbreviations in Sections 3.6.4, 3.6.5 and 3.6.6 are kept the same as in IEEE 802.15.4 standard [105].

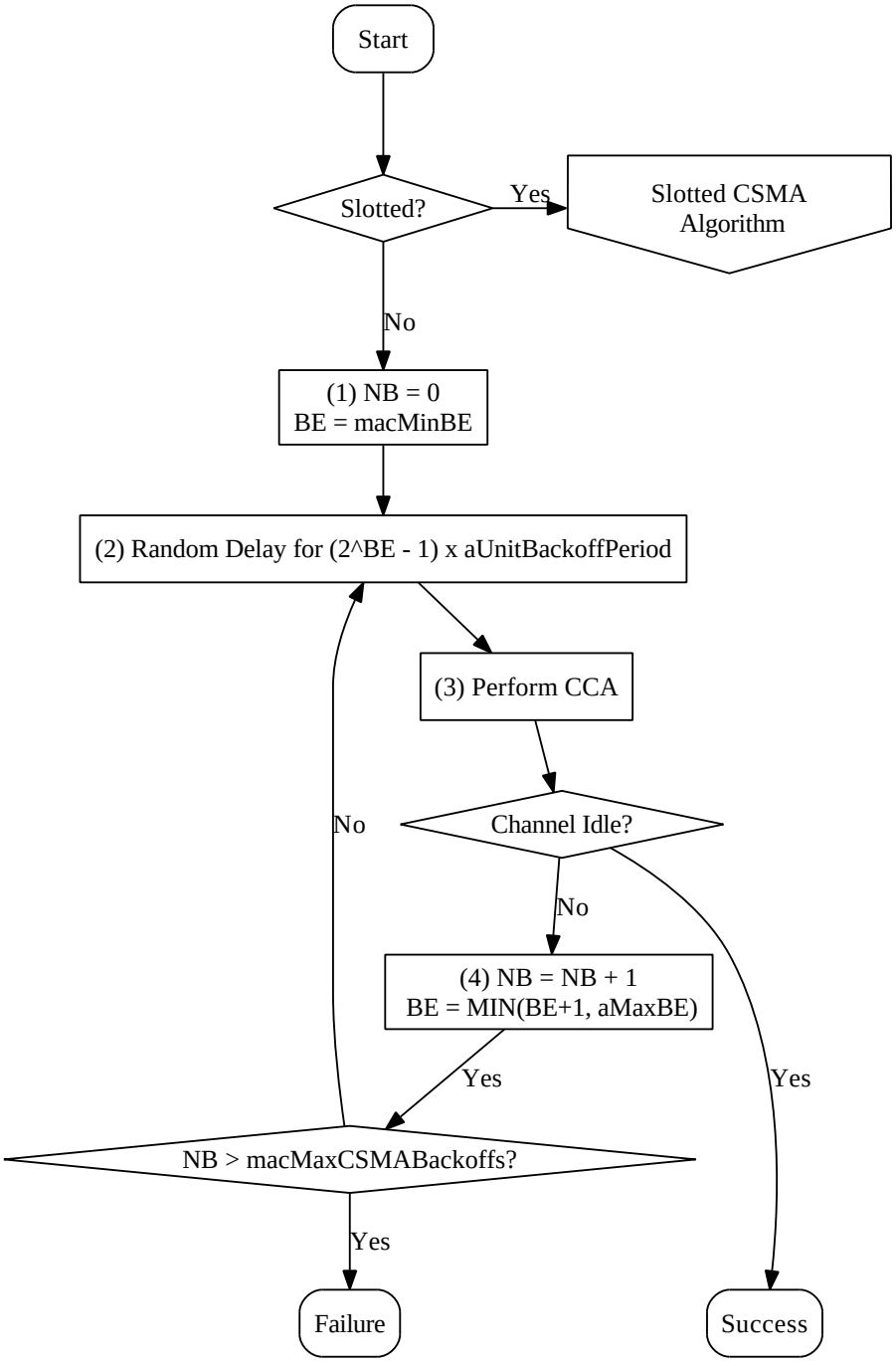


Figure 3.26: The non-beacon unslotted CSMA/CA algorithm

2. The first step in the unslotted CSMA/CA iteration is a random backoff. For the unslotted CSMA-CA, the backoff starts immediately. The delay of random backoff ranges from 0 to (2^{BE} –1) × aUnitBackoffPeriod. The $aUnitBackoffPeriod$ (or Sym_{bf}) is the time period represented as the number of symbols, and it is a MAC sublayer constant.

3. After a random backoff, the Clear Channel Assessment (CCA) is performed to check whether the channel is available for transmission. If the detected energy on the channel is below the CCA threshold, the frame is transmitted; otherwise, it follows the step 4. The duration of CCA detection is Sym_{cca}.

4. If the channel is not idle, both BE and NB are incremented by one. The value of BE is up to aMaxBE (the maximum value of the BE; aMaxBE equals 5 and it remains constant). The maximum value of NB is $macMaxCSMABackoffs$ (range from 0 to 5; default value is 4). If NB exceeds $macMaxCSMABackoffs$, the unslotted algorithm terminates and a frame is lost; if not, it returns to the second step.

Based on the steps detailed above, the best-case channel access time is known as 0.128 ms. The worst-case channel access time is 8.832 ms, and its calculation is as:

$$\sum_{n=1}^{4}((2^n-1)\times Sym_{bf}+Sym_{cca})\times P_{sym}$$

For the theoretical result, we assume the channel accesses are always successful at the first time, and the delay in second step equals the mean delay of random backoff range. Therefore, the timing caused by DF_{csma} is calculated as follows:

$$\begin{aligned} T_{csma} &= (\frac{Sym_{bf}}{2}+Sym_{cca})\times P_{sym} \\ &= 0.288\ ms/frame \end{aligned} \tag{3.7}$$

where

Sym_{bf}	:	20 symbol periods; the number of symbols forming the basic time period
Sym_{cca}	:	8 symbol periods; the duration of CCA detection
P_{sym}	:	0.016 ms/symbol; the symbol period in 2.4 GHz band

3.6.5 Over-the-air RF Transmission

The DF_{rf} per frame relates to the RF raw data rate and the frame overhead (or protocol overhead). As previously mentioned, acknowledgements in 802.15.4 standard are not used in the broadcast mode for our experiments, the timings of acknowledgement and turn-around are omitted in the following calculations.

The raw data rate of the 802.15.4 standard in 2400 MHz band is 250 Kbps. A octet takes 8 bits to be completely RF transmitted, thus the RF raw data rate R_{rf} is 250000/8/1000 = 31.25 octet/ms.

If the raw data rate is fixed, the frame overhead determines the information rate, which normally must be lower than the raw data rate. For example, the ZigBee protocols on top of 802.15.4 standard sometimes only provide a peak information rate of 120 Kbps [109]. In the following, we will describe the frame overhead based on the settings of this experiment.

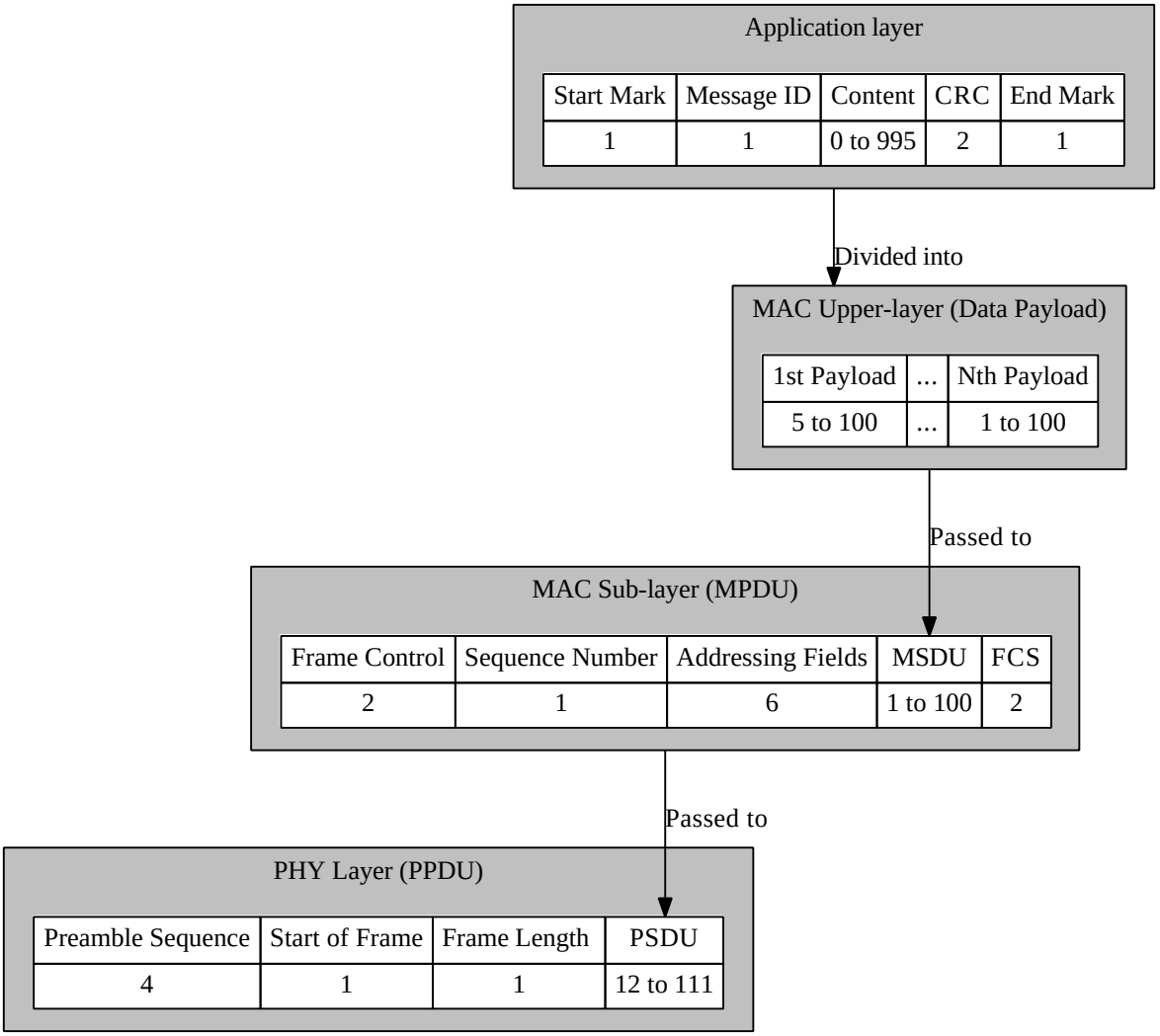


Figure 3.27: IEEE 802.15.4 Frame Format (the unit of size is octet)

The operations of 802.15.4 standard relate to the frame overhead is shown in Figure 3.27. The number 995 in the application layer is the maximum data size set in the experiments of the latter sections, and the data size can be configured by users.In the tables of sub-graphs, the first row shows the names of fields, the second row shows the sizes of fields.

The frame structure of physical layer (PHY) is on the bottom of Figure 3.27. Above the PHY, it is the Medium Access Control (MAC) sublayer. On top of this figure, the message from application layer is divided into the data frame payloads of MAC sublayer.

The Phy Service Data Unit (PSDU) is the data frame payload of PHY. A PSDU is prefixed a PHY frame overhead to form a Phy Protocol Data Unit (PPDU).

The data frame payload of MAC sublayer is referred to as the Mac Service Data Unit (MSDU). The size of MSDU is named as $Size_{msdu}$. A MSDU and a MAC frame overhead

together form a Mac Protocol Data Unit (MPDU) that will be passed to PSDU. The size of MAC frame overhead is a variable because it contains changeable addressing fields from 4 to 20 octets. Our experiments use the 16-bit short addressing with intra-PAN broadcast network, therefore the source PAN ID shall not be present in addressing fields (pp.111-pp.115 of [105]). The addressing fields contain only three 2-octet fields including a destination PAN ID, a destination address, and a source address. Together, the addressing fields are 6 octets. Therefore, the L_{macOH} equals to 11 octets.

The maximum size of PSDU, $aMaxPHYPacketSize$, in [105] is 127 octets, thus the maximum value of $Size_{msdu}$ should be aMaxPHYPacketSize - L_{macOH} = 127 - 11 = 116 octets. However, the $L_{payload}$ in Section 3.6.2 is only 100 octets, thus the $L_{payload}$ is used as the maximum value of $Size_{msdu}$.

As a result, the factor DF_{rf} per frame in millisecond unit can be formulated as:

$$T_{rf}(x) = \frac{x + L_{phyOH} + L_{macOH}}{R_{rf}} \tag{3.8}$$

where

x	:	the size of MSDU ($Size_{msdu}$), and $0 < x \leq L_{payload}$
L_{phyOH}	:	6 octets; the length of PHY frame overhead
L_{macOH}	:	11 octets; the length of MAC frame overhead
R_{rf}	:	31.25 octet/ms; the RF raw data rate

3.6.6 Inter-frame Space

To allow MAC sublayer process data received by the PHY layer in 802.15.4 standard, the transmission of data frame is followed by an inter-frame separation period called Inter-frame Space (IFS). This delay is generated by the XBee-PRO module after sending a data frame. When the size of a message is over $L_{payload}$ (a data frame unit), the DF_{ifs} needs to be considered. This section will briefly introduce DF_{ifs}. In an actual transmission, this factor normally overlaps by the DF_{xbee}, which will be introduced in the next section.

The size of MPDU (L_{mpdu}) designate the IFS period to be either short IFS or long IFS. If L_{mpdu} is smaller than or equal to aMaxSIFSFrameSize (18 octets), aMinSIFSPeriod (or S_{sifs}) is used to form short IFS; otherwise, aMinLIFSPeriod (or S_{lifs}) is used to form long IFS.

The L_{mpdu} can be calculated as $Size_{msdu} + L_{macOH}$, thus the variable $Size_{msdu}$ is used to compare with $aMaxSIFSFrameSize - L_{macOH}$. A new constant, L_{sifs}, represents this limitation:

$$L_{sifs} = aMaxSIFSFrameSize - L_{macOH} = 7\ octets \tag{3.9}$$

The time period of DF_{ifs} is designated as follows:

If $x \leq L_{sifs}$ *then,*

$$T_{ifs} = T_{sifs} = S_{sifs} \times P_{sym} = 0.192\ ms/frame$$

else,

$$T_{ifs} = T_{lifs} = S_{lifs} \times P_{sym} = 0.64\ ms/frame$$

Or, the following equation is in the grammar of Gnuplot:

$$T_{ifs}(x) = (x <= L_{sifs} \; ? \; S_{sifs} \; : \; S_{lifs}) \times P_{sym} \tag{3.10}$$

where

x	:	represents $Size_{msdu}$, and $0 < x \leq L_{payload}$
L_{sifs}	:	7 octets; the maximum size to use short IFS
S_{sifs}	:	12 symbol periods
S_{lifs}	:	40 symbol periods

3.6.7 XBee-PRO Module Operations

Except the factors that have been described in Section 3.6.2, it can be assumed that there should be DF_{xbee} from the operations inside a XBee-PRO module as shown in Figure 3.28.

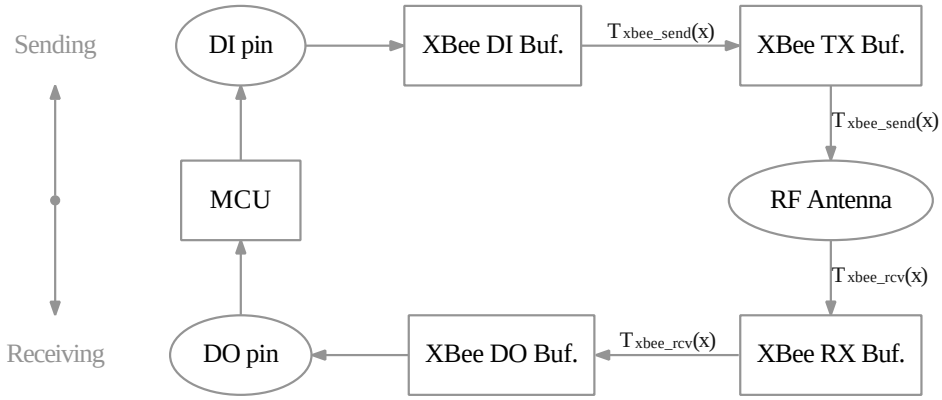


Figure 3.28: Communication Mechanisms of the XBee-PRO module

The first source, named DF_{xbee_send}, causes the delay when sending data. When the serial data from the MCU enters the XBee-PRO module, it enters the Data In (DI) buffer through the DI pin (pin 3), then passes the TX buffer to the RF antenna. The second source, DF_{xbee_rcv}, causes the delay when receiving data. In this process, the RF data passes the RF antenna to the RX buffer, and leaves the Data Out (DO) buffer to MCU through the DO pin (pin 2). By detecting the signals from the RF antenna, and from the DI/DO pins on a pair of communicating XBee-PRO modules, we can estimate DF_{xbee_send} and DF_{xbee_rcv}.

We used a digital serial analyzer (Tektronix DSA71604) in this experiment to estimate these two factors. Because the XBee-PRO module works according to the data frame mechanism of the 802.15.4 standard[105], the estimated values are based on the data frame unit.

The Figure 3.29 is captured when testing the DF_{xbee_send}. The first waveform (the channel one on top) is output from the DI Pin of the XBee-PRO module on a sender. The bottom waveform (channel two) is the output of the RF antenna. The Δt represents the time difference of $t1$ and $t2$; in other words, the value of DF_{xbee_send} plus DF_{csma}.

The value of Δt is about 2.18 ms when $Size_{msdu}$ (see Section 3.6.5) is in the minimum size of five octets, and it linearly increases with $Size_{msdu}$. When $Size_{msdu}$ equals $L_{payload}$, the value

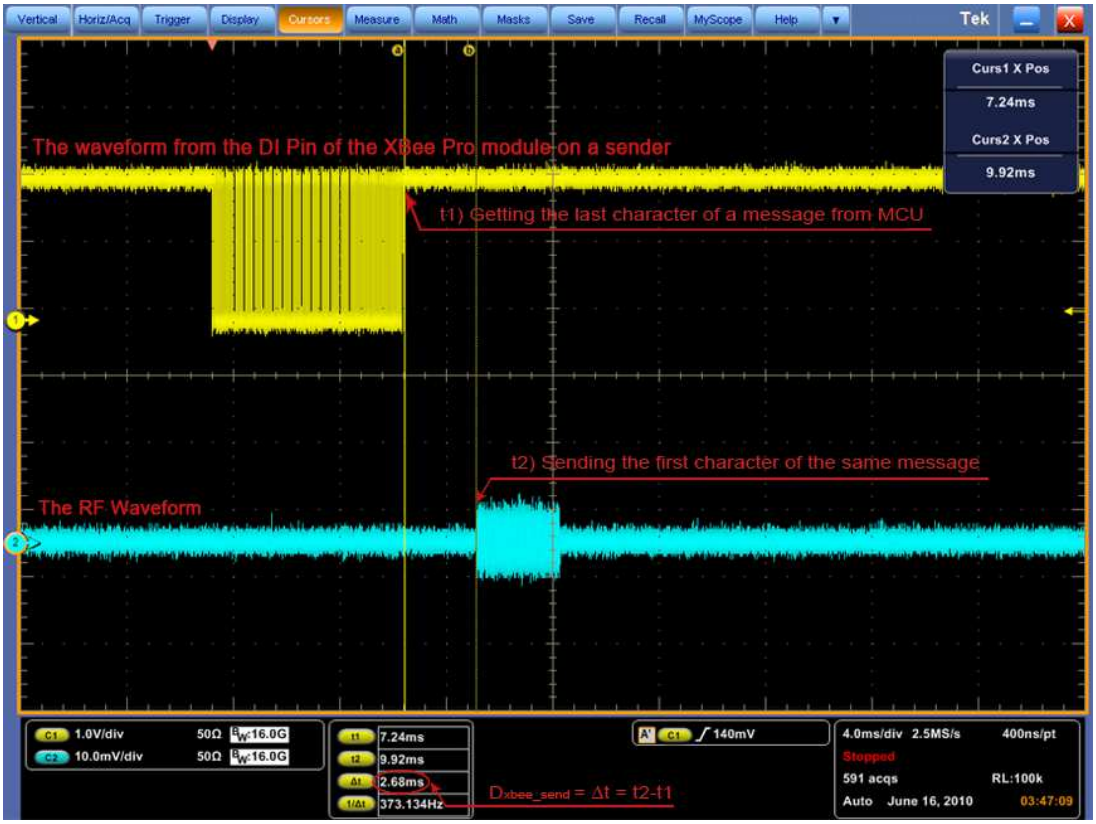


Figure 3.29: The waveforms in the sending process

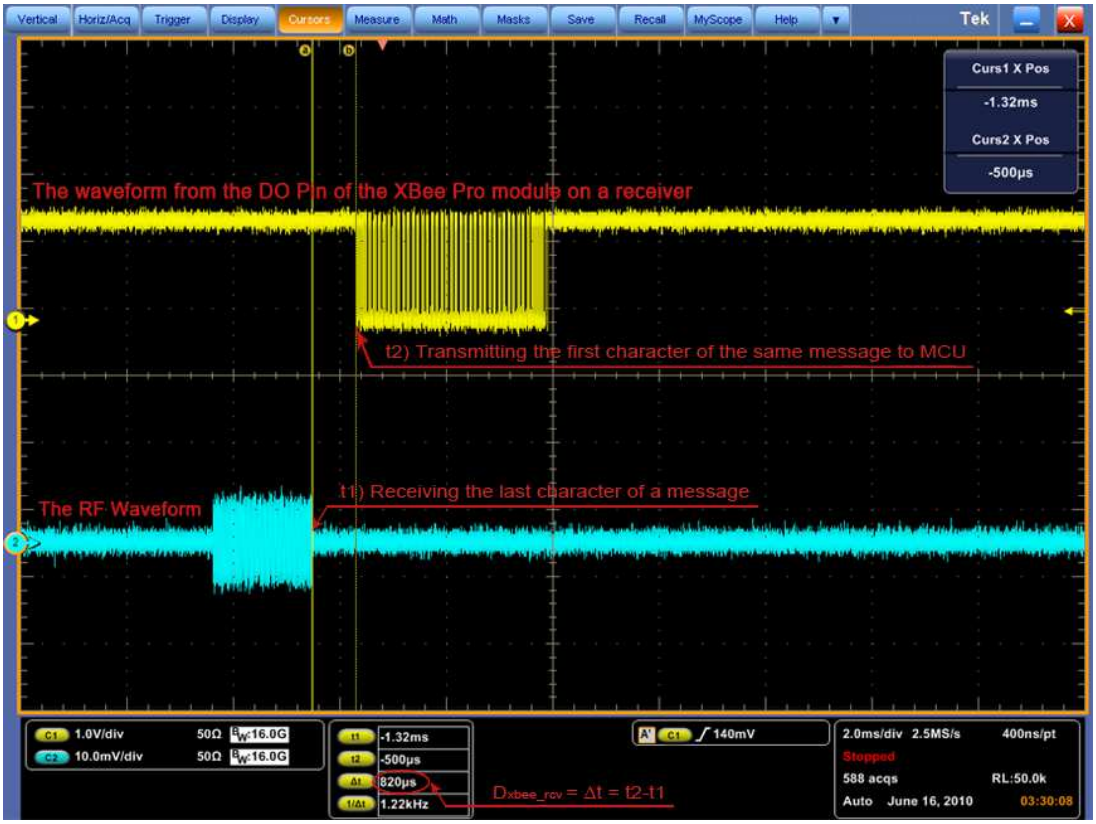


Figure 3.30: The waveforms in the receiving process

of Δt reaches a peak of 2.92 ms. Therefore, a linear equation is used to relate the $Size_{msdu}$ and the DF_{xbee_send} in millisecond unit as:

$$T_{xbee\_send}(x) = mx + a = 0.0078x + 1.853 \tag{3.11}$$

where

x : represents $Size_{msdu}$, and $0 < x \leq L_{payload}$
m : $(2.92 - 2.18)/(100 - 5) = 0.0078$
a : $(2.18 - 5 \times (2.92 - 2.18)/(100 - 5)) - T_{csma} = 1.853$

The method to calculate DF_{xbee_rcv} is the similar with DF_{xbee_send} as shown in Figure 3.30. The first waveform in the figure shows the signals from the DO Pin of the XBee-PRO module on a receiver. The second waveform is still the RF output. In this case, the DF_{xbee_rcv} generally linearly increases with $Size_{msdu}$ from 0.56 ms to 1.12 ms; therefore, a raw equation is given:

$$T_{xbee\_rcv}(x) = nx + b = 0.0059x + 0.531 \tag{3.12}$$

where

x : represents $Size_{msdu}$, and $0 < x \leq L_{payload}$

When the size of data frame is greater than $L_{payload}$, the DF_{ifs} need to be taken in account. Because the DF_{xbee_rcv} is overlap with DF_{ifs}, the greater value of these two delays should be used in calculations. In the settings of LRPC experiments, the DF_{xbee_rcv} is always the greater one.

3.6.8 Evaluation Methods

An extra experiment was conducted to test the methods for calculating the delay in a data frame transmission. Besides, this experiment tries to evaluate the DF_{xbee} equations in previous section.

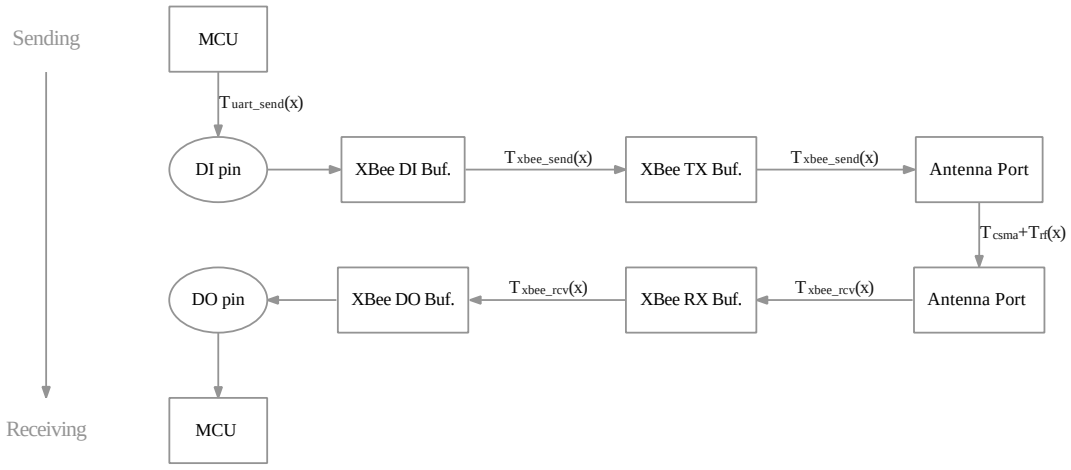


Figure 3.31: Mechanisms of the XBee-PRO module communications

The experiment gives an equation to calculate the sum of delays related to $Size_{msdu}$ by detecting the time difference of signals from two source: the DI pin and the DO pin on a pair

of communicating XBee-PRO modules (Figure 3.31). This sum will be compared with the sum calculated by previous theoretical equations.

Figure 3.32: Delays when sending and receiving a message in five octets

The Figure 3.32 is captured from another machine, a digital phosphor oscilloscope (Tektronix TDS510413), during experiments. It shows a sample in the starting stage when sending DATA/ACK in five octets. From top to bottom, the waveforms are output from four pins on a pair of XBee-PRO modules. The Δt on the upper-right corner is a time difference value between:

- The first character in a message passes through the DI pin of the XBee-PRO module on a sender.
- The same character passes through the DO pin of the XBee-PRO module on a receiver.

The Δt can represent the sum of delays labeled in Figure 3.31:

$$\Delta t = T_{uart\_send}(x) + T_{xbee\_send}(x) + T_{csma} + T_{rf}(x) + T_{xbee\_rcv}(x)$$

The value of Δt in this experiment fluctuates around 3.9 ms when $Size_{msdu}$ is in five octets; and basically, it linearly increases with $Size_{msdu}$ until $Size_{msdu}$ equals $L_{payload}$. The peak value of Δt is 16.9 ms. Thus, a linear equation is given to calculate Δt:

$$\Delta t = 0.137x + 3.216$$

where

x : represents $Size_{msdu}$, and $0 < x \leq L_{payload}$

As shown in Figure 3.33, when the size of data frame is increased from 0 octet to 100 octets, the maximum difference between the oscilloscope result and the theoretical result is less than 0.116 ms, and the minimum difference is 0.00021 ms. Thus, the calculation methods in previous sections are considered to be correct in a data frame unit. The next section will extend the methods to a message in any size.

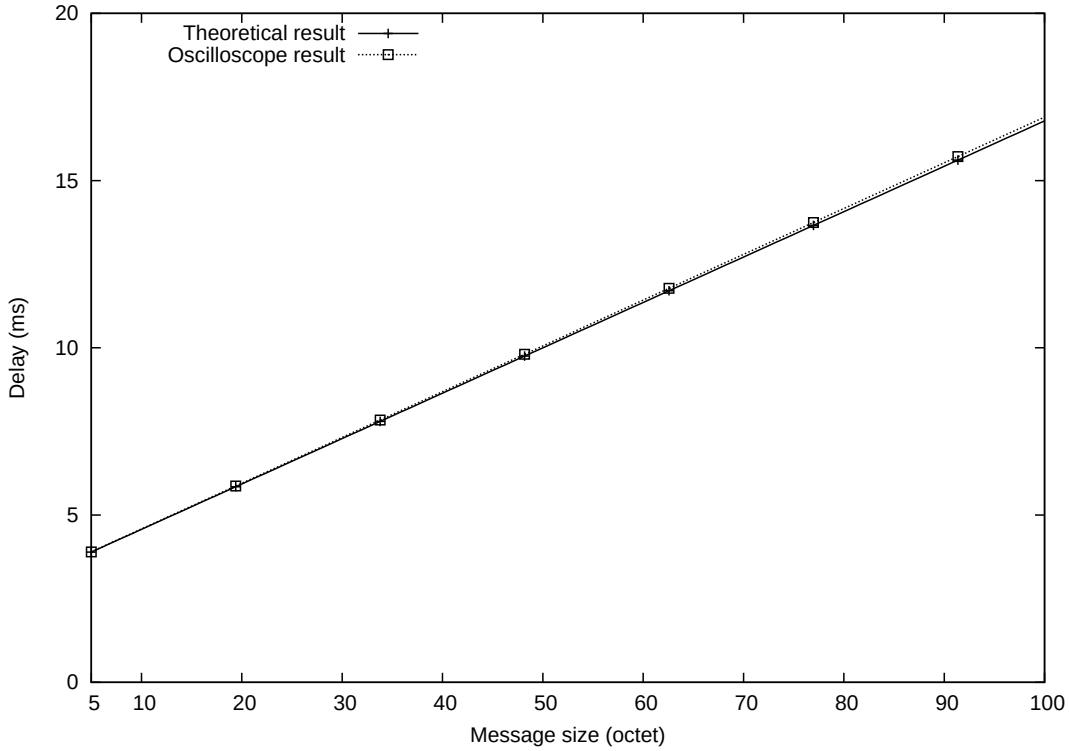


Figure 3.33: Comparing the oscilloscope result with the theoretical result

3.6.9 Theoretical result

The previous sections have introduced the major delay factors and their independent equations for durations, but the analysis so far has not considered the timing sequence of these factors and the bottleneck in transmissions. The remainder of this chapter will put the pieces together and give the theoretical sum of delays in sending a DATA and its ACK as T_{sum}:

$$T_{sum} = T_{data}(x) + T_{ack} \tag{3.13}$$

After data frames are divided from a message, the sending of data frames are placed in two situations based on the message size:

Firstly, when the message size is less or equal to $L_{payload}$, as the first situation in Figure 3.34, the delay factors happen separately and in succession. The T_{sum} is the totality of Eq. 3.14 and Eq. 3.15.

$$\begin{aligned} T_{data}(x) &= T_{uart\_send}(x) + T_{xbee\_send}(x) + T_{csma} + T_{rf}(x) \\ &\quad + T_{xbee\_rcv}(x) + T_{uart\_rcv}(x) + T_{sw}(x) \end{aligned} \tag{3.14}$$

where

x : the size of DATAs, and $0 < x \leq L_{payload}$

$$\begin{aligned} T_{ack} &= T_{uart\_send}(5) + T_{xbee\_send}(5) + T_{csma} + T_{rf}(5) \\ &\quad + T_{xbee\_rcv}(5) + T_{uart\_rcv}(5) \end{aligned} \tag{3.15}$$

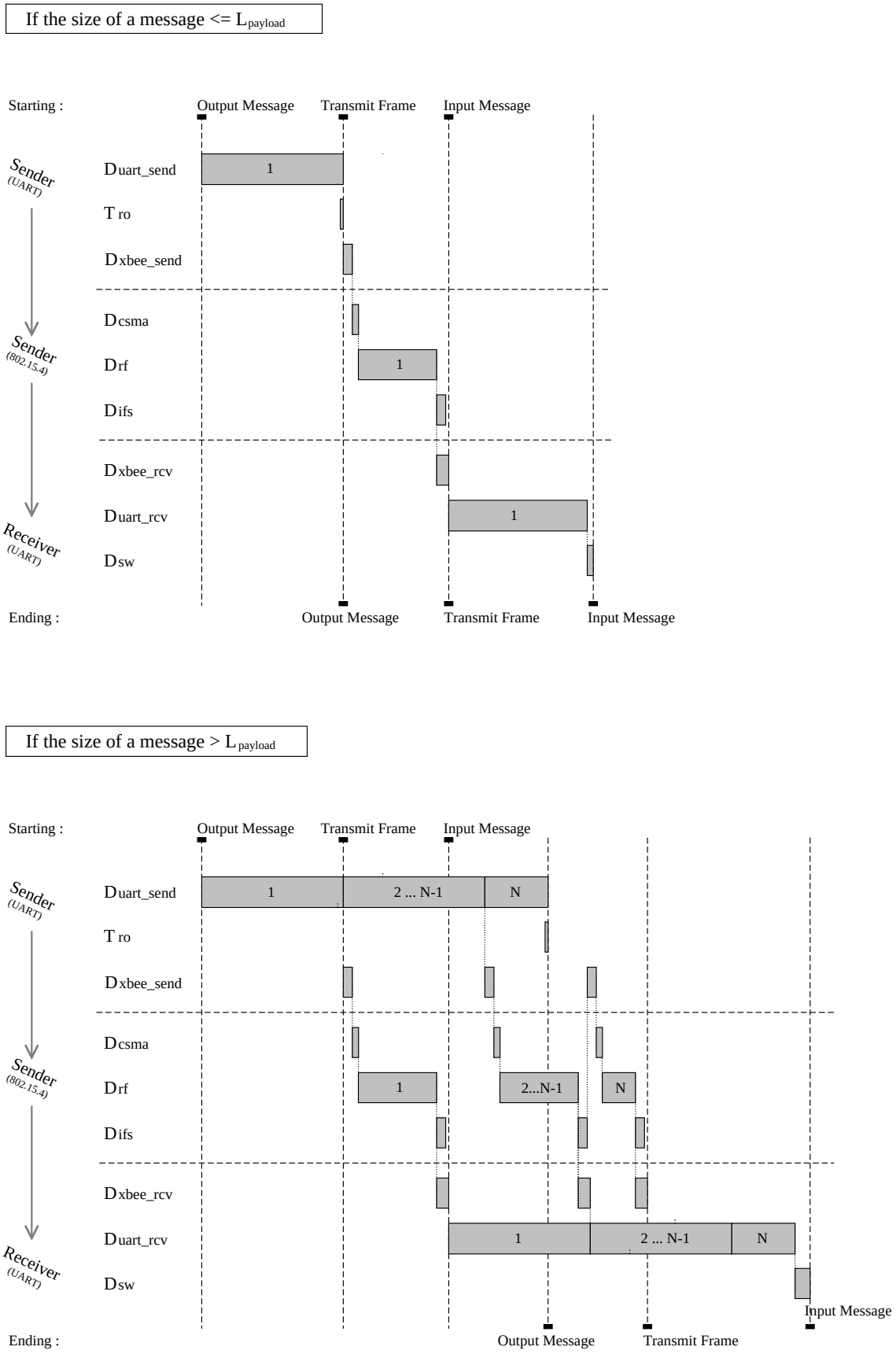


Figure 3.34: The timing sequence of delay factors in sending data frames

$T_{data}(x)$ is for calculating the delay in sending DATAs, and its result is increased with the size of DATAs. T_{ack} is for calculating the delay in sending ACKs. Because ACKs are always five octets, the delay in sending ACKs is constant despite the changes in the size of DATAs.

Secondly, when the message size is greater than $L_{payload}$ as the second situation in Figure 3.34, the delay factors start to work in parallel with each other. Moreover, if the data rates R_{uart} and R_{rf} are different, a transmission bottleneck will happen in the lower speed side. The delay can be calculated as the sum of following items:

1. When messages are transmitted inside the sender, the major delay factor is DF_{uart}. The equation $T_{uart\_send}(x)$ is used for the first payload in a message.

2. Between two XBee-PRO modules, when messages are transmitted over-the-air, the delay is comprised of DF_{xbee}, DF_{csma}, and DF_{rf}. If R_{uart} is greater than R_{rf}, the delay here relates to the number and size of data frames. But if R_{rf} is greater than R_{uart} (as in LRPC experiments), the equations $T_{xbee\_send}(x)$, T_{csma}, T_{rf}(x) and $T_{xbee\_rcv}(x)$ are only used for calculating the first data frame (payload) from a message.

3. The delays inside a receiver are from DF_{uart} and DF_{sw}. The calculations relate to the number of transmitted characters; in other words, the whole size of messages.

The T_{ack} under the second situation is the same as the first one. The delay in sending DATAs is calculated as Eq. 3.16:

$$\begin{aligned} T_{data}(x) &= T_{uart\_send}(L_{payload}) + T_{xbee\_send}(L_{payload}) + T_{csma} \\ &\quad + T_{rf}(L_{payload}) + T_{xbee\_rcv}(L_{payload}) + T_{uart\_rcv}(x) \\ &\quad + T_{sw}(x) \end{aligned} \tag{3.16}$$

where

x : the size of DATAs, and $x > L_{payload}$

The last step in calculations is to take in account the software turnaround time of two milliseconds as described in Section 3.7.1.4, thus there are about 96 milliseconds that can be actually applied by each communication in a sub-test. After 100 milliseconds, all messages are marked as lost ones. In 96 milliseconds, the size of DATAs that should be communicated correctly is less than or equal to 770 octets. But when the size of DATAs is close to 770 octets, the last ACKs begin to get collisions with the next DATAs. Therefore, between 96 and 100 ms, the T_{csma} will not be a constant and there will be a sharp rise in the loss rates. As a result, we define the "effective" size of DATA as 760 octets.

The calculations are shown in Figure 3.35. The line "DATA" combines the results from Eq. 3.14 and Eq. 3.16. The line "ACK" represents the constant T_{ack} from Eq. 3.15. The size of DATAs relates to T_{sum} as the line of "DATA+ACK". Moreover, the figure includes a comparison between the theoretical result and a sample result from a sub-test in the LRPC experiment Section 3.7. The selected sub-test is under the best communication conditions: without fog, in the minimum transmit distance, and with the maximum transmit power. Two lines closely match each other. It indicates the theoretical result reflect the real-world results.

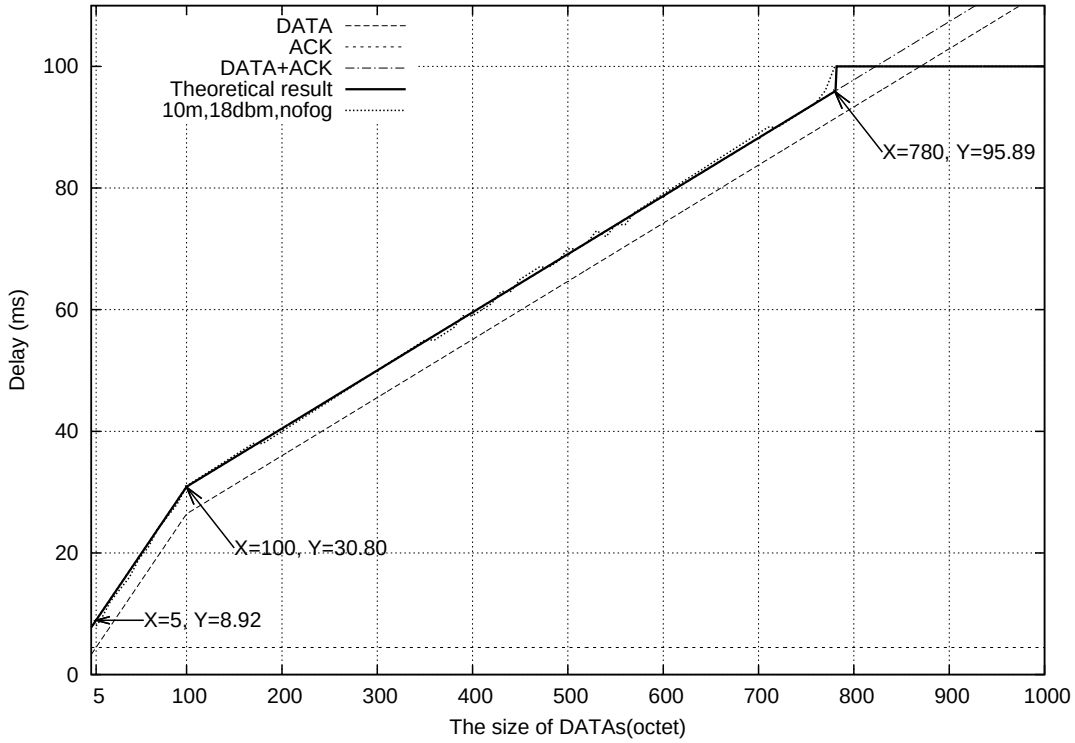


Figure 3.35: The theoretical delay

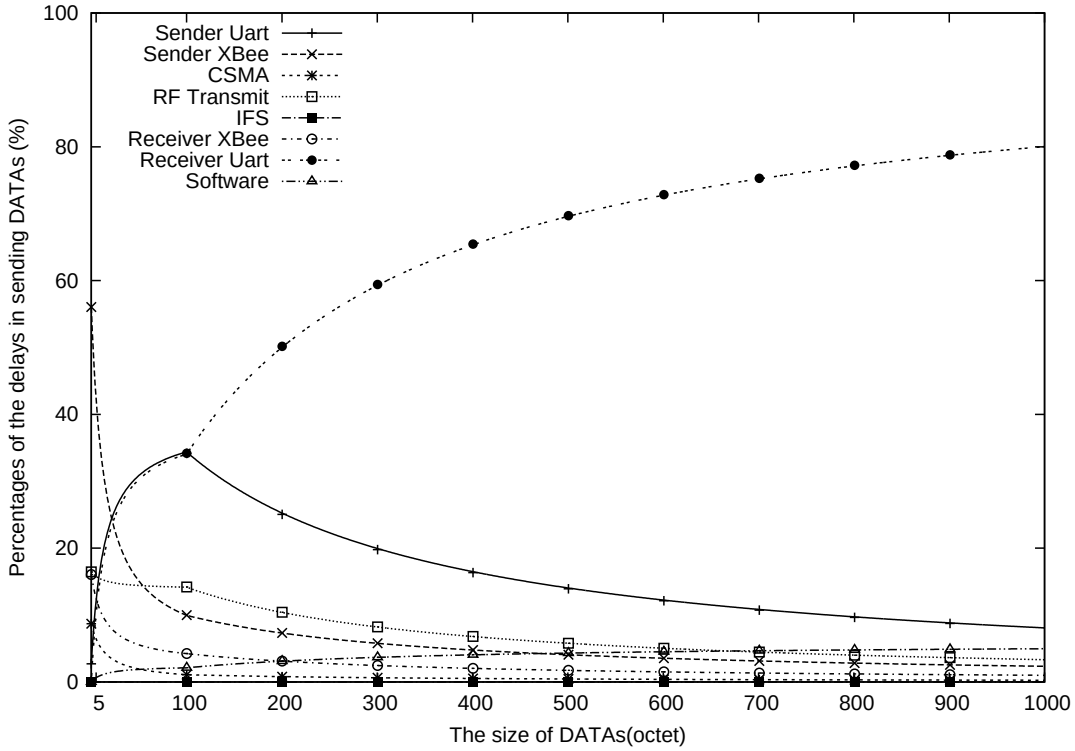


Figure 3.36: The percentage rates of delays

The Figure 3.36 shows the proportions of delays in $T_{data}(x)$ related to the size of DATAs,and this figure can be seen as the relation between the proportions of delay factors and the size of messages. The DF_{xbee} and DF_{uart} are the major delay factors respectively when the size of DATAs increase from 0 to $L_{payload}$. Although in general, the delay is decreased when the message size is reduced. When the message sizes is close to the minimum size like ACK (five octets), reducing the message size will not significantly shorten the delay because the DF_{xbee} exists. Other than that, the DF_{uart} contributes the most affection in the overall delays, especially after the message size is over $L_{payload}$.

3.7 LRPC Experiments

3.7.1 Introduction

The purpose of this experiment is to evaluate the message delay and loss rate of 802.15.4 network on the effects of fog, transmit power, and distance. The experiment results will be used to improve the design of the CIVIC protocol. It is an experiment conducted at LRPC (Laboratoires Régionaux des Ponts et Chaussées) in Clermont-Ferrand.

3.7.1.1 LRPC Test Center

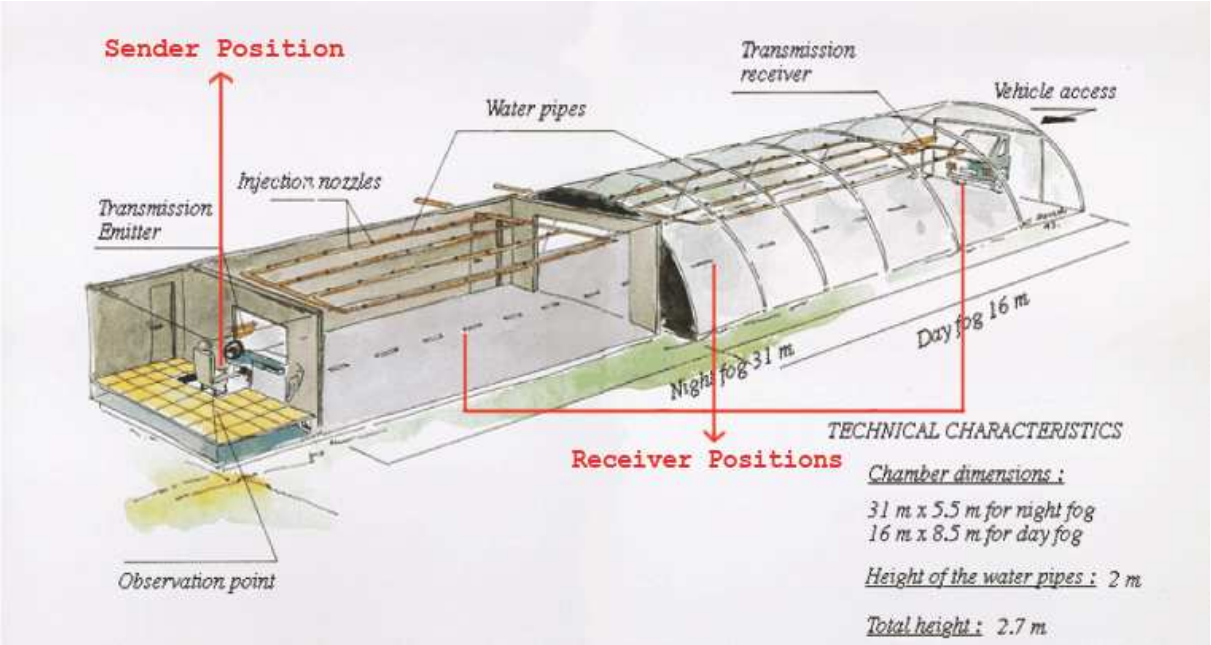


Figure 3.37: The fog chamber of LRPC

The fog chamber at LRPC is a test center to study the factors that affect visibility in fog. The chamber has two parts: an observation station and a night room (31m×5.5m×2.7m) as shown in Figure 3.37 and Figure 3.38.

Half of the night room on the far side of the observation station can be set to the daytime condition, thus it is also called as a day room. The day room in our experiments was covered by black plastic to simulate the nighttime condition.

The fog chamber generates controllable fog by atomizing water at high pressure in diffuser. The produced droplets have a mean diameter of 10 μm, substantially the same as natural fog. The density of fog is measured according to visibility distance (the unit is meter). In the figures of this thesis, the "vis" is used to represent the visibility distance in the unit of meter.

3.7.1.2 Hardware and Software

There were two pairs of LiveNode sensors used in LRPC experiment. Each pair contains a sender and a receiver. LiveNodes are equipped with an ARM7 microcontroller and a

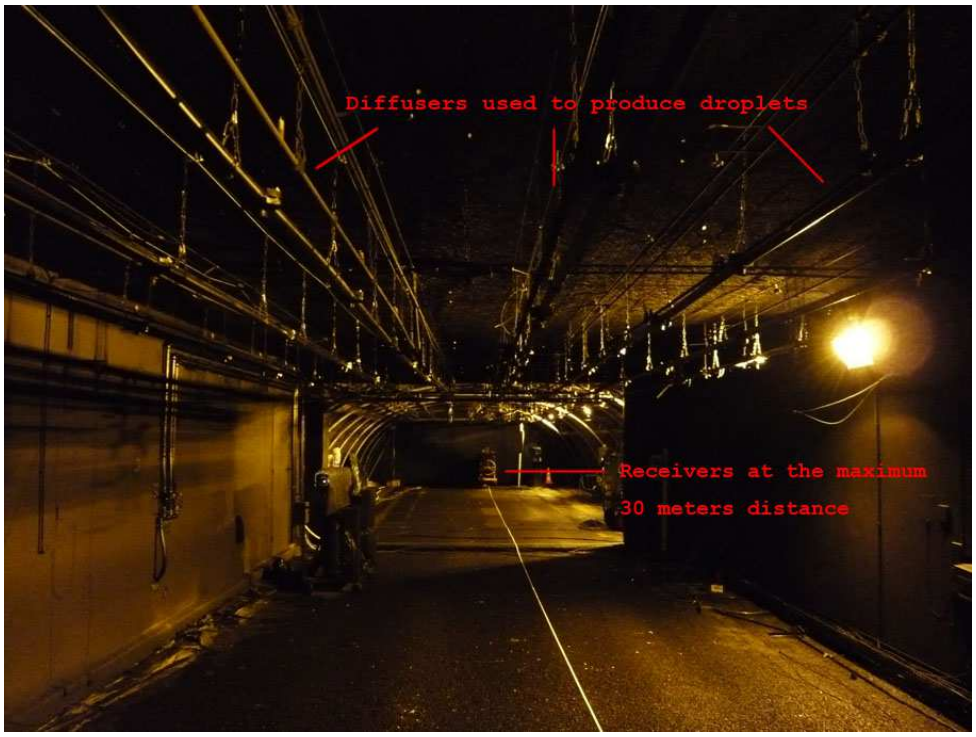


Figure 3.38: The environments of the night room

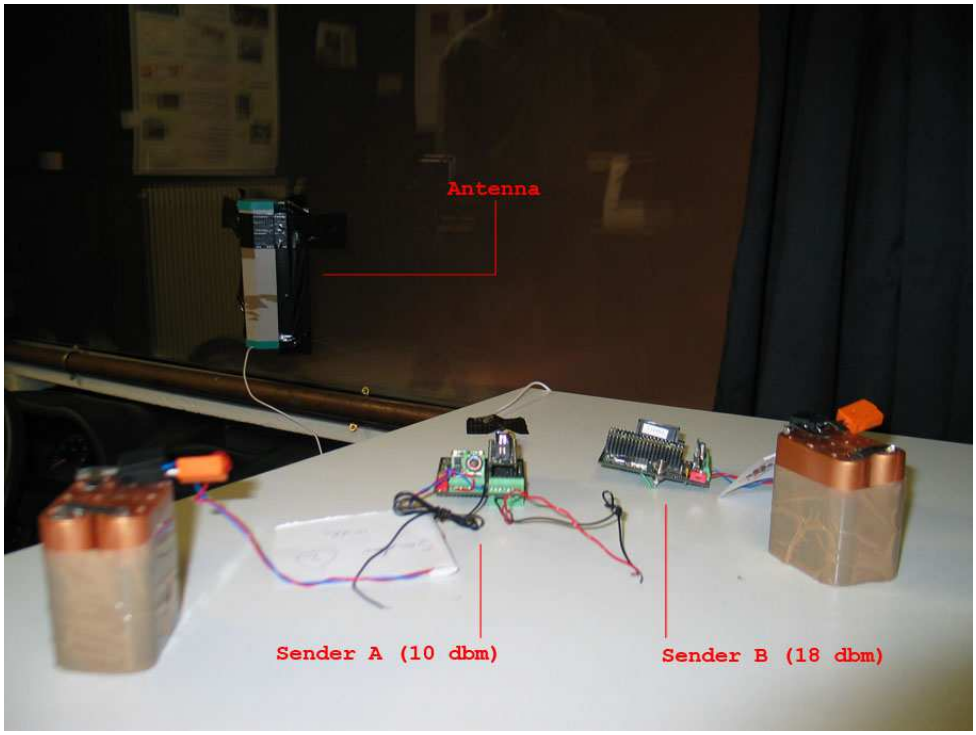


Figure 3.39: The sender LiveNodes at the observation station

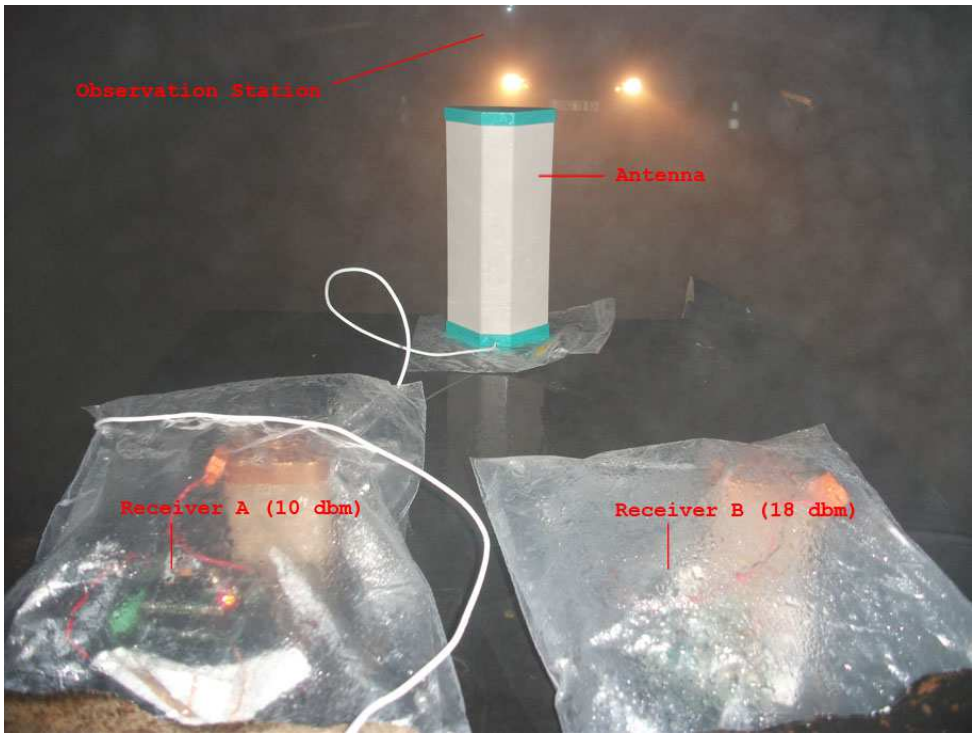


Figure 3.40: The receiver LiveNodes at the night room during experiment

MaxStream XBee-PRO module. The XBee-PRO modules on sender and receiver in the same pair were set to the same RF (Radio Frequency) transmit power of either 10 dBm or 18 dBm. The sleep mode in all XBee-PRO modules was disabled. Each LiveNode was powered by a 9V battery.

The senders were in the observation station as shown in Figure 3.39. The receivers were at night room of fog chamber as shown in Figure 3.40. Therefore, the maximum tested wireless transmit distance in our experiment is 30 meters as shown in Figure 3.38.

In observation station, a ZigBee station was connected to a laptop computer for monitoring network status and collecting experiment result.

The software on LiveNode is mainly written in C language. The PC serial port monitoring software is Terminal v1.9b, which monitors the network messages from the ZigBee station by a serial port. A C++ program is used to process original results. Gnuplot and Excel are used to do further processing and plot figures.

3.7.1.3 802.15.4 Network

The MaxStream XBee-PRO module on LiveNode adopts IEEE 802.15.4 standard published in October 2003 [105]. The IEEE 802.15.4 standard is a Low-Rate Wireless Personal Area Network (LR-WPAN) standard supporting a maximum raw data rate of 250 Kbps for the 2400 MHz band.

The collision avoidance mechanism in 802.15.4 standard is CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) protocol. Since only two XBee-PRO modules were involved in a sub-test, we assumed that there would not be much network interference, thus the experiment was operated in a non-beacon enabled network using unslotted CSMA/CA protocol.

We consider the results of this experiment would be used in the MAC adapter layer of the CIVIC protocol, thus the experiment did not adopt the acknowledgement mechanism from 802.15.4 system provided by XBee-PRO module. The C program running on LiveNodes generated an acknowledgement and turnaround time.

Besides, the network was configured to operate in the intra-PAN (Personal Area Network) broadcast mode as the same networking mode used in the experiments of the CIVIC protocol. A broadcast message in 802.15.4 standard is sent only once, and it cannot be acknowledged or resent.

3.7.1.4 Scenarios

There were 24 sub-tests in our experiment. Each sub-test was done in a scenario combined the changes of three factors including density of fog, transmit power, and transmit distance. The experiments have tested four degrees of density of fog, including zero fog, 20 meters, 10 meters, and the maximum 5 meters visibility distance. The transmit distance was 10 meters, 20 meters or 30 meters. The transmit power was either 10 dBm or 18 dBm. All together, there are 24 scenario combinations [2].

In each sub-test, only one pair of LiveNodes was activated. A LiveNode acted as a sender and it transmitted data messages (DATA) in increasing sizes from 5 bytes to 1000 bytes. The first DATA in a sub-test was 5 bytes, the second one was 10 bytes, and then 10 bytes was increased each time until reaching 1000 bytes. Another LiveNode acted as a receiver, and it acknowledged DATA from sender. The size of an acknowledgement message (ACK) was always 5 bytes.

The message format of DATA is described in Table 3.15. The message format of ACK is in Table 3.16. Both DATA and ACK have the starting mark, message ID, CRC (Cyclic Redundancy Check), and ending mark. The body of DATA only contains the ASCII printable characters.

Field	Starting Mark	ID	Body	CRC	Ending Mark
Size(octet)	1	1	0 to 995	2	1

Table 3.15: The fields of DATA from sender

Field	Starting Mark	ID	CRC	Ending Mark
Size(octet)	1	1	2	1

Table 3.16: The fields of ACK from receiver

The interval of sending DATA was fixed to 100 ms. If a sender could not receive an ACK in 100 milliseconds, it continue sending the rest of DATAs. In this case, the delay is counted as 100 milliseconds. In other words, if a DATA has a delay of 100 milliseconds, it may be a lost DATA, even it can receive an ACK latter.

In addition, to assure an enough time for an ACK to be received in fog conditions, there was a two milliseconds turnaround time in the end of every DATA transmission.

[2]One extra test is in the settings of 25 meters, 10 dBm, and 5 visibility distance. It is not a part of the plan, but to confirm the effect from changing the distances and the densities of fog.

In each time of changing size, a DATA was sent 25 times. There was no requirement for resending the DATA. In the end of it, the sender calculated and stored the average values of message delay and loss rate. After finishing a sub-test, the sender sent these average values to the ZigBee station.

3.7.1.5 Evaluation Metrics

If not specified, the term *message* in this section means a DATA or an ACK with full contents. This thesis uses octet as the basic unit to measure the size of messages.

The *"delay"* or *"message delay"* means the response time between sending a DATA and receiving its ACK: the former is the timing when buffering the first character of a DATA to a XBee-PRO Module; the latter is the timing when receiving the first character of a correct ACK. In other word, the delay involves the actions of a sender and a receiver. This thesis uses millisecond as the unit of this duration.

The *"loss"* means that a DATA cannot receive its corresponding ACK in 100 ms.

3.7.2 Results and Analyses

3.7.2.1 Overview

The last chapter has evaluated the factors that cause the delays in an ideal scenario without disturbances. This chapter will examine the delay factors from transmit power, distance and density of fog.

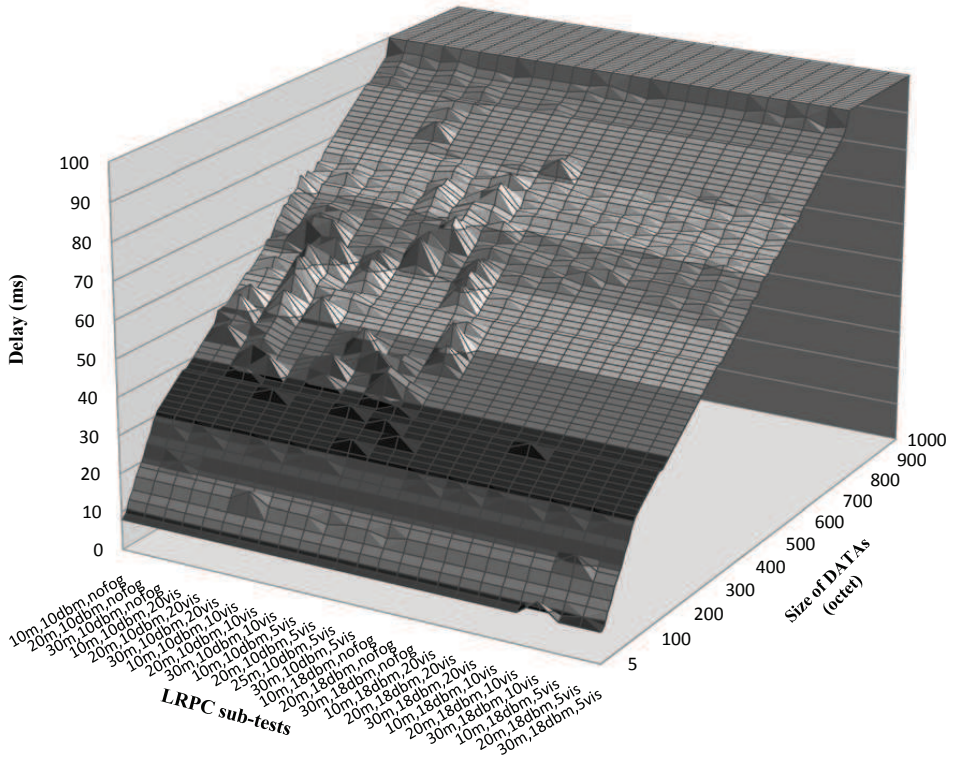


Figure 3.41: Overall delays and loss rates in different transmit powers

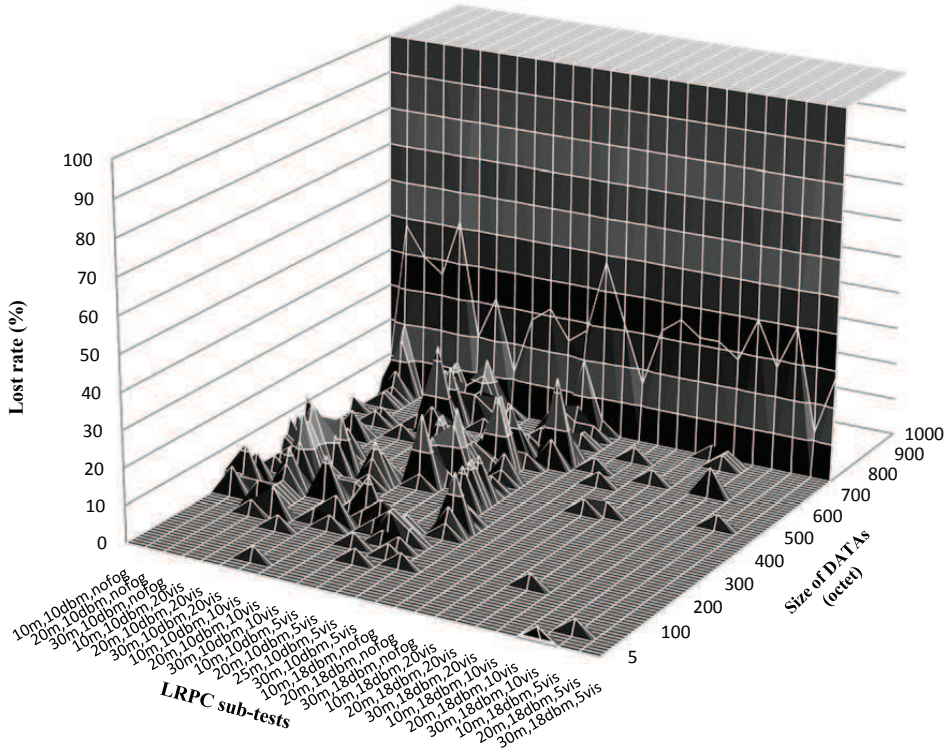


Figure 3.42: Overall delays and loss rates in different transmit powers

We will start the analysis with the overview results from LRPC experiments. The Figure 3.41 and Figure 3.42 demonstrate the results of message delays and loss rates by the vertices of surfaces. The dimension marked as "LRPC sub-tests" are ordered according to transmit powers. The distinction is obvious between transmit powers, but it is not obvious between the transmit distances or densities of fog.

As previously mentioned, the effective size of DATA is less than or equal to 760 octets because the 770-octet DATAs may meet the collision problem. The oversize DATAs are not able to receive their ACKs in 100 ms, and they will be marked as lost ones. The graph Figure 3.43 shows the loss rate of DATAs with sizes between 5 octets and 760 octets. The results from the DATAs with sizes up to 770 octets are given as a comparison. If not specified, the message delay and loss rate in the following sections indicates the ones less than or equal to the effective size.

The Figure 3.44a and Figure 3.44b shows the differences between the best conditions (no fog, the shortest transmit distance, and the maximum transmit power) and the worst conditions (the maximum density of fog, the longest transmit distance, and the minimum transmit power). In general, the lines indicating these two conditions overlap with others, and the differences are only in certain short areas. Because the results in the best conditions fit the theoretical results, the six factors in the last Section 3.6 should not cause these differences.

3.7.2.2 Transmit Power

The transmit power indicates the strength of the signal from a RF transmitter. A smaller transmit power reduces the energy consumption and shortens the transmit distance in most

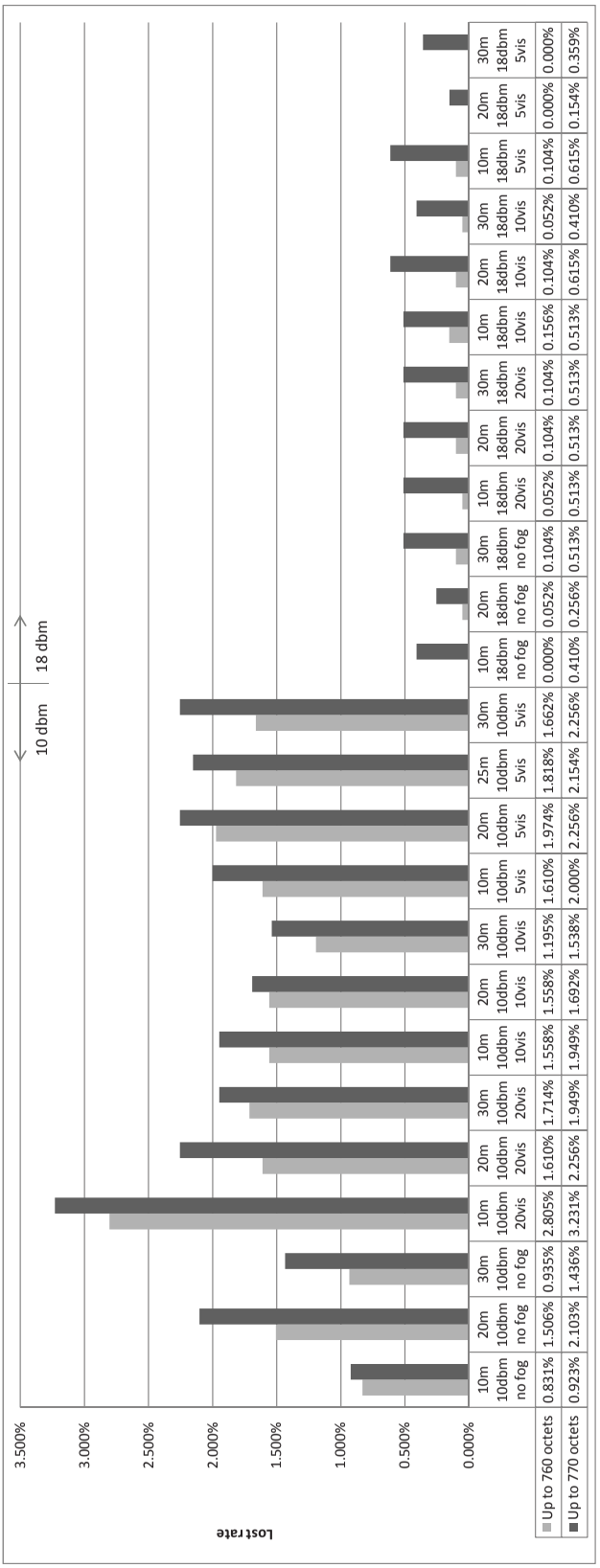
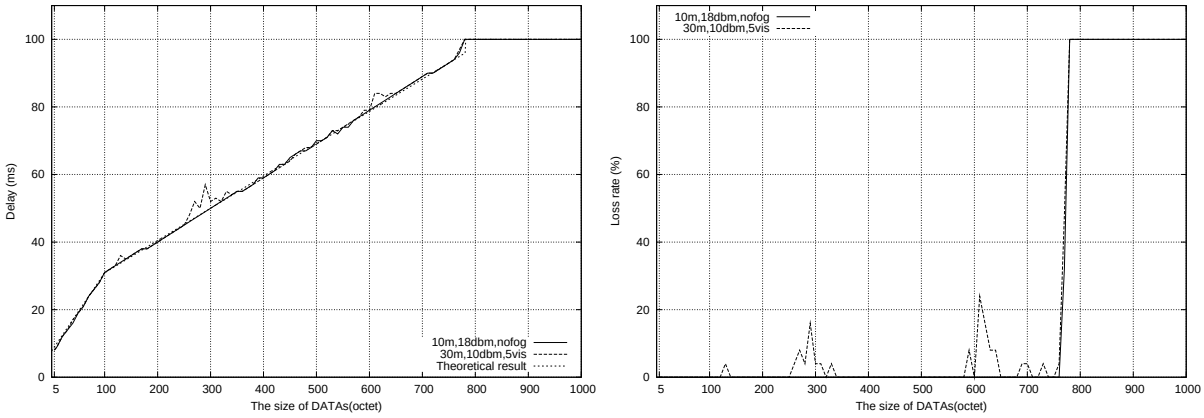


Figure 3.43: The average loss rates when sending DATAs below and equal to the effective size



(a) Delays (b) Loss rates

Figure 3.44: Delays and loss rates in the best and worst conditions

cases. Because 802.15.4 standard operates within the free 2.4 GHz frequency band, a suitable transmit power can reduce the interference from other technologies sharing the same frequency, such as Wi-Fi, Bluetooth, and even microwave ovens in industrial environments.

The 802.15.4 standard specifies a minimum transmit power as 0 dBm, without specified the maximum. When operating in Europe, the maximum transmit power of XBee-PRO modules is at or below 10 dBm. In France, the outdoor operation within the 2.4 GHz band is limited to 10 dBm EIRP. The LRPC experiments tested two of the transmit powers provided by XBee-PRO modules: the minimum of 10 dBm, and the maximum of 18 dBm.

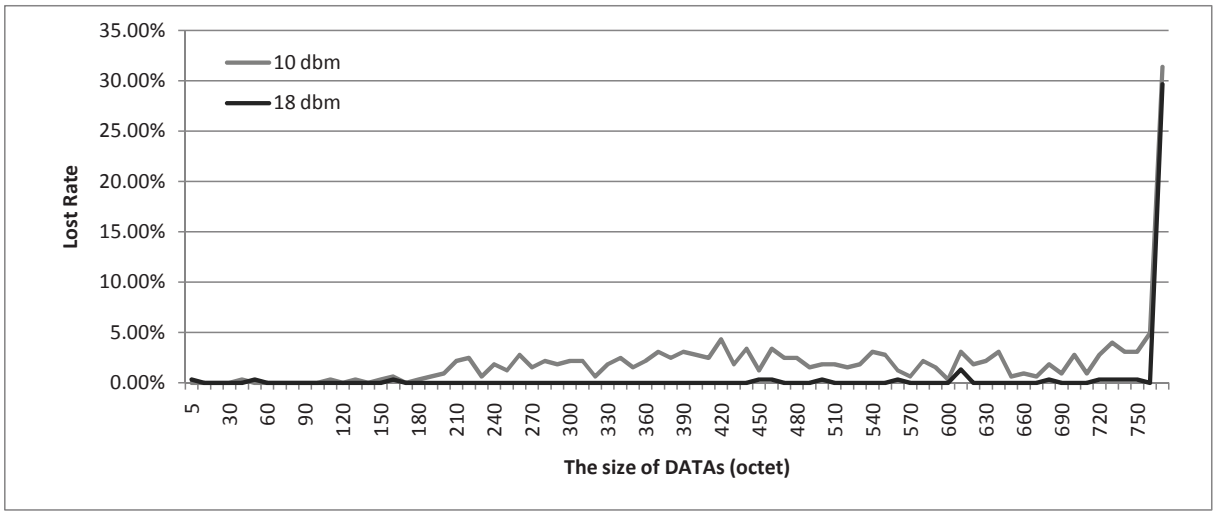


Figure 3.45: The average loss rates under the different transmit powers

The Figure 3.45 shows the average loss rates relate to the size of DATAs. It sums up the sub-test results under a transmit power of either 10 dBm or 18 dBm. If lower than 170 octets, the average loss rates of 10 dBm and 18 dBm are 0.103% and 0.056%. From 180 to 760 octets, the average loss rates of 10 dBm goes up to 2.055% and the increases do not relate to message sizes. In this cases, the average loss rate of 18 dBm is still close to zero (0.073%).

The disturbances from fog and distance are removed in the Figure 3.46 and Figure 3.47.

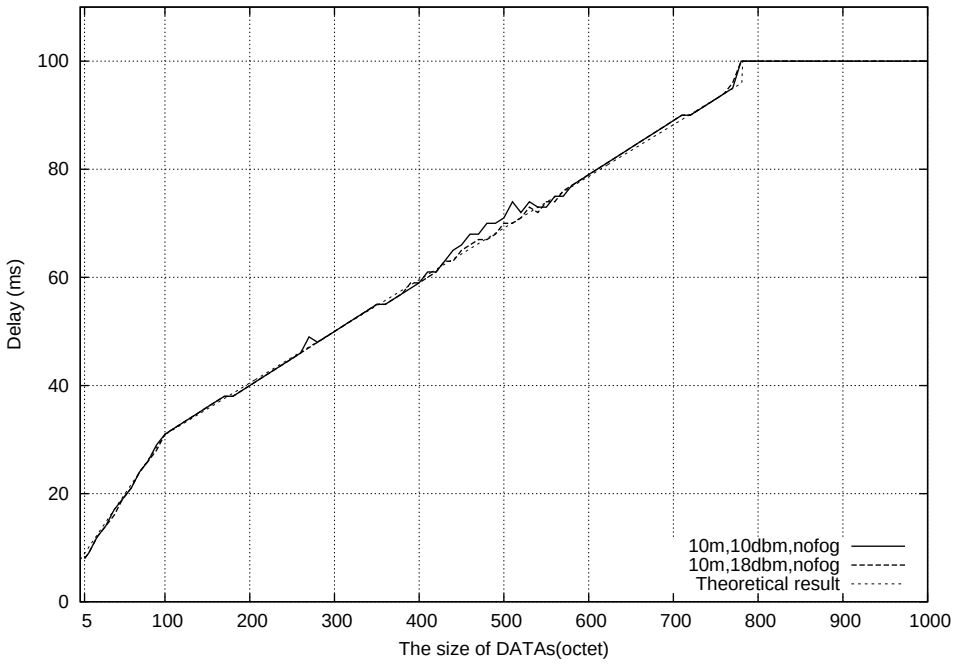


Figure 3.46: The average loss rates under the different transmit powers

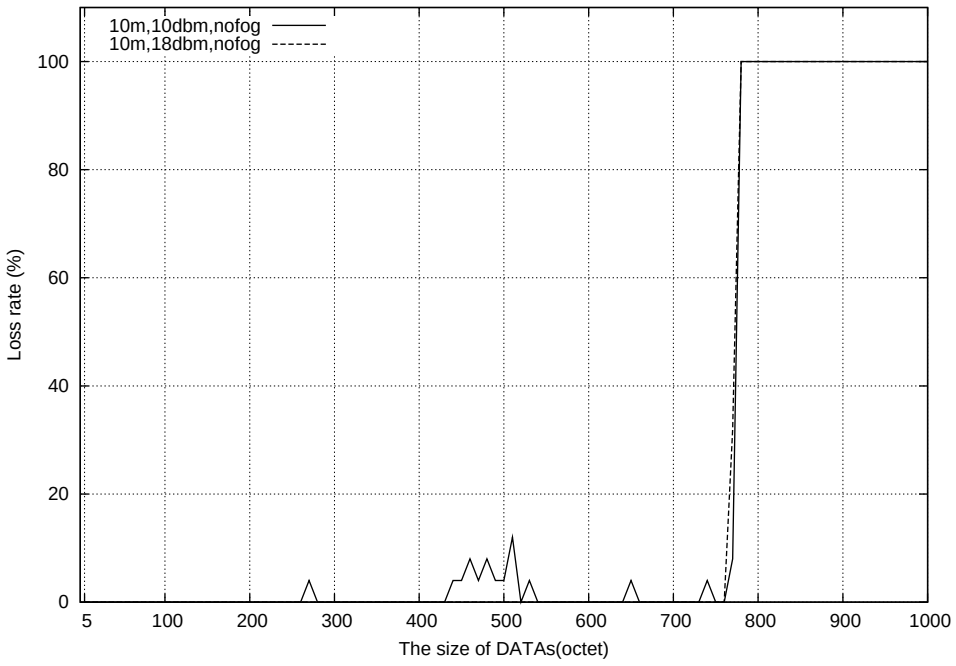
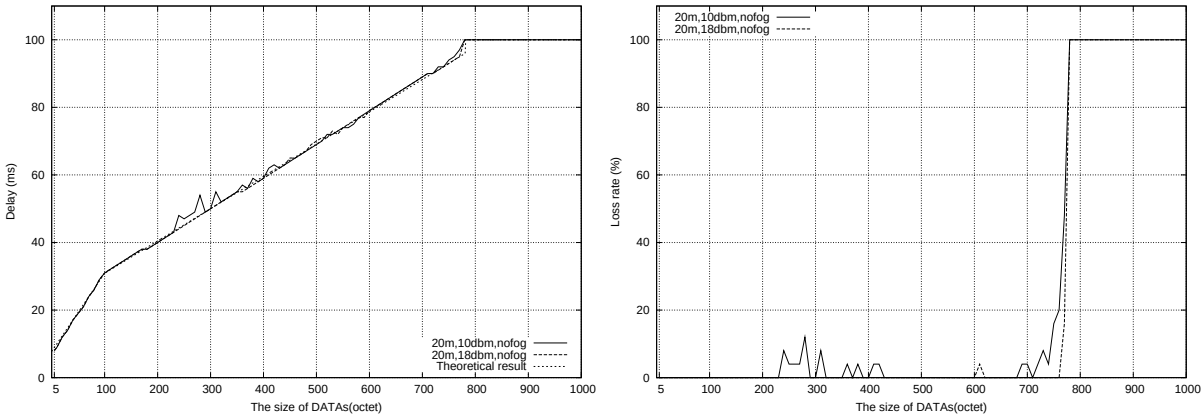


Figure 3.47: The average loss rates under the different transmit powers

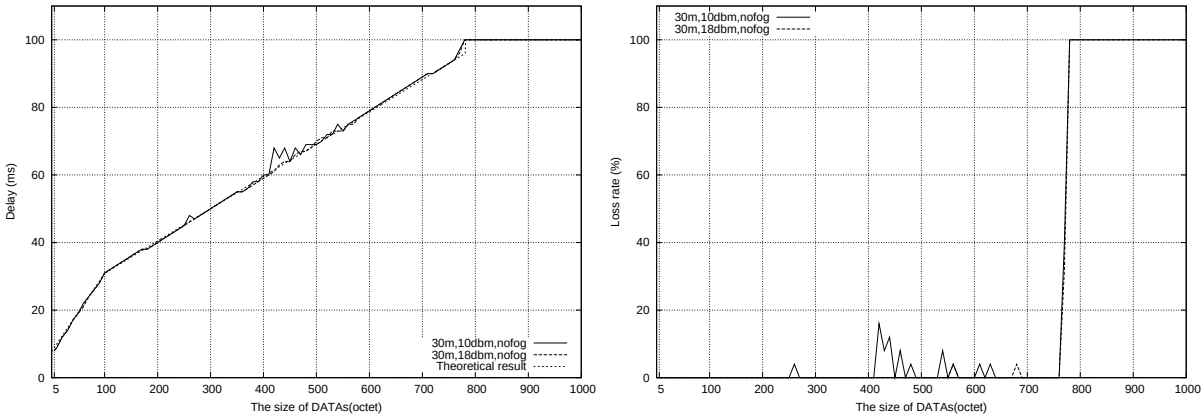
When no disturbance, there is no lost message under transmit power of 18 dBm until the message size reaches 770 octets. Under the transmit powers of 10 dBm and 18 dBm, the average of differences in message delays and loss rates are 0.286 ms and 0.831%. Exceptions are between 440 and 570 octets, with the average of differences 1.286 ms and 3.714%.

Because the exceptions under both transmit powers are in the same data section, and the exceptions does not increase with the size of DATAs, we can only assume the exceptions are caused by the wireless interferences. If this assumption is correct, we should be able to see the exceptions happen in the other data section when only the position of receiver changes. The Figure 3.48 and Figure 3.49 indicate the changes.



(a) Delays (b) Loss rates

Figure 3.48: Effect of output powers (visibility = no fog, distance = 20 meters)



(a) Delays (b) Loss rates

Figure 3.49: Effect of output powers (visibility = no fog, distance = 30 meters)

The averages of difference in the distance of 20 and 30 meters are shown in Table 3.17. The exception loss rates are not increasing with the distance. Note that, when the size of DATAs are close to the effective size (760 octets), even without message loss, the delay will be approximately equal to the maximum 100 ms. The loss in such data section will not has

much impact on the result of delay, and the delay cannot indicate the network status effectively (e.g. the exception data section of 690∼760 in Table 3.17).

Transmit distances (meter)	10	20	30
Message delays (ms)	0.286	0.351	0.299
Loss rates	0.831%	1.455%	0.831%
Main exceptions (octets)	440∼570	240∼310, 690∼760	420∼540
Exception message delays (ms)	1.286	2.5, 0.375	1.308
Exception loss rates	3.714%	5.000%, 7.500%	4.308%

Table 3.17: The average of differences between 18 dBm and 10 dBm without fog

An interesting phenomenon is that when the sender is in 20 meters, the averages of difference loss rate are the highest one. The similar phenomenon can also be found in the graph Figure 3.43 in the overview Section 3.7.2.1. When the transmit power is 10 dBm, except the density of fog is 20 meter visibility, the loss rate in 20 meters is always the highest one. There may be three reasons may cause the phenomenon:

- There is a wireless source near 20 meters distance sharing the same (or nearly the same) frequency.
- The day room in the fog chamber is a semi-circle roof. When the receiver sending ACKs, the first RF reflections may be more concentrate to the central axis where the receiver was located. Therefore, the wireless communication may be more unstable because of the additions and cancellations of RF-wave phases.
- When the receiver is close to the center of fog chamber, it may get the interference from reflections approximately at the same time.

When the sender is in 10 meters and 30 meter, both of the averages of differences on loss rates are 0.831%. We assume that the difference of 0.831% is the pure effect from transmit power because all other factors have been excluded. Therefore, the raw T_{dBm} and L_{dBm} are given to relate the reduction of transmit power (dBm) and the addition of delay (ms) and loss rate (%) as follows:

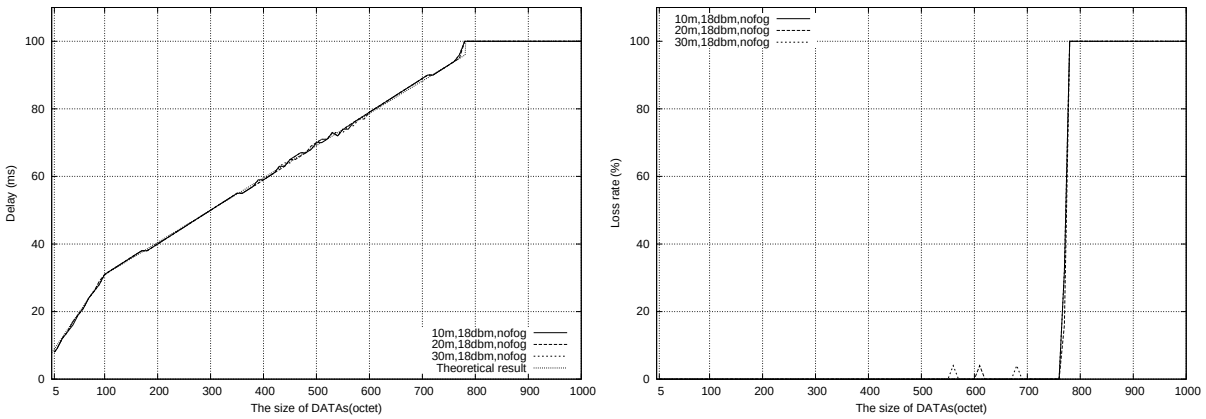
$$T_{dBm} = \frac{0.286 + 0.299}{2 \times (18 - 10)} = 0.0366\ (ms/dbm) \tag{3.17}$$

$$L_{dBm} = \frac{0.831\%}{18 - 10} = 0.1039\ (\%/dbm) \tag{3.18}$$

In summary, the effect between different transmit powers is more obvious than fog and distance. When the size of DATAs is below 180 octets, the average difference of transmit powers 10 dBm and 18 dBm are both close to zero (0.103% and 0.056%). Between 180 and 760 octets, the transmit power of 18 dBm has better performance under disturbances. The average differences of two transmit powers are small (average 1.981%), although there are some exceptions from interference by reflections (maximum 7.500%). The raw Eq. 3.17 and Eq. 3.18 shows the factor of transmit power.

3.7.2.3 Transmit Distance

Some results to evaluate the effects of transmit distance have been shown in the last Section 3.7.2.2 (e.g. the exceptions in 20 meters). This section focus on giving the equation to relate the transmit distance and the loss rate. As we have known that the transmit power of 18 dBm excludes the most of disturbances, the sub-tests with 18 dBm and no fog will be used to evaluate the effect of transmit distance. The Figure 3.50 shows the result of three related sub-tests.



(a) Delays (b) Loss rates

Figure 3.50: Effect of distance on message delay (visibility = no fog, output power = 18 dBm)

A longer transmit distance will normally affect the wireless performance because the loss of transmit power over distance, but since the indoor transmit distance of XBee-PRO is about 90 meters [102], and the length of fog chamber is just 31 meters, we assume that the changes of distance will not greatly affect delay and loss rate. The assumption is confirmed by the sub-test results. The loss rates of 10, 20 and 30 meters increases linearly from 0%, 0.052% to 0.104% as shown in graph Figure 3.43. Moreover, the similar pattern exists in the loss rates of 10 and 30 meter under 10 dBm without fog. The difference of loss rates after subtracting the L_{dBm} is 0.104% ($0.935\% - 0.831\%$) as the same as under 18 dBm.

For the message delays, there are several slight losses when sent messages are close to 760 octets, but as previously mentioned, the loss in this data section will not has much impact on the message delay. The average of differences between 10 meters and 20 meters is 0 ms; between 10 meters and 30 meters, the average of differences is 0.013 ms.

Therefore, the raw T_{dis} and L_{dis} are given to relate the increase of distance (meter), and the addition in delay (ms) and loss rate (%):

$$T_{dis} = \frac{0.013}{30 - 10} = 0.00065\ (ms/meter) \tag{3.19}$$

$$L_{dis} = \frac{0.052\%}{10} = 0.0052\ (\%/meter) \tag{3.20}$$

We assume that the T_{dis} and L_{dis} purely relate to the transmit distance. It is just a raw assumption that can only be used in exactly the no fog indoor environment like the fog chamber at LRPC.

3.7.2.4 Fog

Knowing the fog effects in a wireless transmission are important for the design of an inter-vehicle communication (IVC) protocol like CIVIC. The experiments in this sections tries to discover the message delay and loss in 2.4 GHz band caused by the different densities of fog; but before going into the discussion, we will firstly introduce the general fog effects, and the experiment environments at the fog chamber of LRPC.

Two phenomenons would normally occur when an RF signal travels through a foggy medium: Firstly, the droplets of fog can attenuate an RF signal. The 2.4 GHz signals in thick fog may be attenuated by up to 0.02 dB/km by fog [110], but this reference does not define which density can be considered as "thick". Secondly, the transmit conditions become more complex because of the different densities of fog. A natural foggy medium has a different density; it causes the problems of RF refraction and multiple paths. The combined signal by multiple paths will often result in attenuation, amplification, or signal corruption.

The fog chamber at LRPC provides a controllable fog environment. Our experiments have been planned to be done in three densities, which are measured by the visibility distances of 20, 10 and 5 meters. The actual densities of fog during sub-tests have little difference than the plan as shown in Table 3.18.

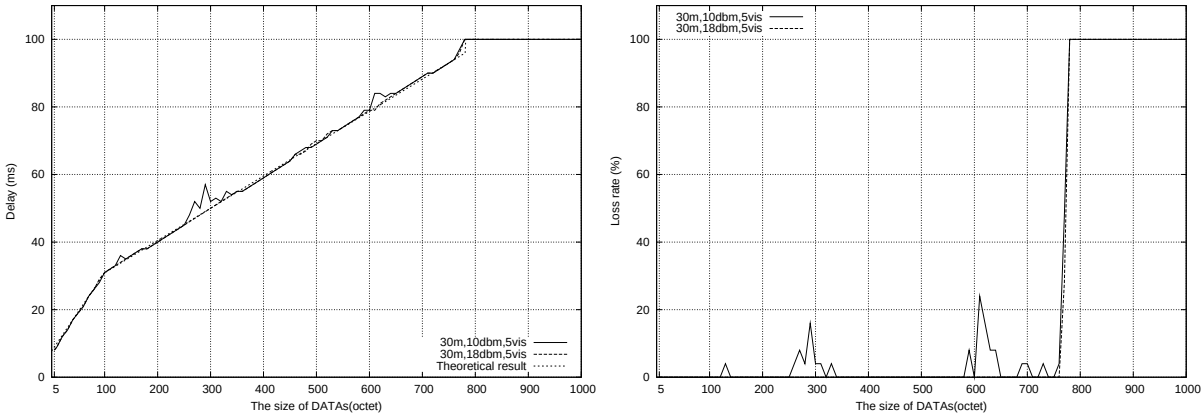
Table 3.18: The average densities of fog in records

	10m,10dbm	**20m,10dbm**	**30m,10dbm**
vis=20, VM	23.00	23.13	14.95
vis=20, POM	19.63	19.28	***14.53***
vis=10, VM	11.46	11.18	10.32
vis=10, POM	10.00	10.19	***7.65***
vis=5, VM	6.90	7.37	8.43
vis=5, POM	6.00	6.28	6.25
	10m,18dbm	**20m,18dbm**	**30m,18dbm**
vis=20, VM	22.77	20.67	22.44
vis=20, POM	19.56	19.13	19.19
vis=10, VM	11.39	11.19	11.82
vis=10, POM	10.94	9.88	10.09
vis=5, VM	6.80	7.78	7.57
vis=5, POM	6.00	6.00	6.00

The rows marked by "POM" are the densities of the whole chamber, and the rows marked by “VM” indicate the density of the half of chamber closing to the observation station. The differences are normally less than the one meter of visibility distance, except two sub-tests (in 30 meters, 10 dBm), which are marked as the bold italic fonts. The actual densities of the whole chamber (i.e. POM) are used in our analysis; therefore, if not specified, the visibility distance (or, *vis*) in the thesis means the density of the whole fog chamber.

Note that, the density could be slightly changed during a sub-test. Sometimes when the density of POM is stable, the density of VM can be changing (especially in the maximum density close to 6 meters), thus the droplets of fog could be flowing during some sub-tests.

We start the following discussion from the longest transmit distance and the maximum density of fog offered by the fog chamber. The Figure 3.51 shows the results relate to the



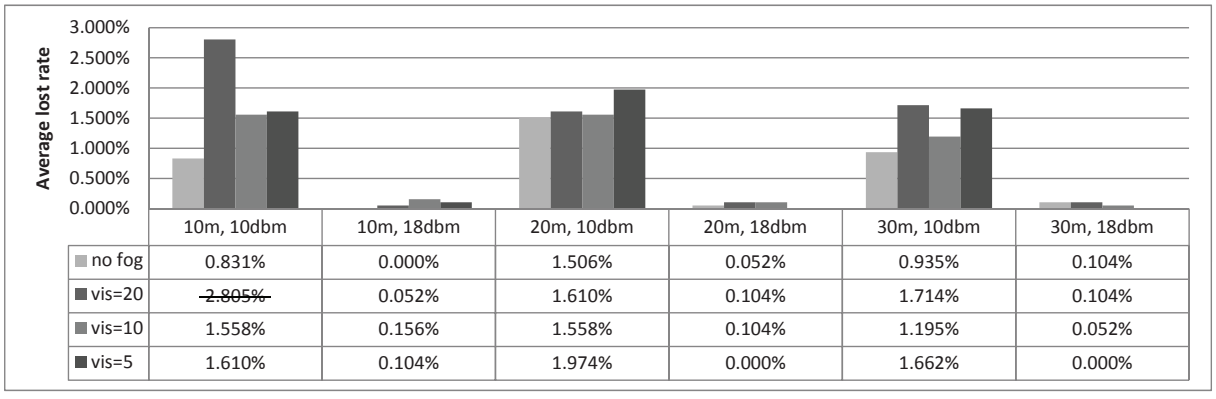
(a) Delays (b) Loss rates

Figure 3.51: Effect of output powers (visibility = 5 meters, distance = 30 meters)

size of DATA. The loss rate of 18 dBm in the conditions is zero. The average differences of message delay and loss rate between 10 dBm and 18 dBm are 0.442 ms and 1.662%.

Note that, when a vehicle driver cannot see through the fog, he or she is exposed to the unseen hazard. This result shows that in the visibility of 6 meters, the IVC designed under the 2.4 GHz band can still work, and it can help to avoid hazard at least in 30 meters ahead. Both 10 dBm and 18 dBm have good performances under the maximum disturbances, but of course, the 18 dBm is better.

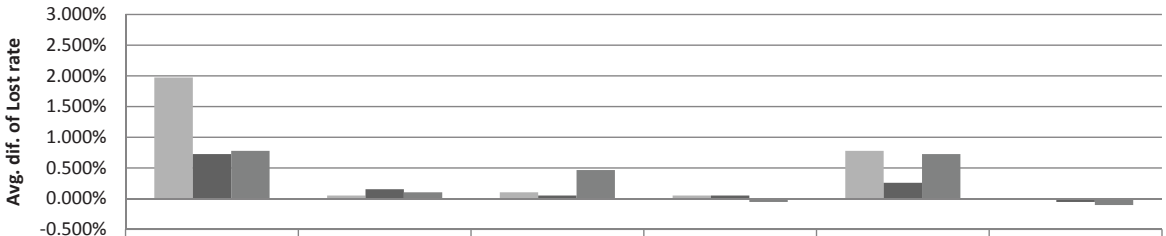
The graph Figure 3.52 shows the fog effects under two transmit powers. The sub-test in 10 m and 10 dBm is an exception, and it will not be used in the following analysis (it is marked by a strikethrough in the following tables). The exception may be caused by the problem from the battery power. A sub-test of 30 meters and 10 dBm has failed for the same reason, but the sub-test has been redone. If a sub-test fails because of the problem of power supply, it is difficult to be found unless it is obvious or it is compared with other results after experiments.



	10m, 10dbm	10m, 18dbm	20m, 10dbm	20m, 18dbm	30m, 10dbm	30m, 18dbm
no fog	0.831%	0.000%	1.506%	0.052%	0.935%	0.104%
vis=20	~~2.805%~~	0.052%	1.610%	0.104%	1.714%	0.104%
vis=10	1.558%	0.156%	1.558%	0.104%	1.195%	0.052%
vis=5	1.610%	0.104%	1.974%	0.000%	1.662%	0.000%

Figure 3.52: The average loss rates with fog effects

The graph Figure 3.52 shows the fog effects by comparing the sub-test results with fog and the sub-test result without fog, thus the only variable is the density of fog including the visibility distances of 20, 10, and 5 meters. The values in the graph table are average differences of loss rates.



	10m, 10dbm	10m, 18dbm	20m, 10dbm	20m, 18dbm	30m, 10dbm	30m, 18dbm
VIS=20	~~-1.974%~~	0.052%	0.104%	0.052%	0.779%	0.000%
VIS=10	0.727%	0.156%	0.052%	0.052%	0.260%	-0.052%
VIS=5	0.779%	0.104%	0.468%	-0.052%	0.727%	-0.104%

Figure 3.53: The average differences of loss rates with fog effects

Because the disturbances have a very minor effect under the transmit power of 18 dBm, and the result under 18 dBm fits the theoretical result, the following analysis will focus on the transmit power of 10 dBm. The Table 3.19 and Table 3.20 combine the data from the Table 3.18 and Figure 3.52, and the data is divided into two groups of transmit powers as 10 dBm and 18 dBm.

Table 3.19: The average differences of loss rate relate to POMs less than 10 dBm

	10m,10dbm	**20m,10dbm**	**30m,10dbm**
POM	~~19.63~~	19.28	14.53
Avg. Dif. of Loss Rate	~~1.974%~~	0.104%	0.779%
POM	10.00	10.19	7.65
Avg. Dif. of Loss Rate	0.727%	0.052%	0.260%
POM	6.00	6.28	6.25
Avg. Dif. of Loss Rate	0.779%	0.468%	0.727%

Table 3.20: The average differences of loss rate relate to POMs less than 18 dBm

	10m,18dbm	**20m,18dbm**	**30m,18dbm**
POM	19.56	19.13	19.19
Avg. Dif. of Loss Rate	0.052%	0.052%	0.000%
POM	10.94	9.88	10.09
Avg. Dif. of Loss Rate	0.156%	0.052%	-0.052%
POM	6.00	6.00	6.00
Avg. Dif. of Loss Rate	0.104%	-0.052%	-0.104%

By the data in first Table 3.19 under the transmit power of 10 dBm, we get a linear Eq. 3.21 with the coefficient of determination $R^2 = 0.135383$ to relate the densities of fog with L_{fog} as follows:

$$L_{fog\_10dbm}(x) = -0.000242x + 0.007293 \tag{3.21}$$

where

x : the densities of fog in the fog chamber with a POM unit

The linear Eq. 3.21 shows the tendency of the increasing of L_{fog} when the density of fog is increasing. A more accurate equation could be a power equation with $R^2 = 0.184609$ as the following:

$$L_{fog\_10dbm}(x) = 0.035457x^{-1.047134} \tag{3.22}$$

When the POM is 30.14 meters, the fog effects is zero by the linear Eq. 3.21. The power Eq. 3.22 may be a better reflection when the density of fog is increasing after the POM equals to 6 meters. The Figure 3.54 indicates the trend of both equations.

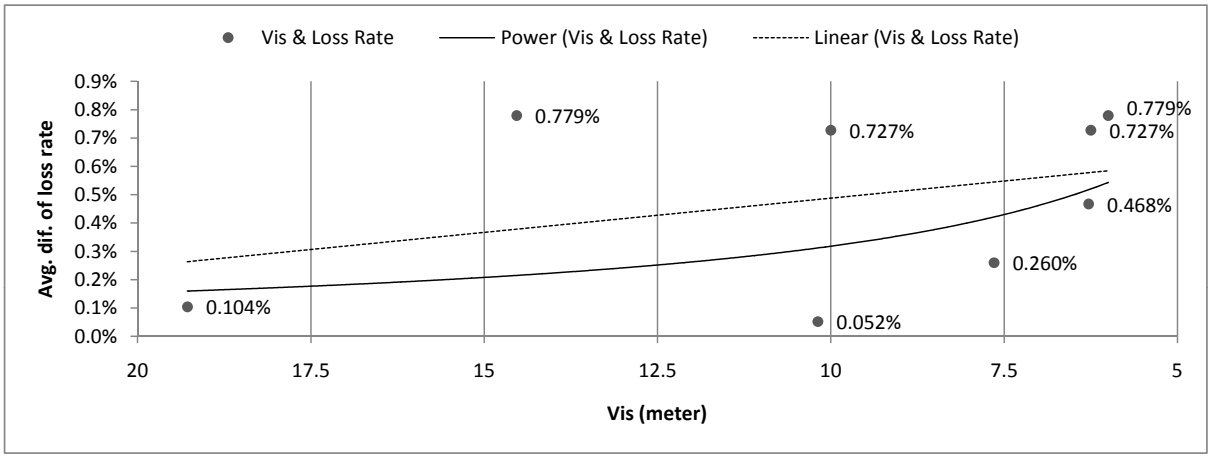


Figure 3.54: The fog effects on loss rates under 10 dBm by the Eqs. 3.21 and 3.22

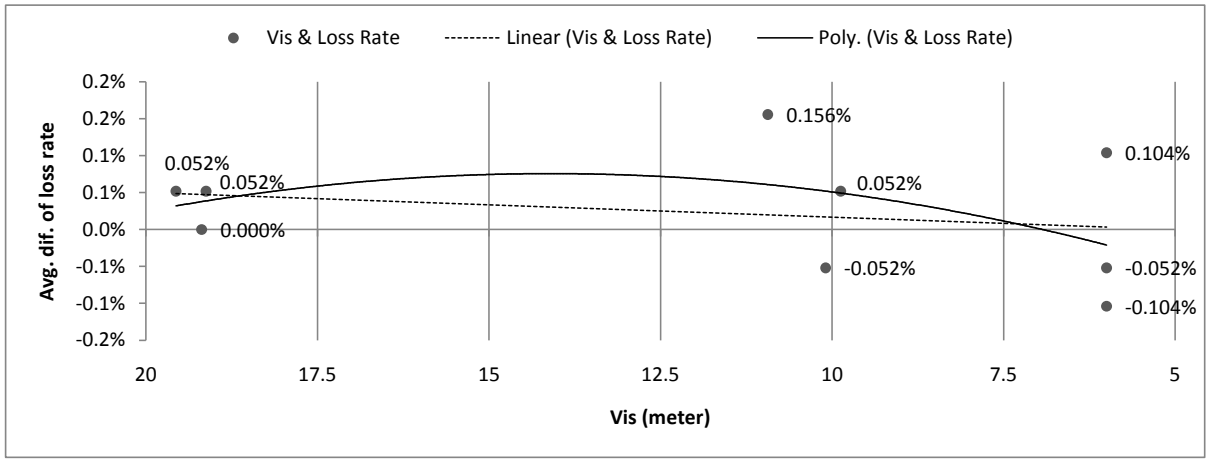


Figure 3.55: The fog effects on loss rates under 18 dBm by the Eqs. 3.23 and 3.24

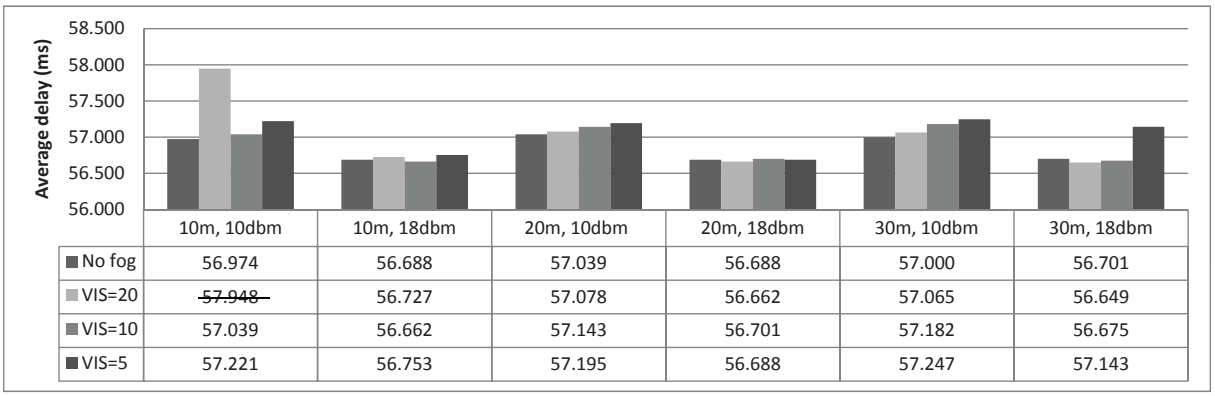
By the data in second Table 3.20 under the transmit power of 18 dBm, we get a linear Eq. 3.21 with the coefficient of determination $R^2 = 0.056764$. The linear equation shows the tendency of the decreasing of L_{fog} when the density of fog is increasing. Although the tendency of equations shows the decreasing, the maximum average difference of loss rate is just 0.156%; therefore, the decreasing may not be caused by the fogs effects, but just the small exceptions from wireless communications. A more accurate equation could be a polynomial

equation with $R^2 = 0.170758$ as the following Eq. 3.24. The Figure 3.55 indicates the trend of the Eq. 3.23 and Eq. 3.24.

$$L_{fog\_18dbm}(x) = 0.000033x - 0.000166 \tag{3.23}$$

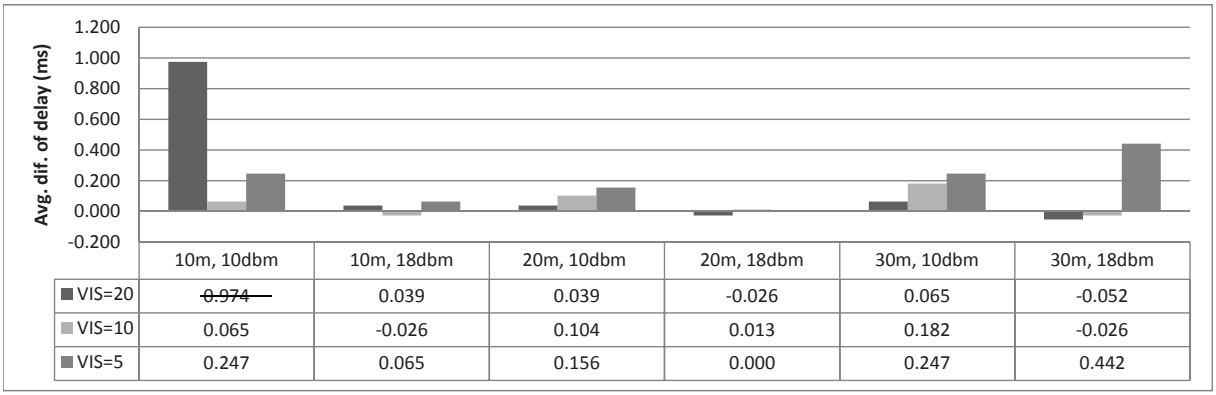
$$L_{fog\_18dbm}(x) = -0.000015x^2 + 0.000415x - 0.002170 \tag{3.24}$$

The following calculations are for the fog effects on message delays under two different transmit powers. The process is the same as the calculations of loss rate. The graph Figure 3.56 and Figure 3.57 the fog effects on delays under two transmit powers. Then we get the following Table 3.21 as the one in the loss rate section.



	10m, 10dbm	10m, 18dbm	20m, 10dbm	20m, 18dbm	30m, 10dbm	30m, 18dbm
No fog	56.974	56.688	57.039	56.688	57.000	56.701
VIS=20	~~57.948~~	56.727	57.078	56.662	57.065	56.649
VIS=10	57.039	56.662	57.143	56.701	57.182	56.675
VIS=5	57.221	56.753	57.195	56.688	57.247	57.143

Figure 3.56: The average delays with fog effects



	10m, 10dbm	10m, 18dbm	20m, 10dbm	20m, 18dbm	30m, 10dbm	30m, 18dbm
VIS=20	~~0.974~~	0.039	0.039	-0.026	0.065	-0.052
VIS=10	0.065	-0.026	0.104	0.013	0.182	-0.026
VIS=5	0.247	0.065	0.156	0.000	0.247	0.442

Figure 3.57: The average differences of delays with fog effects

The Eq. 3.25 ($R^2 = 0.693577$) and Eq. 3.26 ($R^2 = 0.888362$) is given by the graph Figure 3.57 under the transmit power of 10 dBm. The Figure 3.58 shows the trend of these two equations.

$$T_{fog10tp}(x) = -0.014532x + 0.283627 \tag{3.25}$$

$$T_{fog10tp}(x) = 3.270625x^{-1.510576} \tag{3.26}$$

Table 3.21: The average differences of delay relate to POMs

	10m,10dbm	**20m,10dbm**	**30m,10dbm**
POM	~~19.63~~	19.28	14.53
Avg. Dif. of Delay	~~0.974~~	0.039	0.065
POM	10.00	10.19	7.65
Avg. Dif. of Delay	0.065	0.104	0.182
POM	6.00	6.28	6.25
Avg. Dif. of Delay	0.247	0.156	0.247
	10m,18dbm	**20m,18dbm**	**30m,18dbm**
POM	19.56	19.13	19.19
Avg. Dif. of Delay	0.039	-0.026	-0.052
POM	10.94	9.88	10.09
Avg. Dif. of Delay	-0.026	0.013	-0.026
POM	6.00	6.00	6.00
Avg. Dif. of Delay	0.065	0.000	0.442

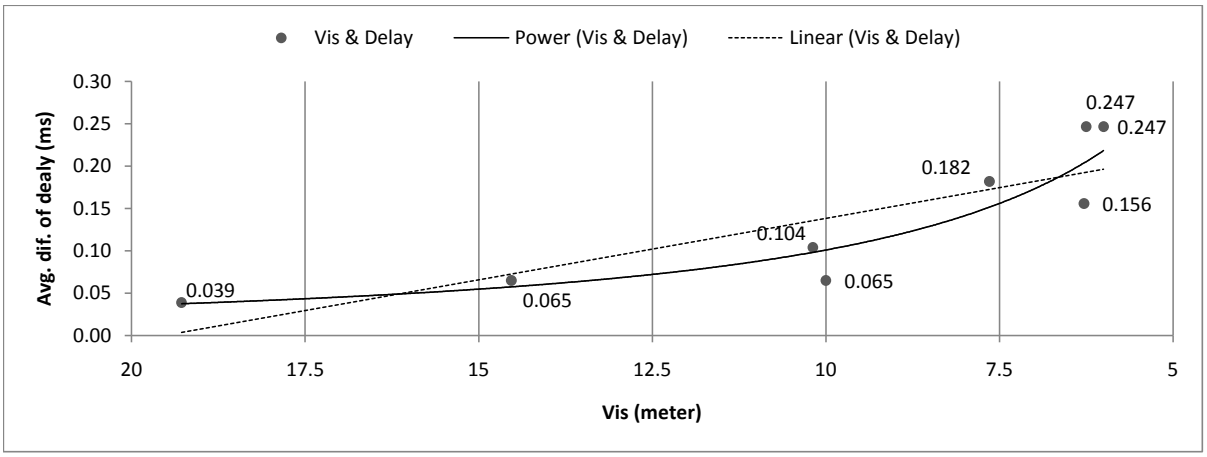


Figure 3.58: The fog effects on delays under 10 dBm by the Eqs. 3.25 and 3.26

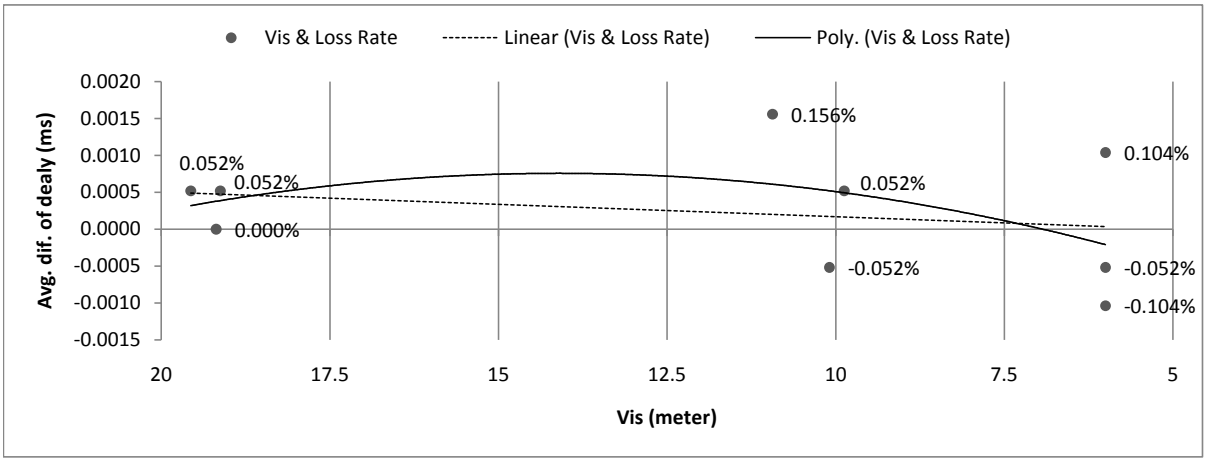


Figure 3.59: The fog effects on delays under 18 dBm by the Eqs. 3.27 and 3.28

The Eq. 3.27 ($R^2 = 0.1992$) and Eq. 3.28 ($R^2 = 0.36549$) are given under the transmit power of 18 dBm with the Figure 3.59 to show the trend.

$$T_{fog18tp}(x) = -0.0115x + 0.1846 \tag{3.27}$$

$$T_{fog18tp}(x) = 0.003271x^2 - 0.096410x + 0.630394 \tag{3.28}$$

3.7.3 Conclusion and Limitations

The experiments in fog environment are important for improving our communication system. The network in 2.4 GHz band performs well in the experiments. Even with the worst disturbance on fog (POM=6m) and distance (30m), the communication under the transmit power of 18 dBm still gets a zero loss rate, and the loss rate under 10 dBm is 1.662% (Section 3.7.2.1). If the size of DATAs is lower than 170 octets, the loss rate under 10 dBm is close to zero in all sub-tests (Section 3.7.2.2). For all sub-tests under 18 dBm, the loss rates are always close to zero. Note that, the delay and loss rate in this report indicate the quality of a whole message communication including sending of DATA and ACK (Section 3.7.1.5).

This section and the last section evaluate all the possible factors that cause the message delay and loss rate. There are two groups of these factors:

The first group of factors relates to the network technology including the UART and RF transmissions, CSMA/CA, IFS, XBee-PRO operations and software operations in the Section 3.6. The current bottleneck is in UART transmission. However, the 115200 baud is already the highest standard baud provided by the XBee-PRO module. Over than the 115200 baud, more error results will be caused. The software delay in this report should be slightly lower than the actual software delay, because we only consider the time consumption in the major processing: the processing for DATAs on receiver. The maximum delay in CSMA/CA is 8.832 ms. If the XBee-PRO modules in the experiments are more than two, the delay in CSMA/CA should be much higher. Moreover, there is about three milliseconds delay from the hardware operations of a XBee-PRO module (Section 3.6.7), but the XBee-PRO module only provides very limited options to control the hardware operations. the delay from XBee-PRO module is the major delay factor the short message like ACK. The percentage of these delays in a transmission is shown as Figure 3.36. The factors in this group cause the delays in an ideal scenario without disturbances.

The second group of factors is from the disturbances including transmit power, distance and fog in Section 3.7.2. Although the wireless features under two transmit powers of 10 dBm and 18 dBm have many different, the delay and loss rate are normally reduced when lowering the transmit power, adding the transmit distance, and increasing the density of fog. However, there are two exceptions:

- When the transmit power is 10 dBm, the wireless conditions in the transmit distance of 20 meters is the worst one (Section 3.7.2.2).
- When the transmit power is 18 dBm, the loss rate and delay decrease with the increasing of density of fog (Section 3.7.2.4), but it is not sure to be caused by the wireless interferences or the real fog effects.

The equations to perform the theoretical calculations of all the delay factors are given in Section 3.6 and Section 3.7.2. We assume that in an ideal scenario, when sending message below or equal to 760 octets, the loss rate should not be zero, thus the equations to calculate the loss rate are only given for the second group of factors from the disturbances in chapter Section 3.7.2.

There are two possible fails in the theoretical calculations of chapter Section 3.6: Firstly, there is a factor in the theoretical calculations that has not been fully considered. From all the graphs indicating delay, we found that nearly each 170-180 octets, there a short pause on the delay results. It is assumed to be caused by the CTS hardware flow control. The size of DI buffer is 202 octets. When the DI buffer is 17 octets away from full, the XBee-PRO signals the MCU to stop sending data by de-asserting the CTS pin. When the DI buffer has 34 octets available in memory, the CTS pin is re-asserted. We did enable the CTS on MCU, but in the oscilloscope experiments, there is no change on the CTS pin. Therefore, we are not sure the CTS hardware flow control is actually enabled. However, it may be just because the period during the change of CTS is too short. Secondly, when the size of DATAs is about to 770 octets, the ACKs begin to compete with the next DATAs in the same channel, so the collisions happen. In this situation, the T_{csma} in Section 3.6.4 will not be correct because it assume the channel accesses are always successful at the first time.

In the chapter Section 3.7.2, the available samples are not enough for completing the accurate calculations. Except the experiment design, the major reasons are from the problem of interference in Section 3.7.2.2 and battery power in Section 3.7.2.4. For the interference problem, it is normal for indoor experiments. For the battery power problem, if there will be another experiment at fog chamber, at least the sender in the observation station should be using the fixed power supply. Besides, the fog densities in two of the sub-tests are lower than the requirements (Section 3.7.2.4).

3.8 Simulations on Shawn

3.8.1 Introduction of Shawn

Shawn is a new network simulator for abstract algorithms and high-level protocols in combination with the speed to handle large networks [111] [112].

Instead of requiring to implement full low-level protocol effects (e.g. data and message encoding, the physical effects, processor limitations in *NS-2* [113]), *Shawn* is initially implemented with the abstract and exchangeable models. It is possible to begin with a simple algorithmic sketch and extend the sketch into a completely distributed protocol. This design allows researchers to focus on the actual research problems but not the simulation itself. Moreover, the sacrifice of some low-level details can increase the speed of running simulations of large wireless (sensor) networks. The Figure 3.60 is from [111] and it classifies the application area of existing simulators along two axes, showing abstraction level and number of simulated network nodes.

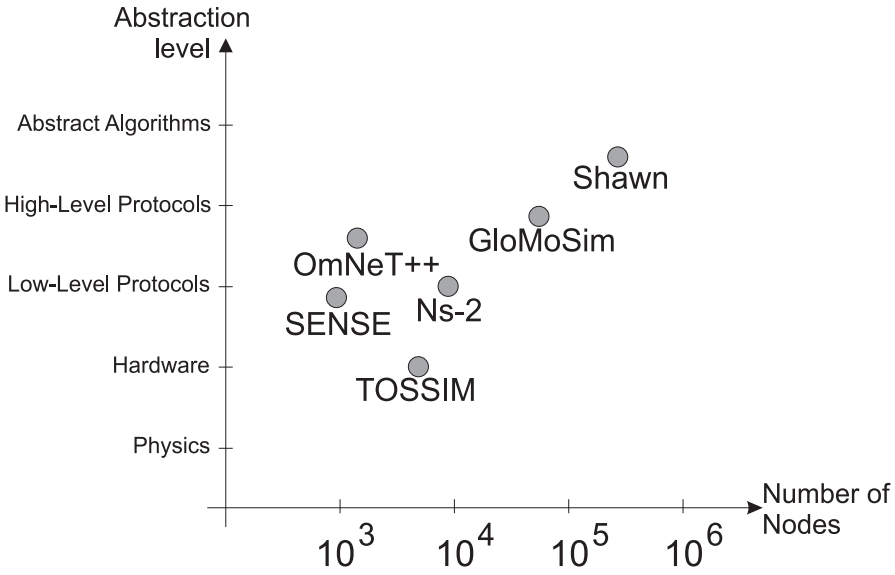


Figure 3.60: Intended application area of simulators

Shawn is licensed under the *BSD* license allowing for commercial applications. It is available for download at http://www.swarmnet.de/shawn.

3.8.2 Simulation Results

3.8.2.1 Beaconless 802.15.4 Network

The first experiment was to test the basic networking performance of IEEE 802.15.4 and see its possibility to be used in VANET or WSN applications. Moreover, because *Shawn* is a relatively new simulator, it is important to firstly evaluate the simulator itself.

The experiment was done in a network with all nodes connecting with each other (width=10 height=10, radio range=100). The transmission model used the beaconless CSMA/CA in IEEE

802.15.4 as in our real world experiments in Section 4.2.

The experiment started with two nodes and finished when the connection loss happening (losing the *HELLO_RPY* message). A node sends a *HELLO_REQ* message in the first millisecond of an experiment "round", and the other nodes reply the *HELLO_REQ* message with a *HELLO_RPY* message (message sizes in Table 3.1 and the details in Section 3.2.2). The results were evaluated by the loss rate (*Message Loss Rate* and *Node Loss Rate*) and delay (*Average HELLO_RPY Delay* and *Last HELLO_RPY Delay*). If a *HELLO_RPY* could not be received in an experiment "round", it was assumed to be lost and the next round starts. The simulations were done in a number of times, and the average values were used.

Number of Nodes	2	3	**4**	5	**6**	7
Message Loss Rate	0%	0%	**8%**	15%	25%	29%
Number of Loss Nodes	0	0	0	0	**2**	3
Node Loss Rate	0%	0%	0%	0%	40%	50%
Average HELLO_RPY Delay (sec.)	0.007	0.018	0.022	0.031	0.025	0.029
Last HELLO_RPY Delay (sec.)	0.007	0.023	0.030	0.046	0.035	0.042

Table 3.22: Beaconless IEEE 802.15.4 results on Shawn

The results in Table 3.22 show that the beaconless IEEE 802.15.4 is easy to be affected by the packet interference, and we got the similar results in our real-world experiments. After the number of nodes was increased to four, some Hello messages had started to be lost. When the number of nodes was increased to six, two nodes were completely lost in the network, and the network starts to become unstable. Therefore, there is no need to be continuing adding numbers of nodes after seven nodes have been added.

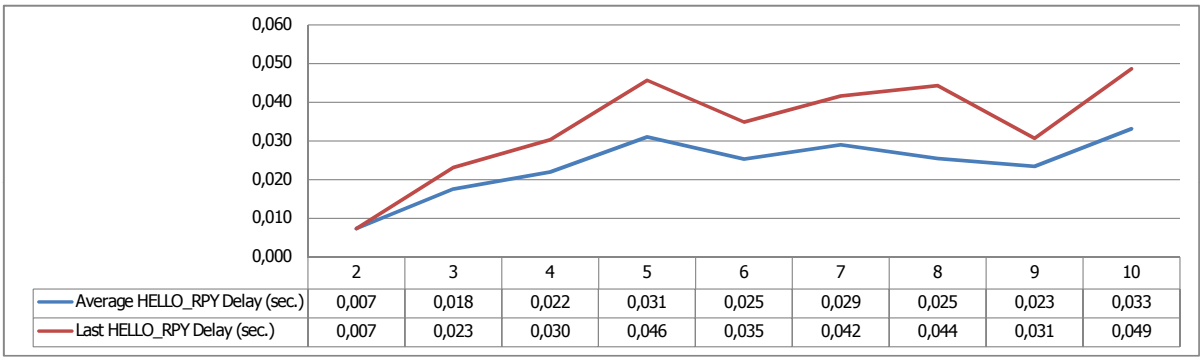


Figure 3.61: Delay for only the connected nodes

Figure 3.61 shows only results about the message delay with up to ten nodes, and the results do not include the ones from completely lost nodes. Note that, the results relating to delay from a network simulator are normally not accurate enough. The basic "time unit" in *Shawn* is called *round*, it was implemented as the unit of *second* for our simulations. A *round* can be divided into a double variable (called "*time*" in *Shawn*), which was implemented as the unit of *millisecond*. There is no real *second* and *millisecond* units in *Shawn*. It is the same case as in *NS-2* because they are not designed to be a network emulator. Although the delay results are not

accurate, the increasing trends of delay results are basically matching the ones from real-world experiments.

3.8.2.2 CIVIC Protocol Network

The second experiment was to implement and test CIVIC protocol on *Shawn*. Based on the result from the last simulation, we have implemented CIVIC protocol with the contention-based forwarding as mentioned in Section 2.3.2 because the Hello messages are easy to cause too much routing overhead in high density networks. Note that, it does not mean that the mechanism of *one-hop link stability* in Section 3.2.1.3 is not necessary, it only means that the original beaconless IEEE 802.15.4 need to be modified before it will be used for high density networks. For the real-world experiment in the next chapter (with only nine *LiveNodes*), we still implemented and tested CIVIC protocol with the link-stability mechanism.

Because there is no Hello message to get neighbor information, three new rules are set for the contention-based greedy forwarding in CIVIC protocol:

- Only the nodes in progress direction of the last sender can be used as the next-hop nodes to forward messages. The routing message contains the location inforamtion of the last sender and the destination. Plus, a node can know its own position. Three locations are enough for a node to decide whether it is in the progress direction toward the destination.
- A additional delay is set before forwarding a routing message to avoid network interference. The value of the delay is based on the distance between the destination node and a node that has passed the first rule and it is ready to forward a routing message. The shorter distance gets shorter delay, thus the found routing path should be the shortest one.
- The same routing request messages can be forwarded only once by a node. In other word, only one routing message passing the second rule can be forwarded by the next forwarder node, and the redundant messages are dropped.

An example of requiring a routing path in a static network is shown in Figure 3.62. The red node is the destination node, the gray edges are the nodes receiving the routing requiring message (a distance-based routing tree), and the blue nodes are those in the feedback routing path (exactly the *ROUTE_RPY_PATH* message). For the simulation only, the limitation on the hop number is removed.

Except reducing the routing overhead as previously mentioned, there are two more advantages of the contention-based model which can be proved by the simulation results: firstly, the multi-path routing request can reduce the chances to get in *void* areas (Figure 3.63); secondly, it appears to be more suitable for the dynamic networks (Figure 3.64 and Figure 3.65). The last two figures are the snapshots in dynamic networks. The gray edges in these two figures only demonstrate the speeds of movements. The blue points exist means that a routing path was found, but when it was drawn on these figures, the positions of the nodes in the routing path had been changed.

Note that, the Shawn simulation in this section is one of the ongoing works. The implementation methods and simulation results are not final.

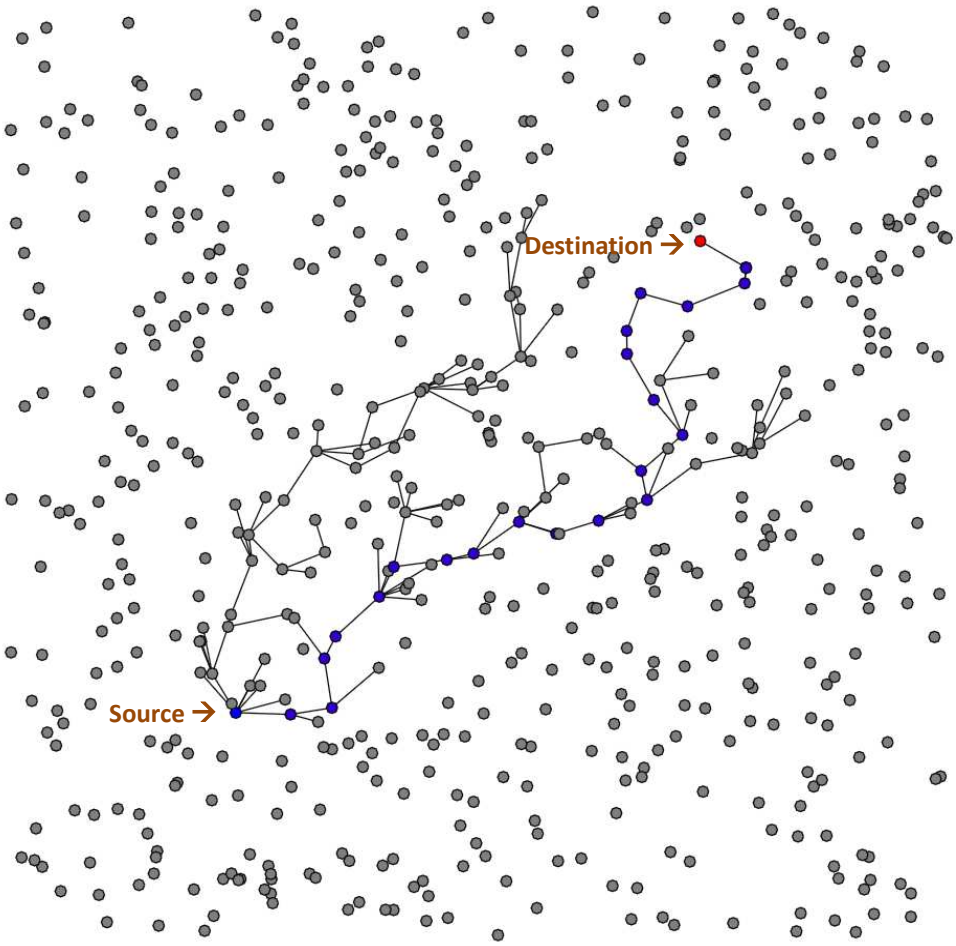


Figure 3.62: Routing request (multi-path) and reply (single-path) by CIVIC protocol

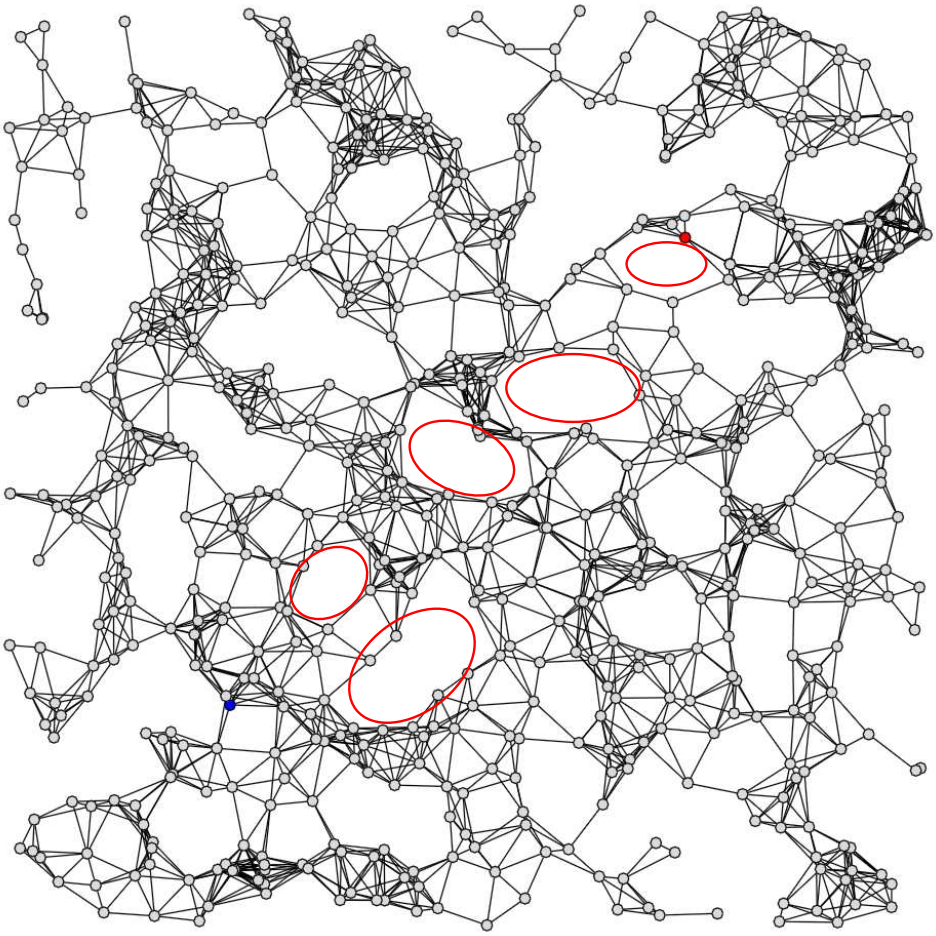


Figure 3.63: Full static network connections and void areas

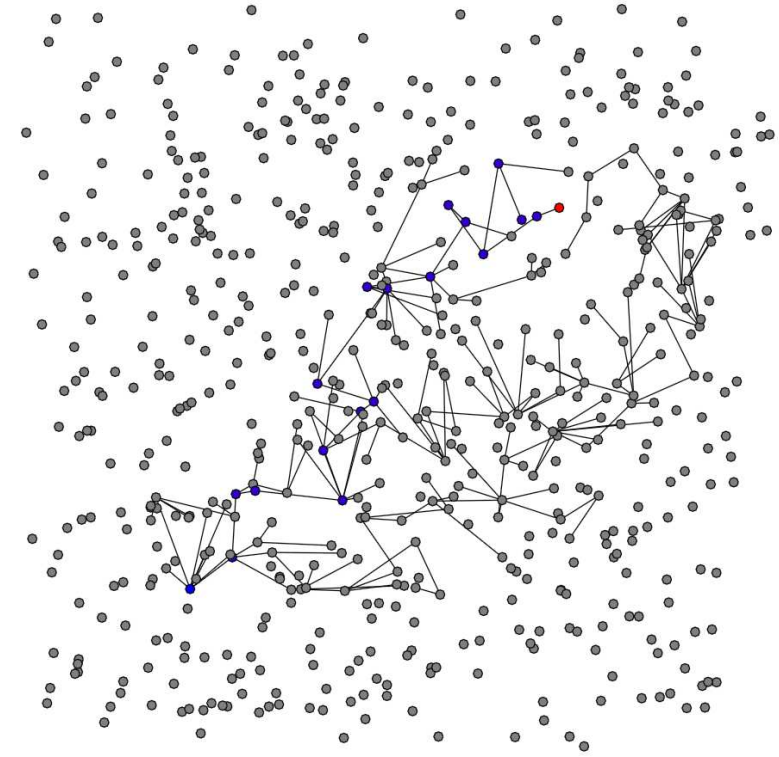


Figure 3.64: A routing path created in a slowly dynamic network simulation

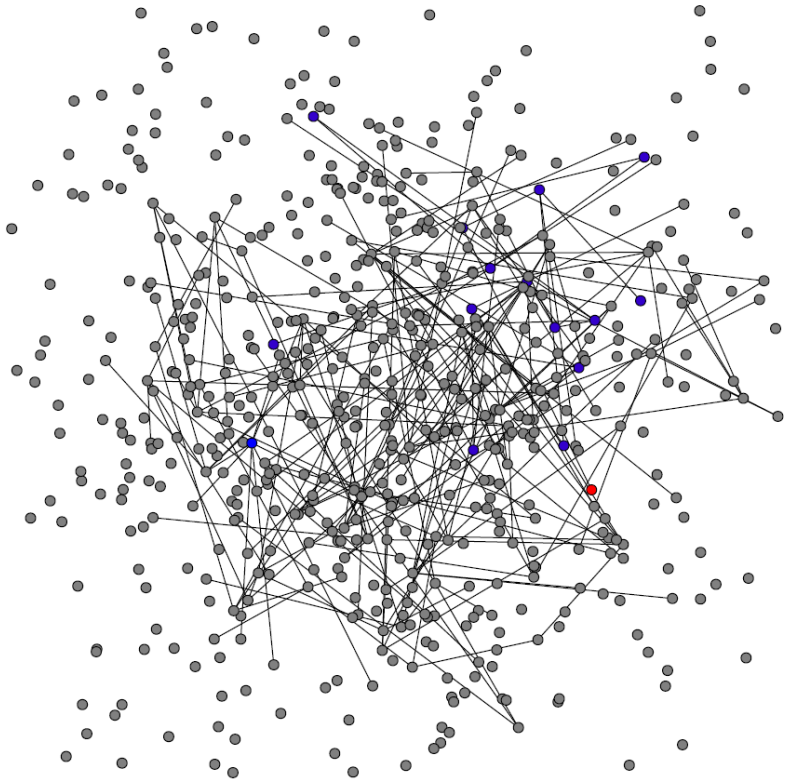


Figure 3.65: A routing path created in a highly dynamic network simulation

Chapter 4

Applications: Inter-vehicle Communication

This chapter is to show the results from real-world applications by field experiments (Section 4.2) and an IVC project named *MobiPlus* (Section 4.3). Before we give the results, we will first describe the software implementations for these applications (Section 4.1).

4.1 Software Implementation

As previously mentioned, the communication system in this thesis can be applied to the indoor WSN applications such as smart home, telemedicine and civil structure monitoring, thus it is important to know the indoor disturbance factors that affect the communications. For example, in indoor environments, the multipath phenomenons caused by reflection and scattering could be more obvious than outdoor environments; in outdoor environments, the wireless conditions are variable and complex because there could be unexpected interference sources and more serious wireless disturbances (e.g. absorption, refraction and diffraction).

A part of indoor experiment results are shown in "Theoretical Evaluations" (Section 3.6) and "LRPC Experiments" (Section 3.7). The indoor experiments in this section were conducted in the LIMOS (Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes) laboratory at ISIMA (Institut Supérieur d'Informatique, de Modélisation et de leurs Applications). This section also shows the results from two outdoor experiments. The first experiment was carried out at the car park of ISIMA campus. The next experiment was performed on PAVIN platform (Plate-forme d'Auvergne pour Véhicules Intelligents).

The program of our communication system is mainly written in C language, except for the hardware-related parts in assembly language. The program can be compiled successfully by both ARM-ELF-GCC 4.4.2 [114][115] and IAR C/C++ Compiler for ARM 4.40A [96]. All the assembly modules, C modules and header files are listed in Appendix B. This section only explains some modules needed by the following descriptions.

Figure 4.1 and 4.2 show the layer-based implementation of the communication system. The former provides more details about the message flow of the CIVIC protocol, and the latter provides more details on the task control of the HEROS operating system.

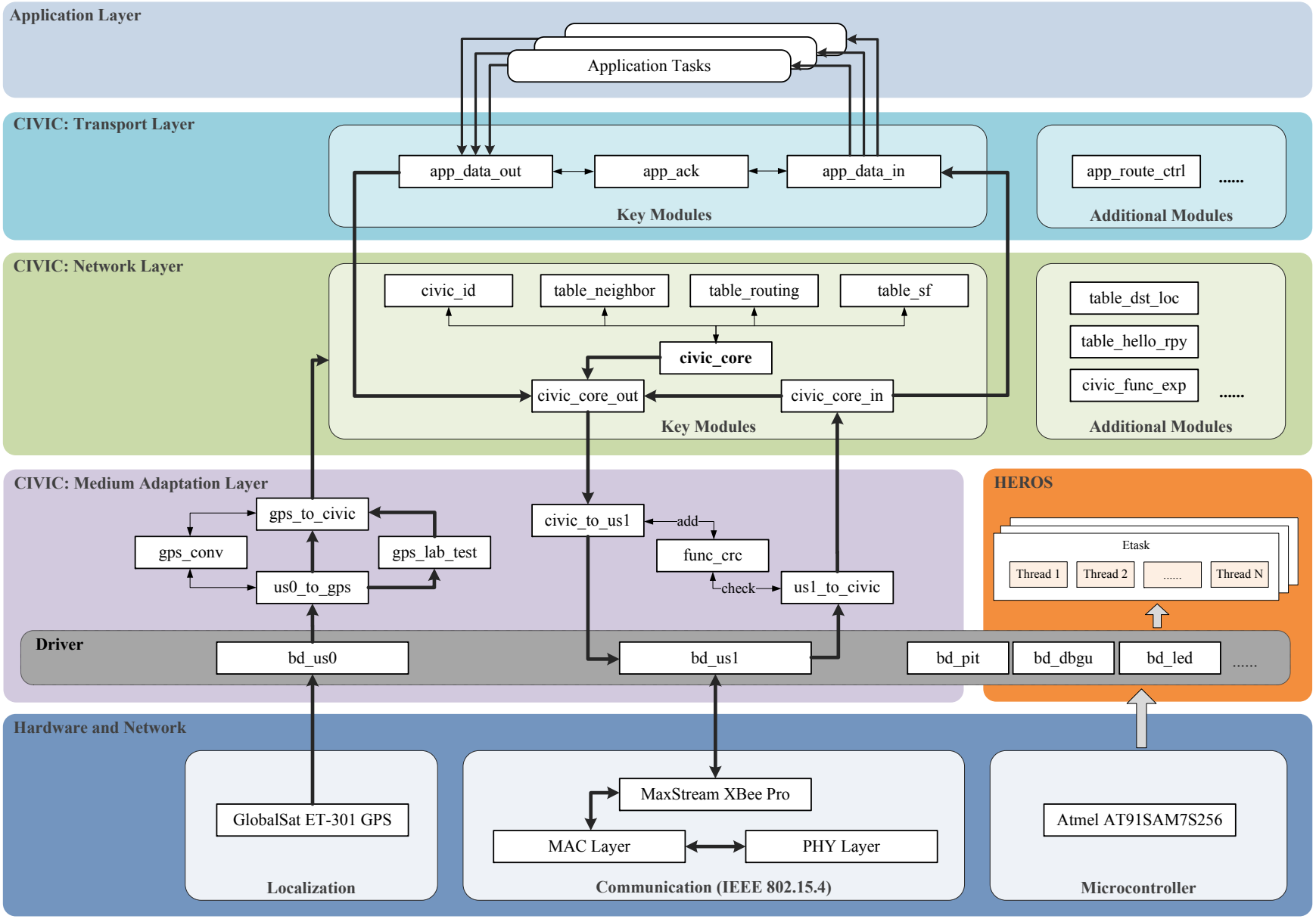


Figure 4.1: Layer-based Message Flows

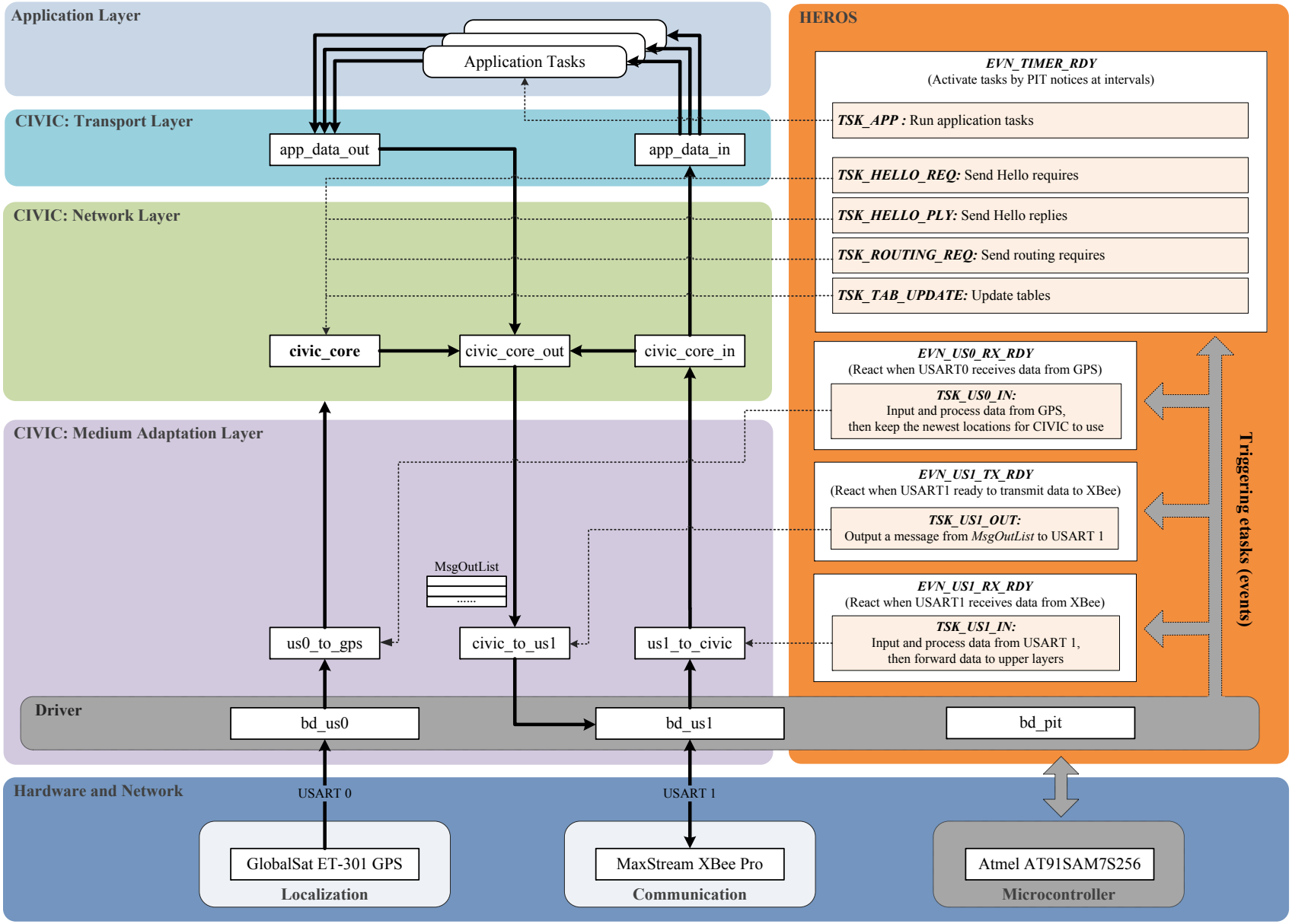


Figure 4.2: HEROS Communication Tasks

A CIVIC layer contains a stand alone group of C modules, and provides simple input/output message interfaces in the header files. There are two type of interfaces: one is for intra-layer usages, one is for inter-layer usages. A higher layer (or the task center) can control a lower layer by the inter-layer interfaces (the header files in "ext" folder of Appendix B). Note that, the layer division of the CIVIC protocol is based on the implementation of the message delivery mechanism that is suitable for low-resource embedded system, thus it is not providing the complete functions like OSI or TCP/IP layer models.

The key modules about the network layer that will be introduced here is *civic_core*, *civic_core_out*, *civic_core_in* and *civic_id*: The first module contains the settings of real-time task will be used by the HEROS. The *civic_core_out* handle two output functions:

- Generating and sending a new message based on the message format defined in Section 3.2.2
- Forwarding a routing message if it is not sent to itself

The *civic_core_in* deliver message from the medium adaptation layer to the correct directions:

- If it is a hello or routing message, passing it to the module in current layer.
- Is it is an application message, passing it to the transport layer.

The *civic_id* handles ID-related functions. Besides, the table-related C modules have been introduced in Section 3.2.2.3.

The current medium adaptation layer has two major parts: The first part converts information from GPS (USART 0) to the variables (in *gps_to_civic*) that CIVIC can directly read. The functions for coordinate conversions are in (in *gps_conv*). The modules of first part is driven by the *us0_to_gps* with is controlled by an event-driven task of the *USART 0*. The *gps_lab_test* is an optional module. For the indoor experiments to simulate a mobile scenario, the location from GPS is replaced with a random location. This module does these conversion works.

The second part handlers the messages to/from the network layer. The *civic_to_us1* handle the sending, and it is controlled by an event-driven task of *USART 1*. The *us1_to_civic* handle the receiving, and it is also controlled by an event-driven task of USART 1.

In the following, the etask and threads settings of HEROS are given. There are nine basic tasks implemented as threads in the following experiments, and they belong to four etasks.

- *TIMER_RDY* etask:
  - *Application*: The custom tasks for applications, for example, reactive routing path searches and sending real-time data. For most of the following experiments, the task controls the sending of locations from GPS. It drives the message flow to lower layers until reaching *MsgOutList*.
  - *Hello Request*: Sending Hello requests by *civic_core*. It drives the message flow to lower layers until reaching *MsgOutList*.
  - *Hello Reply*: Sending Hello replies by *civic_core*. It drives the message flow to lower layers until reaching *MsgOutList*.

  - *Routing Request*: Proactive routing path searches by *civic_core*. It drives the message flow to lower layers until reaching *MsgOutList*.
  - *Table Update*: Clear the outdated nodes from the routing table and the neighbor table at intervals. Besides, the clearing may also include the Hello reply table and the destination table by *civic_core*.

- *US1_TX_RDY* etask:

  - *Message Out*: When the last message output is finished, an event notice is issued from the interrupt *US1_TX_RDY*, and then the functions in *civic_to_us1* are activated to output the next message from *MsgOutList*. This task is also controlled by the etask *TIMER_RDY* to avoid a too short sending interval. It drives the message flow to lower layers.

- *US1_RX_RDY* etask:

  - *Message In*: Input raw message from XBee Pro by *us1_to_civic*. It drives the message flow to upper layers.

- *US0_RX_RDY* etask:

  - *GPS In*: Input raw message from GPS by *us0_to_gps*. It also drives the message flow to upper layers until the network layer.

If not specified, the real-time experiments in the following sections use the above implementations. The memory consumptions after implementing the above tasks are shown in the Table 4.1. When it is without the optimization from compiler, all together they take about 39 KB memory to run. There are six parts in the table from the minimum to the maximum summary memory consumption.

	App. Task	**GPS**	**MAL***	**CIVIC**	**HEROS**	**Misc.**
Code	256	2920	1920	5708	2744	15248
Data	33	226	512	1048	5928	168
Const	1	175	1377	58	356	372
Sum	290	3321	3809	6814	9028	15788

* *MAL*: Medium Adaptation Layer. It is not including the GPS part.

Table 4.1: Memory consumption of the communication system (unit is byte)

The sizes of some parts could be changed depending on applications:

- The application tasks in most our following experiments are a simple one. It sends an additional GPS data to a destination node at intervals.
- The size of GPS part includes the replacement of random locations. If requiring more GPS information, the size of this part will be also increasing.
- The size of Heros part will grow if more threads/etasks are added.

- In all the parts, if the sizes of tuples (or buffers) are increased, the sizes of *Data* will also be increased.
- The misc part mainly includes the math and string functions which are used by the parts of GPS and CIVIC to calculate and convert the data relating to coordinate, direction, speed, angle and distance.

4.2 Field Experiments

4.2.1 Indoor Multi-sensor Experiments

The first experiment is to test the embedded communication system in a simulated highly dynamic network. Nine sensors (LiveNodes) are used. A sensor is set as the destination node. The other sensors keep sending routing requests and application data to this destination sensor. The maximum interval of *US1_TX_RDY* for message sending is set to be in 100 ms. We implement all mechanisms of our communication system for the experiment.

The locations from GPS are replaced by random ones because it is difficult to get correct GPS in an indoor experiment. The locations to which are replaced are the random locations within a car park (about 80x60 meter squares) of our campus. The random mobile speed is from 0 to 30 meters in one second to simulate a highly dynamic network.

Moreover, to simulate the radio radius in the indoor experiment, sensors only response to messages sent from the distance less than 50 meters. Because each sensor actually receives all messages from the network, it requires more operations by both CIVIC and HEROS. This experiment focuses on testing the overall network performance.

The experiment contains four identical sub-tests, each runs 15 minutes, and then gets average values. Table 4.2 indicates the network performance. The loss rate is calculated based on the missing serial number. When a message outputs to the network by a sensor, the CIVIC protocol attaches a serial number (one byte) to indicate its place in the message sequence of the sensor. By monitoring the serial number, we know the message loss rate for every sensor.

	Average per sensor	**S0**	**S1**	**S2**	**S3**
Message Rate (msg/sec)	**3.41**	2.46	3.63	3.67	3.57
Loss Rate (%)	**4.13**	0.85	6.54	9.84	1.02
Message Number (in 15 mins)	**2973**	2207	3074	2995	3203
	S4	**S5**	**S6**	**S7**	**S8**
Message Rate (msg/sec)	3.51	3.14	3.9	3.64	3.2
Loss Rate (%)	1.51	3.94	2.77	3.33	7.34
Message Number (in 15 mins)	3130	2734	3536	3190	2689

Table 4.2: Individual sensor status in an indoor experiment

In order to monitoring the network in a graph interface and analyze messages in a more convenient way, a PC software is developed specially for our experiments. The network screenshots and result analysis of this section are from the software.

In general, most of sensors are sending routing requests and application data to the accurate direction. Figure 4.3 shows an example of CIVIC routing topology. The *S0* at the corner is the destination node.

The message loss rate in this experiment should be caused by network traffic overhead, interferences, shading and message collisions. As previous mention, all nine nodes are actually receiving message from the whole network, so they have the same conditions of the incoming network traffic. Moreover, the mobile speed in the experiment is very high, so it is very easy for a node to be running out of the radio radius (e.g. the *S7* in Figure 4.3).

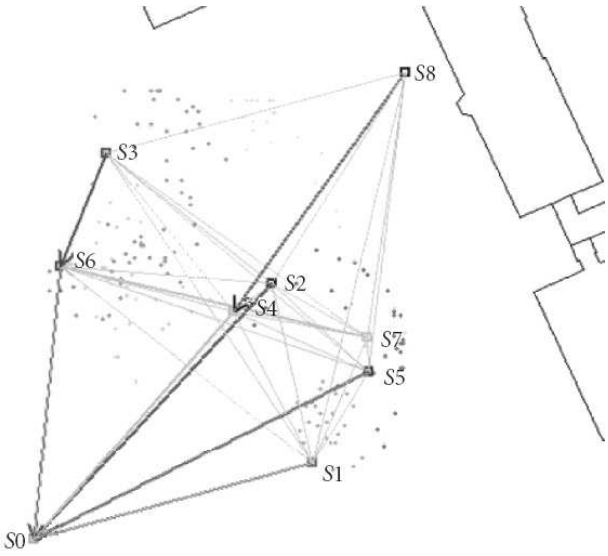


Figure 4.3: Data sending flows to a destination node

4.2.2 Car Park at ISIMA Campus

This experiment was actually done in the car park of our campus as shown in Figure 4.4. The purpose of our experiments is to implement the communication system on a outdoor sensor network, test its feasibility and make it reliable. At the same time, the experiments serve also to validate LCD-GPS (Section 3.2.1.2 and 3.4.4).

Figure 4.4: Location of the experiment in a parking area

All LiveNode sensors in the experiment are randomly distributed on the tops of cars. A sensor at the corner is set as the receiver of routing request and application messages (original GPS information). The network is monitored by four ZigBee base stations, which are also randomly distributed. The Figure 4.5 demonstrates the network deployment when using nine static sensors.

Because the outdoor radio range of ZigBee Pro is up to 1.6 km and the space available to our experiments is limited, we define a filter for each sensor to remove the packet sent from the distance longer than 40 meters. It forces some sensors to send data in multi-hops, so we can evaluate the efficiency of packet forwarding in CIVIC protocol. Because each sensor actually

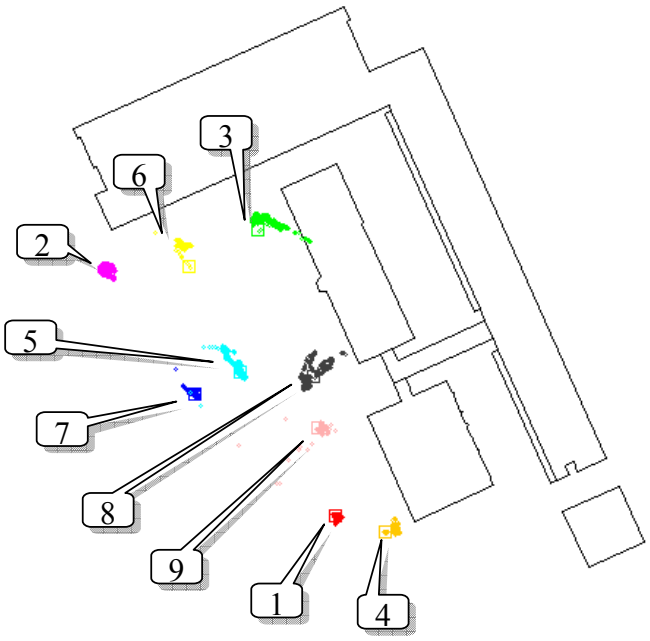


Figure 4.5: Network deployment for nine sensors

receives all packets from the network, if CIVIC protocol can work under this setting, it should perform better in applications that are more practical.

4.2.2.1 First Experiment Scenario

The first experiment is to compare the DANKAB broadcast with the SF (simple Flooding) broadcast in the routing process of the CIVIC protocol. There are reasons to choose SF in this experiment: SF is reliable in terms of coverage, and it has been practically used in low-density networks. Besides, to compare with two broadcasting methods, they must be implemented on the same hardware platform, and run under the same conditions. Resource constraints of embedded sensor limit the choice of broadcasting methods.

This experiment is done in a static network with nine sensors. The parameters used for evaluations are the overall packet number, the packet loss rate, and the average routing hop distance (the maximum hop is four). In both broadcasts, at five seconds intervals, eight sensors require routing to a single sensor at the corner. Both DANKAB and SF broadcast stop when 50 routings are found.

	Message Number	**Message Loss Rate**	**Average Routing Hop**
SF	3453	0.0913	2.1
DANKAB	1144	0.0297	1.39

Table 4.3: Comparing SF and DANKAB routing approaches

The result of comparison is shown in Table 4.3. The DANKAB has better performance in all three parameters. Moreover, comparing to SF routing paths as shown in Figure 4.6, all DANKAB routing paths are on the correct direction to the destination node as shown in Figure 4.7.

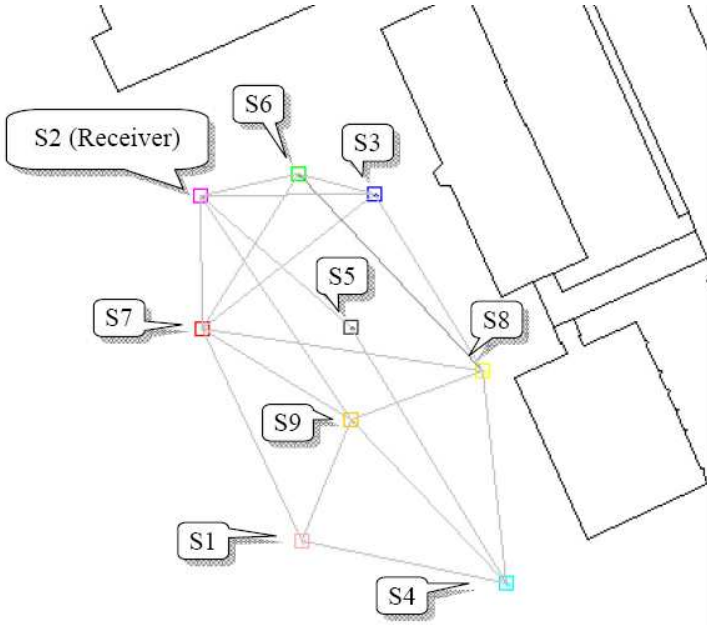


Figure 4.6: Routings topology by SF

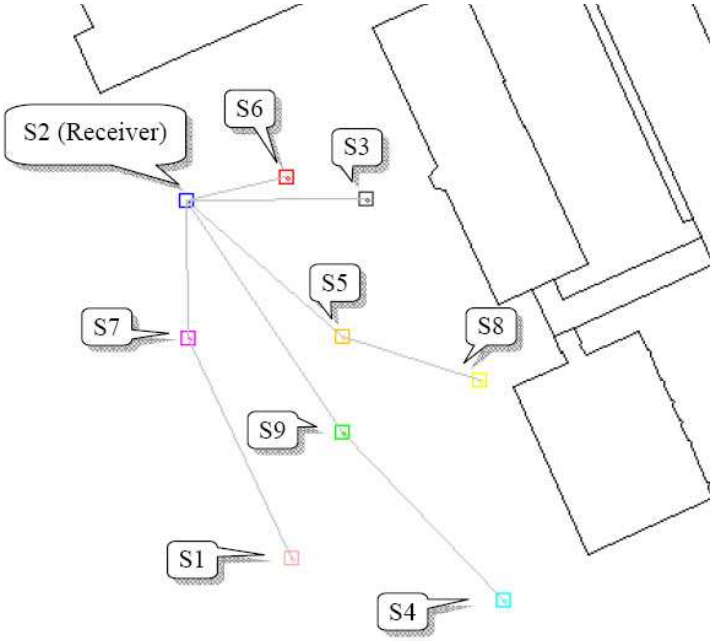


Figure 4.7: Routing topology by DANKAB

4.2.2.2 Second Experiment Scenario

The second experiment is a comparison of the application packets sending with or without acknowledgement.

The network deployment and the target sensor are the same as in the first experiment. The experiments are run after all sensors obtain routing paths by DANKAB. At one-second intervals, sensors send their original GPS data to the target sensor. The GPS data are wrapped in the application messages. The experiments stop when 100 application packets are sent, thus ideally there should be 800 application packets received by the target sensor.

In the experiment without acknowledgement, all sensors just keep sending until finished. In the experiment with acknowledgement, if an acknowledgement has not been received, the sender assumes that the target sensor is busy, and waits three seconds before sending the data again.

This target sensor (S2) is connected to a laptop by its debug port. The main parameters for comparison are message number received by debug port, and the overall packet number received by ZigBee stations (including Hello and routing messages). The result is shown in Table 4.4.

	Received Message (debug port)	**Receiving Rate**	**Overall Message Number**
No Ack	773	97%	1043
With Ack	851	100%	2260

Table 4.4: Comparing the efficiency with or without acknowledgement

4.2.2.3 Third Experiment Scenario

The third scenario is to evaluate the factor of mobile sensor. It is done in a network with nine sensors. This experiment compares a static network with nine sensors and a similar network with one mobile sensor as in Figure 4.8.

	Received Number (debug port)	**Receiving Rate**
Static Network	773	97%
Mobile Network	652	82%

Table 4.5: Comparing the factor of mobile sensor

The result in Table 4.5 shows that a part of the application data is lost because of the sensor movements, but overall the CIVIC protocol can perform well in a non-static network. The data analysis after experiments indicates that the packet loss is mainly caused by two reasons:

- It is caused by the radio interference.
- The interval for routing request is set to be too short, so the new routing path can not be updated in time.

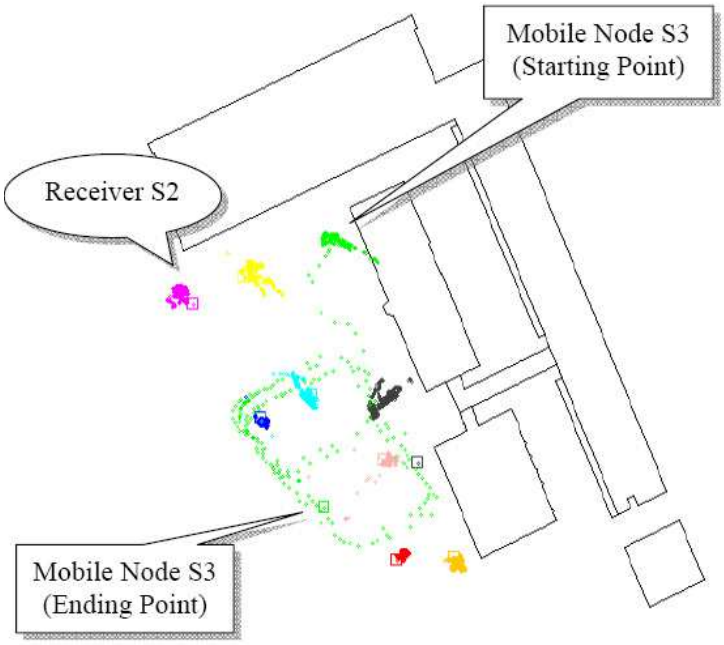


Figure 4.8: Experiment with a mobile sensor

4.2.3 PAVIN Platform

To demonstrate the feasibility and reliability of the CIVIC protocol in real-world mobility scenarios, we have performed new experiments on the PAVIN platform Figure 4.9, which is an experimental site dedicated to the development of intelligent autonomous vehicles. The purpose of our experiments on PAVIN platform is to test the feasibility and reliability of the CIVIC protocol on in the mobility scenarios of urban contexts. The experiments in this paper were conducted between 2pm and 4pm on April 23, 2009. The experiment duration was about 90 minutes.

Figure 4.9: PAVIN platform

The PAVIN platform is a 4750 m2 experimental site located in the Blaise Pascal University (France). It contains roads and buildings similar to urban areas. An electric vehicle called Cycab Figure 4.10 was used to carry the mobile node moving along the roads in platform during experiments.

Figure 4.10: A Cycab with LiveNode in PAVIN platform

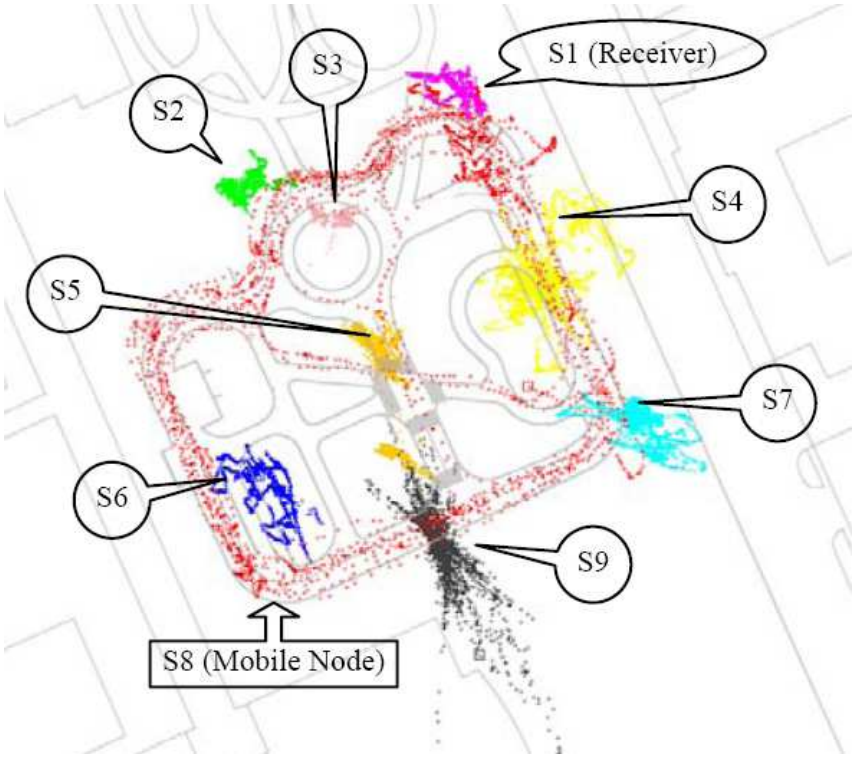


Figure 4.11: Location changes during experiments

There were nine sensors used in PAVIN experiments. The Figure 4.11 shows the network deployment and the position changes of sensors during experiments. A sensor at the corner (S1) was set as the receiver of routing request and application data. A sensor as the mobile node (S8) was put on top of the Cycab for mobile tests. The other sensors were randomly distributed

on the platform. The position changes could be caused by actual sensor movements or just inaccurate GPS information.

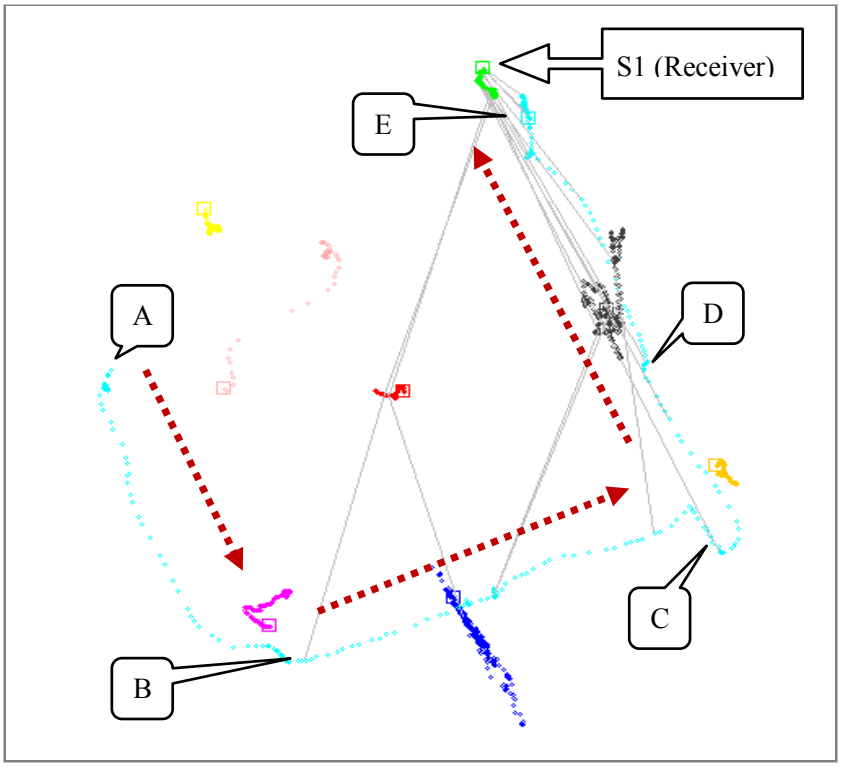


Figure 4.12: CIVIC protocol performing the DANKAB routing requests

The Figure 4.12 demonstrates the typical examples of CIVIC routing processes passing through the mobile sensor. In this scenario, the Cycab carrying sensor S8 is driven from point A to point E. It requests routing every five seconds intervals starting at point B. The lines in this figure indicate the routing requests or routing replies of the CIVIC protocol by using DANKAB. From point B to C, the multi-hop routing processes are correctly performed in the right direction to the receiver (S1). Between point D and E, sensor S8 is in one-hop distance from the receiver.

Table 4.6 summarizes the overall network status in the experiments for about 90 minutes. The error rate for packets is evaluated by checking the CRC in packets received by the ZigBee station, thus it is different from the ones that are calculated by the sequence number in packets.

	Received Characters	**Received Messages**
All Received	2239740	45016
Correctly Received	N/A	43823
Error Rate	N/A	0.0265
Data Rate	415 character/sec	8.3 message/sec

Table 4.6: Overall network status

Table 4.7 indicates the status for individual sensors. The error rate is calculated by the missing serial number. When a sensor sends a packet in our experiments, it attaches a serial

number (one byte) to indicate its place in the packet output sequence. In this table, the sensor S3 gets higher error rate due to poor power supply.

	Average per sensor	**S1**	**S2**	**S3**	**S4**
Message Rate (msg/sec)	**0.9**	1.09	0.94	0.51	0.90
Loss Rate (%)	**3.97**	3.15	2.97	17.17	2.81
Message Number (in 90 mins)	**4869**	5867	5059	2779	4871
	S5	**S6**	**S7**	**S8**	**S9**
Message Rate (msg/sec)	0.97	0.95	0.89	1.01	0.86
Loss Rate (%)	2.16	1.52	1.64	2.85	1.50
Message Number (in 90 mins)	5250	5125	4795	5427	4650

Table 4.7: Overall network status

4.3 MobiPlus Project

The CIVIC protocol is used as a prototype to experiment three projects in different areas: inter-vehicle communication (MobiPlus project), environmental data collection (Net-ADDED European project) and telemedicine (LiveCare project). In this paper, we present only the MobiPlus project.

MobiPlus project is supported by the SMTC (Syndicat Mixte des Transports en Commun de l'agglomération clermontoise) of Clermont-Ferrand city in France. It focuses on improving the public service on urban transportation system particularly to disabled passengers Figure 4.13.

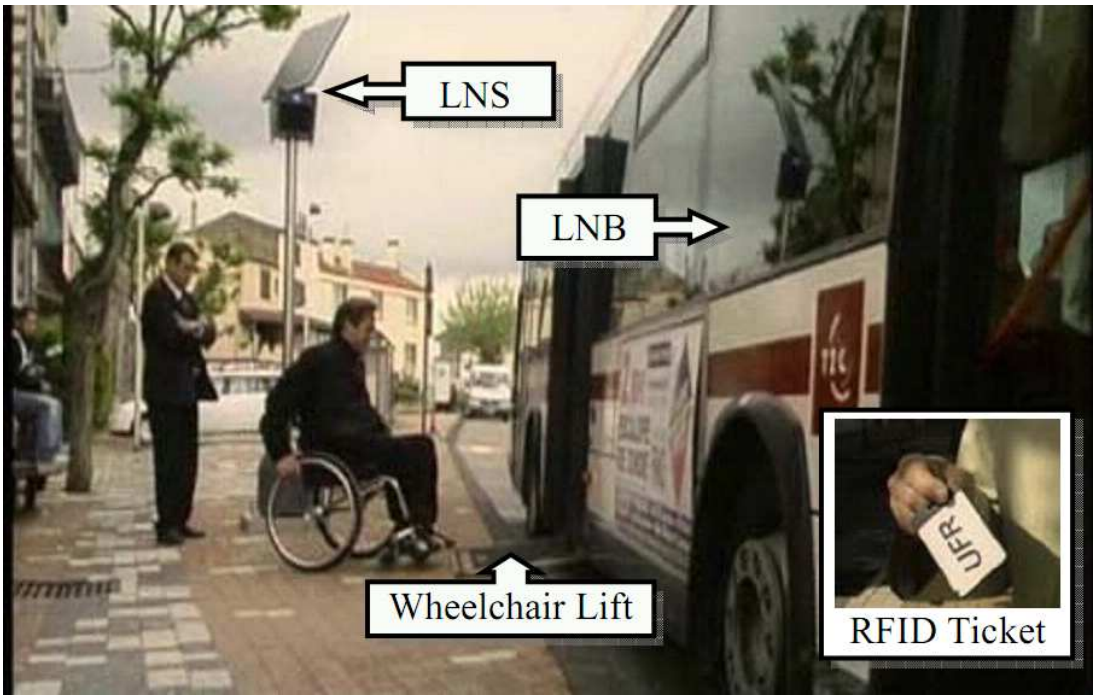


Figure 4.13: MobiPlus project in Clermont-Ferrand (France)

The MobiPlus has two major components: LNB (LiveNode Bus) and LNS (LiveNode Station). The LNS contains an RFID (Radio-Frequency IDentification) reader, which detects the presence of the disabled passenger who has an RFID electronic ticket. The RFID electronic ticket contains the information about the specific needs of the passenger, and this information will be sent to the bus that he or she is waiting for. Thus, when LNB arrives at LNS, related services according to these specific needs will be provided. For example, if a wheelchair user is present, the wheelchair lift on LNB will be activated. If a sight-deprived passenger is present, the voice notice from LNS will be played.

The LiveNode sensors embedded at LNB and LNS communicate with CIVIC protocol, which adopts Wi-Fi and ZigBee. The GPS is used to localize the LNB and to estimate its arrival time.

Chapter 5

Applications: WSN Precision Agriculture

5.1 An Additional Auto-clustering Algorithm for WSNs

5.1.1 Concept and Scenario

As previously mentioned, our embedded communication system targets two network fields: VANET and WSN. They are both high-dynamic networks but the changes of network topologies are by different reasons. The former is because the high-mobility features of network nodes (vehicles); the latter is mainly because the resource constraints of network nodes cause the faults in network nodes (wireless sensor). Besides, the faults are caused by unanticipated environment variations. The deployment of WSNs is hard to predict, and in some cases, sensor nodes are randomly deployed by aircraft. Owning to the features of WSNs, the communication system is designed with an auto-clustering protocol. The auto-clustering protocol has two usages in the communication system: it can be a part of the routing approaches of the CIVIC protocol when it is used in WSNs; it can be implemented as a part of the management protocol in the application layer. The thesis focuses on the first usage; for the second one we will only be briefed in this section.

As a geographic routing protocol like CIVIC, to reduce the energy consumption on network communications and location operations is particularly important. WSNs are usually grouped by large numbers of low-cost sensor nodes with one or more sinks. A sink with high hardware capabilities holds the connection between sensor nodes and data proxy. The hardware characteristics of sensor nodes are similar to those of a tiny computer system, but they have to cope with stringent resource constraints in terms of CPU, energy, memory, bandwidth and transmission distance. In a cluster-based hierarchy routing scheme, most sensor nodes in clusters only route data within a local area and respond only to local commands. As for cross-cluster communications, only the selected sensor nodes in/between clusters are involved to the processes. Therefore, the amount of overall network communications is reduced, and consequently the energy consumption is decreased. Moreover, a cluster-based hierarchy scheme does not require the availability of location system on each sensor node, or at least, not an all time requirement. The requirement to have the locations of sensor nodes becomes the requirement of having the locations of clusters in most of time. When a cluster of sensor nodes is off duty, they can be switched to sleep mode to conserve more energy.

As for the management aspect of WSNs, it is impractical to manually initialize or

reconfigure hundreds of low-cost sensor nodes. The network topology in a WSN frequently changes because of the node faults. In some applications such as disaster monitoring and battlefield surveillance, the sensor nodes are often required to operate in dangerous environment where accessibility is highly restricted. The classic design approach to deal with these problems is to divide a network into clusters to enable management functions to be undertaken at local levels. Comparing to network management in most traditional wired and wireless networks, which is a software service that helps human manager monitoring network status and maintaining network performance, the network management in WSNs is the autonomous processes of organizing, monitoring and controlling all elements and services of a WSN [116]. Other advantages of a cluster-based hierarchy scheme can be found in [117].

The auto-clustering protocol in this thesis is capable of autonomously dividing sensor nodes into a set of single-level clusters using only locally-available information. As shown in Figure 5.1, bridge nodes between neighboring clusters are introduced to assist with the tasks of cluster formulation, and the tasks of routing after the clusters are formulated. Each cluster has a master node on duty, which is responsible for routing, coordination, and aggregation. Other sensor nodes in the cluster, which are referred to as slave nodes, are on duty for sensing the environment.

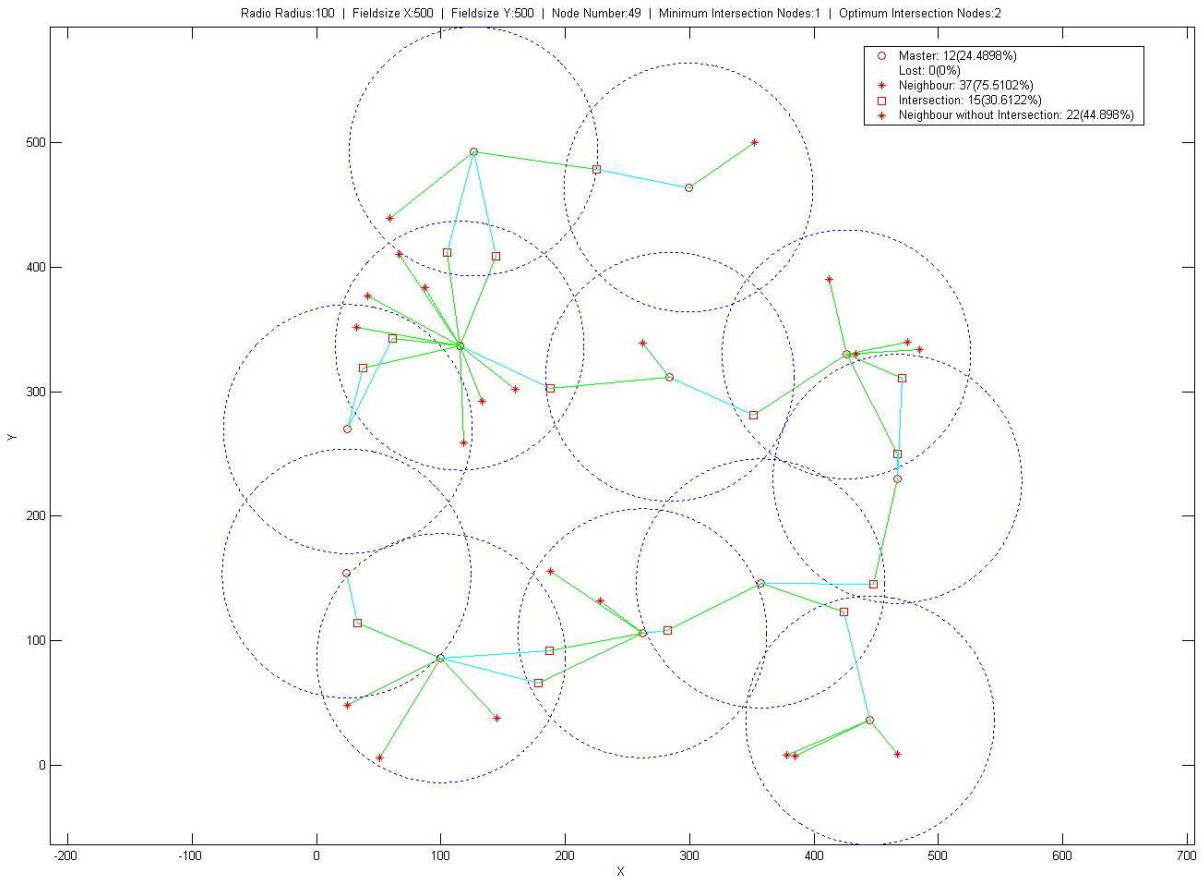


Figure 5.1: An example result of auto-clustering algorithm

5.1.2 Criteria for the Selection of Master Nodes

In this section, clustering model and two criteria for the selection of master nodes are presented to be followed by the centralized and localized auto-clustering approaches which are based on these criteria.

5.1.2.1 Clustering Model

A closer look of the network scenario is in Figure 5.2. It shows an example of the network scenario in hierarchy view. Sensor nodes A, B and C are master nodes; sensor nodes k, l, m, and n are slave nodes; sensor node i, j is bridge node, and S is the sink.

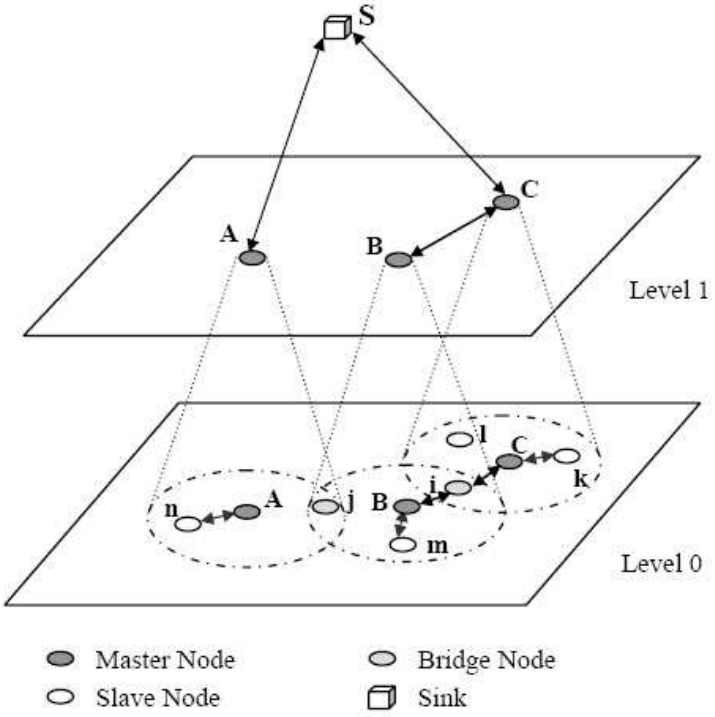


Figure 5.2: Auto-clustering network scenario

- A *Master Node* is responsible for coordinating the slave nodes, aggregating and routing data from slave nodes to the sink or other master nodes. In the hierarchical expression, master nodes are considered as the parent nodes of slave nodes and the children nodes of the sink.
- A *Slave Node* only performs data sensing operation. It periodically or responsively sends the monitored data to a master node. In the hierarchical expression, it is the end-node. The slave nodes communicate with the sink through their master nodes, for example the routing n-A, m-B and k-C.
- A *Bridge Node* is selected from the intersection of slave nodes in two neighboring clusters. A bridge node is a subset of slave nodes but only acts as the gateway of two neighboring clusters. The master nodes of two neighboring clusters can communicate with each other through their bridge nodes, for example the routing B-i-C.

- Unlike sensor nodes, the *Sink* has a large memory, high processing speed and sufficient power support. It connects master nodes and data proxy through Internet or satellite, e.g. the routing n-A-S, k-C-S and m-B-i-C-S. In the hierarchical expression, the sink is the root without superior. There is only one sink in the present network scenario.

The present clustering model is based on the following assumptions:

- Any sensor node can use the functions of the CIVIC protocol including:
  - Having the knowledge of the one-hop neighbor nodes
  - To make peer-to-peer multi-hop communications with sink and other sensor nodes
- At the initialization stage, the WSN is stationary, there is no node failures or channel errors.
- There is no lost node; in other words, all nodes must be a clustering role among master, slave or bridge nodes.

WSN is represented by an undirected and simple graph $G = (V, E)$. The vertices V is the set of sensor nodes, where each sensor node is equipped with an omnidirectional antenna. The edges E is the set of wireless links, where each edge connects exactly two distinct vertices, thus an edge also represents a neighbor relationship between two sensor nodes. Moreover, the set of clusters is represented as C.

The items in the following list denote individual elements in G:

- For following all, if not specified, i and j are both positive integers and they represent the IDs of vertices.
- A vertex is represented as v_i. The number of vertices (*N*odes) is $N = |V|$, and $\bigcup_{i=1}^{N} v_i = \{v_1, v_2, v_3, ..., v_N\} = V$.
- A edge is represented as e_{ij}. It is an undirected link between v_i and v_j. The number of edges (*L*inks) is $L = |E|$, and $\bigcup_{i=1}^{L} e_i = E$.
- A cluster is represented as c_i. It implicates v_i is the master node of c_i. The number of clusters (*B*unchs) is $B = |V|$, and $\bigcup_{i=1}^{B} c_i = C$.
- A vertex acting as a master node is shorten as a master and denoted as m_i. Likewise, a bridge is b_i, and a slave is s_i.

Normally, a smaller value of B represents a better clustering result. If M is the maximum number of vertices in a cluster, by considering the third assumption, the minimum and maximum values of B is:

$$\frac{N}{M} \leq |B| \leq \frac{N}{2} + 1$$

5.1.2.2 Criteria for the selection of master nodes

Suppose a original vertex v_i is selected to be a master, an other vertex v_j which fulfills the following two criteria will also qualify for selection as a master.

Criterion 1

$$n_j \notin c_i$$

There is no two master nodes in a same cluster. For some applications with R is the radio range (assuming it is a circle), the distance between two masters must be greater than R.

Criterion 2

$$1 \leq |c_i \bigcap c_j| \leq 2$$

If two vertices v_i and v_j are selected as masters, there must also be at least one vertex node connecting them. Consequently, one or two vertices will be selected to act as the bridge nodes in order to provide the link between the masters.

After these two criteria, if more than one vertex can be selected as the next master, use the one with a higher ID number then continuing.

5.1.3 MATLAB Simulations

The clustering algorithm has been implemented in MATLAB simulation by two approaches: the centralized approach and the localized one. The MATLAB simulation simply prove the feasibility of the previous two criteria without considering the network capability and packet collision, and the result has been published in [118]. The further development and simulation on the *Shawn* simulator were done by Wu in [14].

5.1.3.1 Centralized Approach

The computing in centralized approach is assumed to be done by a sink, thus the neighbor information must be sent from nodes to the sink by multi-hop transmissions.

After the network has been deployed, each node open a time window to exchange the neighbor information. In the beginning of the time window, each node broadcasts a one-hop HELLO messages to notice their neighbors about its existing. Nodes know their neighbors by the HELLO messages and send out the neighbor lists to the sink in the end of the time window. The sending of the neighbor lists can be assumed to be carried out by a geographic routing protocol, for example, the previously mentioned CIVIC protocol.

When the collection of neighbor lists is finished, the sink then randomly starts the computing from a sensor node, which is assumed to be the first master node, and then selects the other master nodes based on the two criteria until all nodes are clustered. Since the sink is assumed to have the high hardware capabilities and get the information covering the whole WSN, the processing steps can be optimized and calculated repeatedly until the sink achieves the best clustering results.

When the clustering process has finished, the sink broadcasts confirm messages to selected master nodes and bridge nodes that include the topology routing information, and then the selected master nodes notice the slave nodes.

5.1.3.2 Localized Approach

The initial steps of the localized approach are similar to those outlined for the centralized algorithm: a time window is opened for exchanging HELLO messages. But instead of letting the sink to do the clustering computation, the nodes with the higher ID numbers compute the localized clustering results in the end of the time window, e.g. nodes 6, 10, 11 in Figure 5.3(a). These nodes have the highest ID numbers by comparing with their one-hop neighbors' ID numbers, and they are called the initial master nodes.

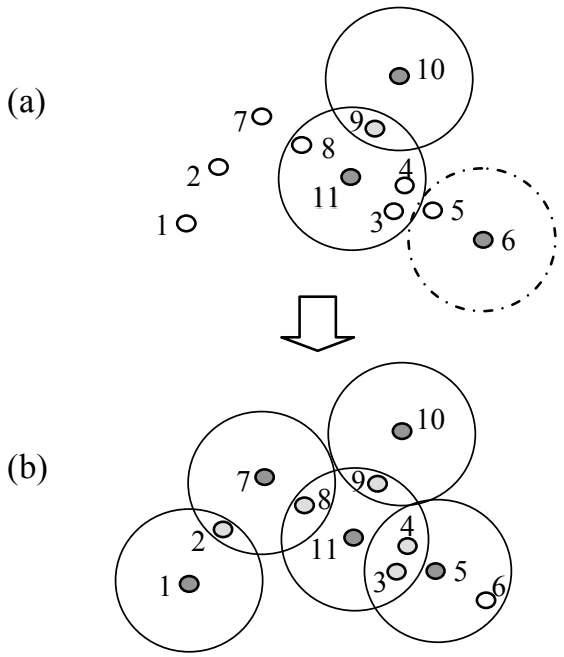


Figure 5.3: An example of master node selection

Since these initial master nodes have known their one-hop neighbors, they can ask these neighbors to sent back their neighbor lists. Based on the two-hop neighbor information, the initial master nodes can decide the next round master nodes by the previous two criteria. Then, the next round master nodes take on the clustering computation, and the clustering processes gradually spread over the network. For instance, the node 7 is selected to be the next master node of initial master node 11, and the clustering processes finish at node 1.

The advantage to have multiple initial master nodes is to deal with separate networks. If all nodes are assumed to be connected with each other, only one initial master node is needed by the localized approach. However, if multiple initial master nodes are existing, the separate clustering processes will meet each other definitively (e.g. node 9, and nodes from 3 to 5). Our solution for the MATLAB simulation is a simple one: the ID numbers from the initial master nodes are assumed to be carried by the clustering relating messages. If two separate clustering processes meet, the process carrying a higher ID number can stop the other process and reform the clusters in the other process. For example, node 5 becomes a master node in the second round because it passes the clustering relating message from initial master node 11, and then node 6 becomes a slave node of the new master node 5.

5.1.4 Simulation Results

Figure 5.4 shows the variation of the number of master nodes/cluster with the radio radius based on a total 200 nodes.

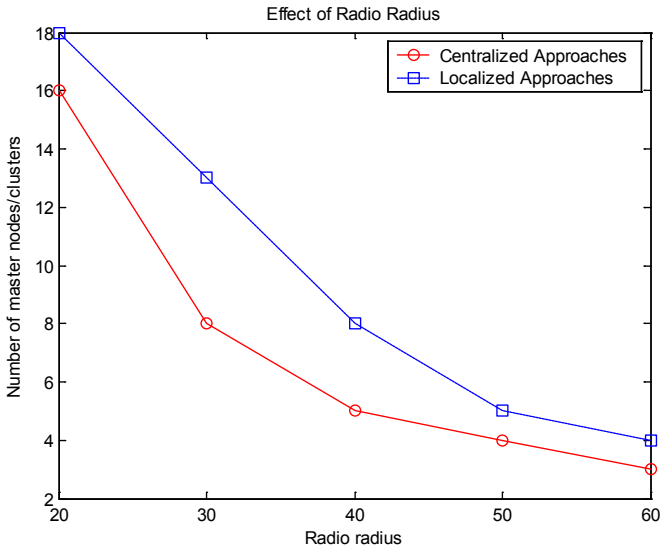


Figure 5.4: Number of master nodes/cluster vs radio radius

Figure 5.5 shows the variation of the number of master nodes/cluster with the total number of sensor nodes at a fixed 20 m radio range. Both set of results are based on a field area of $100 \times 100\ m^2$.

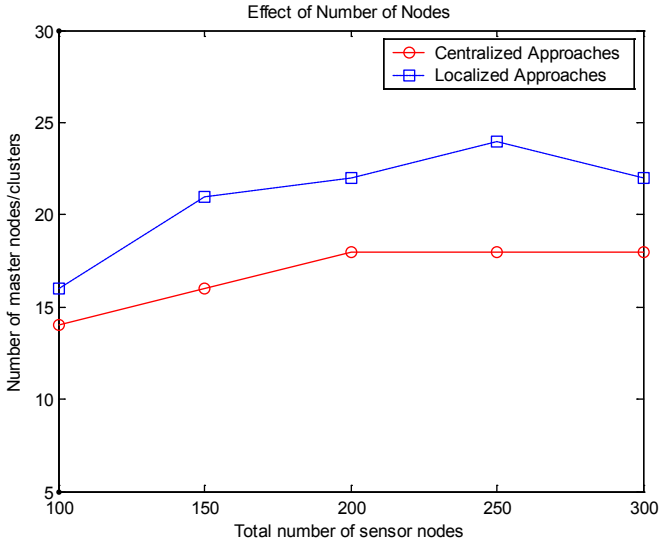


Figure 5.5: Number of master nodes/cluster vs total number of sensor nodes

The results prove both centralized and localized approaches are feasible, and the centralized approach always get the better results. However, note that although the simulated results on MATLAB appear to suggest that the centralized approach may be better than that of the

localized approach, the latter is more suitable for the real world applications because it is scalable because it is not always practical to send all neighbor lists to sink for computing. In the Figure 5.5 the number of master nodes/cluster starts to be moderate after 200 nodes because the coverage limitation is reached. The newly added nodes are in the coverage of the old ones.

The following Figure 5.6 Figure 5.7 Figure 5.8 shown in the end of this sub-section use the optimal centralized auto-clustering approach in a field of $500 \times 500\ m^2$ populated with 400 sensor nodes. The radio range is 65 m.

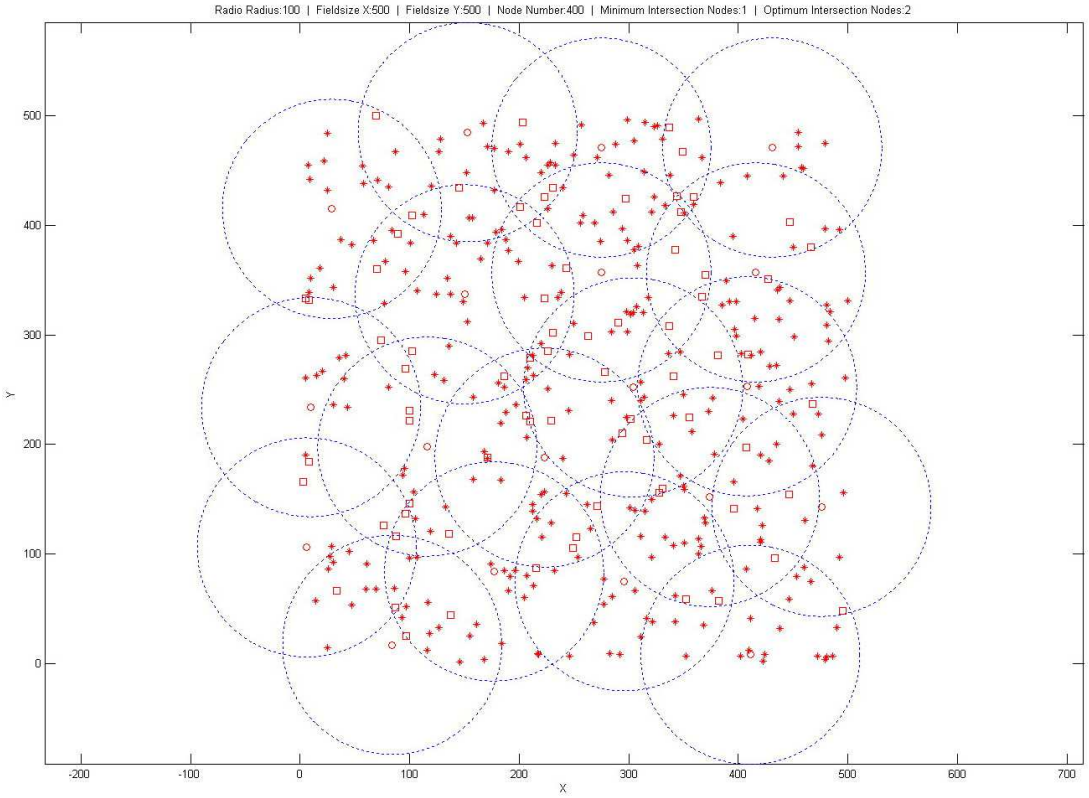


Figure 5.6: Cluster formulation

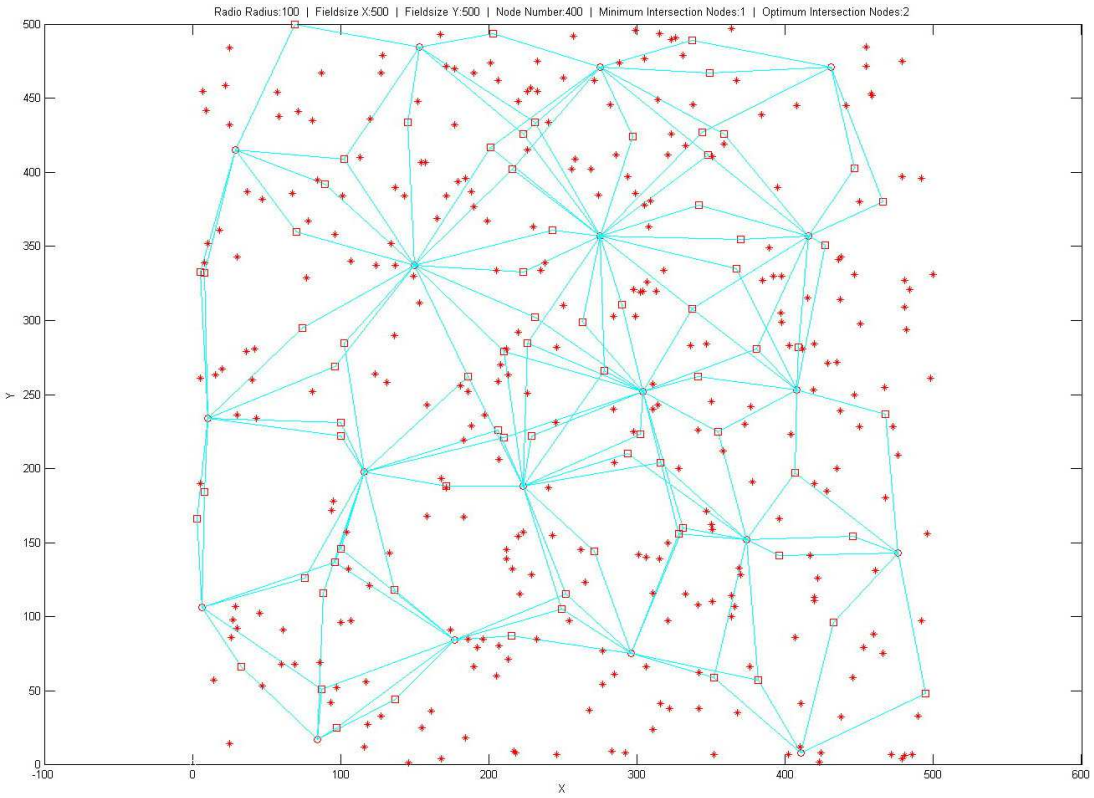


Figure 5.7: Inter-cluster network connections

.

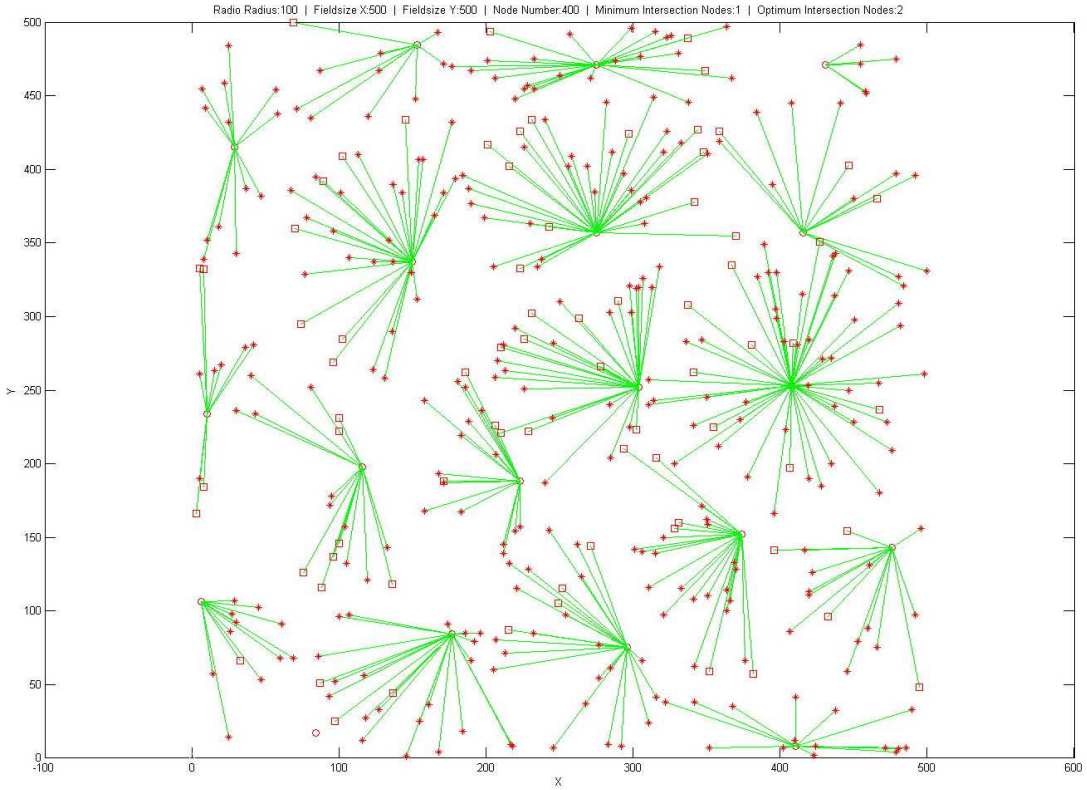


Figure 5.8: Intra-cluster connections

5.2 NeT-ADDED Project

The works in this chapter and the CIVIC protocol are used in the environmental data collection project *NeT-ADDED (New Technologies to Avoid Digital Division in E-Divided areas)*. The Net-ADDED project is a two-year *European Commission (EC)* programme to develop and validate technical features improving performance of deployment and operation of hybrid satellite-wireless technologies. There are twelve partners from different countries (France, Morocco, Greece and Turkey).The participation of our team (UBP/Cemagref) is in the field of precision agriculture, and our major task to use sensors (*LiveNodes*) to collect environmental data (e.g. soil moisture).

Our design is to develop a *VPN (Virtual Private Network)* platform called *LivePlatform* based on WSN technologies as shown in Figure 5.9. The design of component layers integrates the *WIN* concept [119], which supports the use of intelligent network capabilities to provide seamless terminal services, personal mobility services, and advanced network services in the mobile environment. Thus, it enables to extend the wireless recovery from an access point and eases the deployment of wireless network in the rural area.

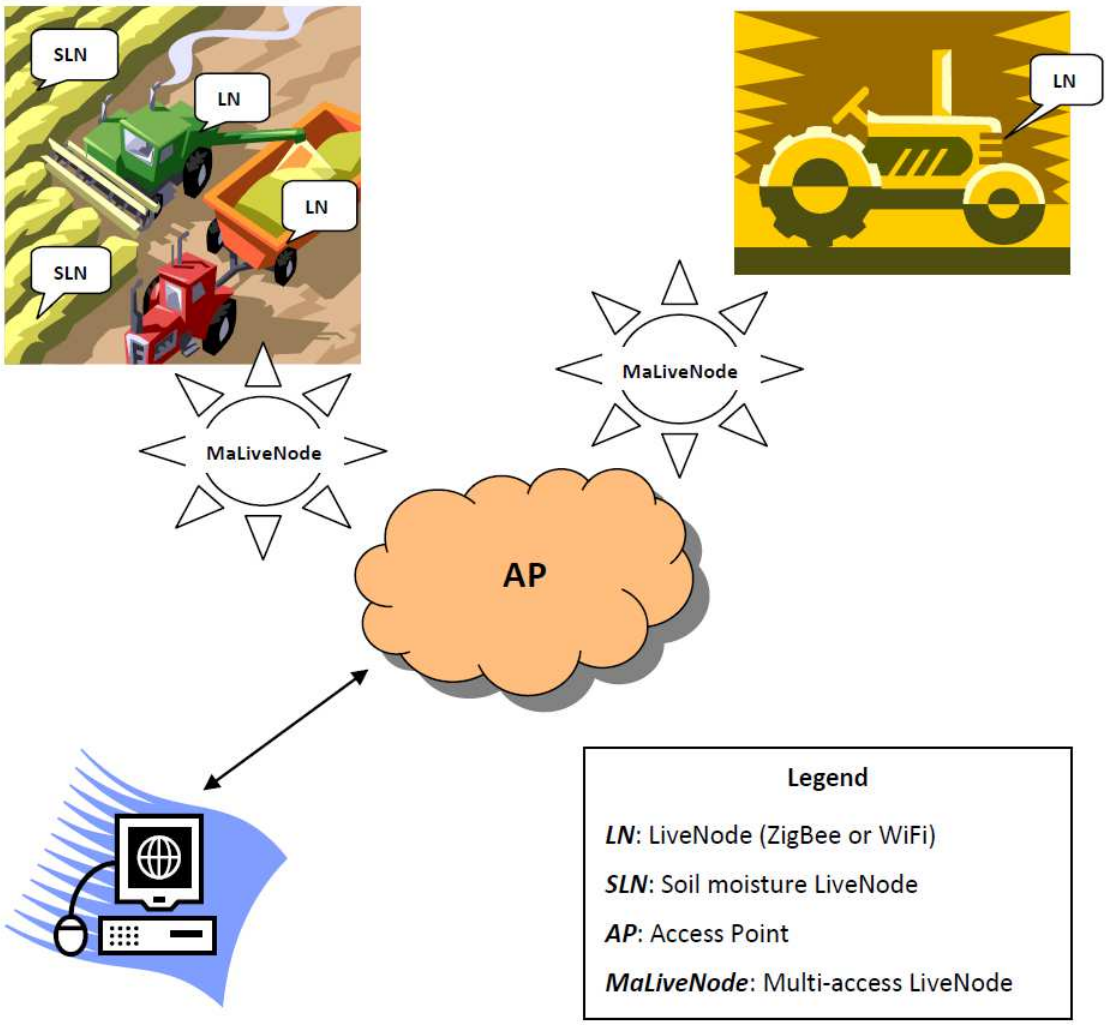


Figure 5.9: LivePlatform dedicated to precision agriculture

The *LivePlatform* is conceptually divided as four parts:

- Application layers (Section 3.2.2)
- Middleware: Distributed *IN/OUT primitive* based on LINDA concept (Section 3.3)
- Embedded software
  - *HEROS* (Hybrid Event-driven and Real-time multitasking Operating System) Section 3.3
  - *CIVIC* (Communication Inter Véhicule Intelligente et Coopérative) Section 3.2
- Embedded hardware:
  - *MaLiveNode* (Multi-access LiveNode) in Section 3.4
  - *SLN* (Soil Moisture LiveNode)
  - *LN* (LiveNode with WiFi or ZigBee) in Section 3.4

All these parts except the SLN have been introduced. The SLN is a LiveNode connection with a soil moisture sensor as shown in Figure 5.10.

Figure 5.10: Soil Moisture LiveNode

Chapter 6

Conclusions and Ongoing Works

The thesis has presented the state of art of routing technique researches on dynamic networks in Chapter 2. The routing techniques are classified into three broad categories: *topological*, *hierarchical* and *geographic* routing protocols. The geographic routing protocols are the ones more commonly adapted in the dynamic networks. But as for highly dynamic networks, none of these techniques can be considered to be exactly suitable for an embedded communication system. It is still an open issue for the current researches in routing techniques. The new researches about Geocast and DTN-based protocols could be more suitable for highly dynamic networks, and our ongoing routing solutions are more similar to the first one.

Embedded communication systems normally contain two basic software components: a protocol stack to manage network communications and an operating system to interface with hardware and schedule tasks or events. Based on such structure, this thesis presents a new low cost and low memory footprint design and its implementation for embedded IVC applications with CIVIC as protocol stack, and HEROS as embedded OS. The design, algorithms and implementation for both software components are given in the thesis.

Under our designs, CIVIC adopts the DANKAB mechanisms to provide a resource-awareness and rapid convergence routing algorithm, while HEROS proposes an Etask/Thread modular architecture and adopts a tuple-based IN/OUT primitive communication mechanism to provide both event-driven and real-time multitasking operation modes. At present, the CIVIC protocol has been ported on HEROS to perform real-world applications. The results from implementation and experiment show that this embedded system has a small resource consumption (about 43 KB) and is adaptable to different applications. Moreover, thanks to a low message sending delay (about 26 ms when sending 100 B) and a good reliability (shown in the fog experiments), the present design may be used to implement low cost embedded collision avoidance device by combining GPS receiver and IVC data.

Currently, the communication embedded system is still under development and evaluation, the result of network performances is not so perfect, but it shows the adaptability of this system to high-mobility scenarios like VANETs. The ongoing works of CIVIC are to continue reducing memory consumption and continue improving its communication QoS. For HEROS, its programming models are not completed, thus a hardware abstract layer or interface will be provided in future.

My works were mainly in the CIVIC protocol, thus the theoretical evaluations, simulations and real-world experiments are all given in details. The theoretical evaluations provide

the isolated formulas for computing potential network delay and loss rate, and the LRPC experiments prove the correction of the formulas, thus isolated formulas can be used in our network related projects in future. The LRPC experiments also prove that our embedded communication system can work well in different foggy environments. However, by the limitation of experiment conditions and available duration, the available samples in LRPC experiments are not enough for completing the accurate calculations. The ongoing works for this part will be the experiments in the rainy environments. In the end of the chapter, the results from Shawn simulation are presented, but it is also one of the ongoing works. The implementation methods and simulation results are not final.

Then, the thesis present two application domains: the VANET (IVC) in Chapter 4 and the WSN in Chapter 5. The implementations and real-world experiments are detailed in the first one, and the MATLAB simulation results are in the second one. The real-world experiment results prove our designs in the aspects of message delay, system latency and memory consumption. The ongoing works are to test the embedded communication system in a higher dynamic network with more sensor nodes.

The current shortage of our embedded communication system is from the choice of the beaconless IEEE 802.15.4 standard. Although the 2.4 GHz frequency in the standard works well in the fog experiments, the CSMA/CA mechanism in the beaconless mode performs badly with the increasing number of networking nodes. Note that, to use beacon-based 802.15.4 standard is not practical for highly dynamic networks like VANETs.

Because the XBee Pro hardware model used by LiveNode does not provide the low-level access, it is difficult to modify the 802.15.4 standard in our real-world experiments. An ongoing work related to the improvement of network standard will be done in the Shawn simulator.

Bibliography

[1] T. Clausen and P. Jacquet, "Optimized link state routing protocol (OLSR)," RFC 3626, Internet Engineering Task Force, October 2003.

[2] R. Ogier, F. Templin, and M. Lewis, "Topology dissemination based on reverse-path forwarding (TBRPF)," RFC 3684, Internet Engineering Task Force, 2004.

[3] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, pp. 234–244, 1994.

[4] C. E. Perkins, "Ad hoc on-demand distance vector routing protocol," internet-draft, IETF MANET Working Group, November 1997. Expiration: Mai 20, 1998.

[5] C. Perkins, E. Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing (retrieved 2010-06-18)," RFC 3561, 2003.

[6] D. B. Johnson, D. A. Maltz, and J. Broch, "DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks," in *In Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5*, pp. 139–172, Addison-Wesley, 2001.

[7] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3, pp. 1405–1413 vol.3, 1997.

[8] R. Dube, C. D. Rais, K.-Y. Wang, and S. K. Tripathi, "Signal stability-based adaptive routing (SSA) for ad hoc mobile networks," *Personal Communications, IEEE [see also IEEE Wireless Communications]*, vol. 4, no. 1, pp. 36–45, 1997.

[9] Z. J. Haas and M. R. Pearlman, "The performance of query control schemes for the zone routing protocol," *IEEE/ACM Trans. Netw.*, vol. 9, no. 4, pp. 427–438, 2001.

[10] C.-K. Toh, "Long-lived ad hoc routing based on the concept of associativity," internet-draft, IETF MANET Working Group, March 1999. Expired.

[11] S. Agarwal, A. Ahuja, J. P. Singh, and R. Shorey, "Route-lifetime assessment based routing (rabr) protocol for mobile ad-hoc networks," in *IEEE International Conference on Communications*, vol. 3, pp. 1697–1701 vol.3, 2000.

[12] M. S. Corson and A. Ephremides, "A distributed routing algorithm for mobile wireless networks," *Wireless Networks*, vol. 1, pp. 61–81, 1995. 10.1007/BF01196259.

[13] S. Guo, O. Yang, and Y. Shu, "Improving source routing reliability in mobile ad hoc networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, pp. 362–373, 2005.

[14] J. Wu, K.-M. Hou, X. Diao, and J.-J. Li, "Clustering and fuzzy position based routing in wireless sensor network for smart environment," tech. rep., Laboratoire LIMOS UMR 6158 CNRS, Université Blaise Pascal Clermont-Ferrand II, France, 2010.

[15] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *Wireless Communications, IEEE Transactions on*, vol. 1, no. 4, pp. 660–670, 2002.

[16] A. Manjeshwar and D. P. Agrawal, "TEEN: a routing protocol for enhanced efficiency in wireless sensor networks," in *Parallel and Distributed Processing Symposium., Proceedings 15th International*, pp. 2009–2015, 2001.

[17] Q. Fang, F. Zhao, and L. Guibas, "Lightweight sensing and communication protocols for target enumeration and aggregation," in *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '03, (New York, NY, USA), pp. 165–176, ACM, 2003.

[18] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A two-tier data dissemination model for large-scale wireless sensor networks," in *Proceedings of the 8th annual international conference on Mobile computing and networking*, MobiCom '02, (New York, NY, USA), pp. 148–159, ACM, 2002.

[19] X. X. Diao, M. Kara, J.-J. Li, K. M. Hou, H. Zhou, and A. Jacquot, "Cooperative inter-vehicle communication protocol with low cost differential gps," *Journal of Networks*, vol. 4, no. 6, pp. 445–457, 2009.

[20] A. Boukerche, H. A. B. F. Oliveira, E. F. Nakamura, and A. A. F. Loureiro, "Vehicular ad hoc networks: A new challenge for localization-based systems," *Comput. Commun.*, vol. 31, pp. 2838–2849, July 2008.

[21] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward, "A distance routing effect algorithm for mobility (DREAM)," in *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '98, (New York, NY, USA), pp. 76–84, ACM, 1998.

[22] Y.-B. Ko and N. H. Vaidya, "Location-aided routing (lar) in mobile ad hoc networks," *Wireless Networks*, vol. 6, no. 4, pp. 307–321, 2000.

[23] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris, "A scalable location service for geographic ad hoc routing," in *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, (New York, NY, USA), pp. 120–130, ACM, 2000.

[24] I. Stojmenovic, "Home agent based location update and destination search schemes in ad hoc wireless networks," Tech. Rep. TR-99-10, SITE, University of Ottawa, September 1999.

[25] 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 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, (New York, NY, USA), pp. 78–87, ACM, 2002.

[26] D. Liu, X. Jia, and I. Stojmenović, "Quorum and connected dominating sets based location service in wireless ad hoc, sensor and actuator networks," *Comput. Commun.*, vol. 30, pp. 3627–3643, December 2007.

[27] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, (New York, NY, USA), pp. 654–663, ACM, 1997.

[28] P. F. Tsuchiya, "The landmark hierarchy: a new hierarchy for routing in very large networks," *SIGCOMM Comput. Commun. Rev.*, vol. 18, pp. 35–42, August 1988.

[29] S. M. Das, H. Pucha, and Y. C. Hu, "Performance comparison of scalable location services for geographic ad hoc routing," vol. 2, pp. 1228–1239 vol. 2, 2005.

[30] R. Friedman and G. Kliot, "Location Services in Wireless Ad Hoc and Hybrid Networks: A Survey," Tech. Rep. CS-2006-10, Department of Computer Science Technion, Israel, October 2006.

[31] H. Sui and J. R. Zeidler, "Optimal Transmission Ranges for Randomly Distributed Packet Radio Terminals," *IEEE Transactions on Communications*, vol. 32, no. 3, pp. 246–257, 1984.

[32] T.-C. Hou and V. Li, "Transmission Range Control in Multihop Packet Radio Networks," *IEEE Transactions on Communications*, vol. 34, no. 1, pp. 38–44, 1986.

[33] G. G. Finn, "Routing and Addressing Problems in Large Metropolitan-Scale Internetworks," Tech. Rep. ISI/RR-87-180, University Of Southern California, Information Sciences Institute, March 1987.

[34] E. Kranakis, H. Singh, and J. Urrutia, "Compass Routing on Geometric Networks," in *11 th Canadian Conference on Computational Geometry*, (Vancouver), pp. 51–54, August 1999.

[35] I. Stojmenovic and X. Lin, "Loop-free hybrid single-path/flooding routing algorithms with guaranteed delivery for wireless networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1023–1032, 2001.

[36] M. Heissenbüttel, T. Braun, T. Bernoulli, and M. Wälchli, "Blr: beacon-less routing algorithm for mobile ad hoc networks," *Computer Communications*, vol. 27, no. 11, pp. 1076–1086, 2004.

[37] B. M. Blum, T. He, S. Son, and J. A. Stankovic, "Igf: A state-free robust communication protocol for wireless sensor networks," tech. rep., Computer Science Department, University of Virginia, 2003.

[38] M. Zorzi and R. R. Rao, "Geographic random forwarding (GeRaF) for ad hoc and sensor networks: multihop performance," *Mobile Computing, IEEE Transactions on*, vol. 2, no. 4, pp. 337–348, 2003.

[39] D. Chen, J. Deng, and P. K. Varshney, "Selection of a Forwarding Area for Contention-Based Geographic Forwarding in Wireless Multi-Hop Networks," *IEEE Transactions on Vehicular Technology*, vol. 56, pp. 3111–3122, September 2007.

[40] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker, "Geographic routing made practical," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, (Berkeley, CA, USA), pp. 217–230, USENIX Association, 2005.

[41] K. C. Lee, P.-C. Cheng, and M. Gerla, "Geocross: A geographic routing protocol in the presence of loops in urban scenarios," *Ad Hoc Netw.*, vol. 8, pp. 474–488, July 2010.

[42] R. K. Gabriel and R. R. Sokal, "A New Statistical Approach to Geographic Variation Analysis," *Systematic Zoology*, vol. 18, pp. 259–278, September 1969.

[43] G. T. Toussaint, "The relative neighbourhood graph of a finite planar set," *Pattern Recognition*, vol. 12, no. 4, pp. 261 – 268, 1980.

[44] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," in *Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, DIALM '99, (New York, NY, USA), pp. 48–55, ACM, 1999.

[45] B. Karp and H. T. Kung, "GPSR: greedy perimeter stateless routing for wireless networks," in *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, (New York, NY, USA), pp. 243–254, ACM, 2000.

[46] F. Kuhn, R. Wattenhofer, and A. Zollinger, "Asymptotically optimal geometric mobile ad-hoc routing," in *Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, DIALM '02, (New York, NY, USA), pp. 24–33, ACM, 2002.

[47] F. Kuhn, R. Wattenhofer, and A. Zollinger, "Worst-case optimal and average-case efficient geometric ad-hoc routing," in *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '03, (New York, NY, USA), pp. 267–278, ACM, 2003.

[48] B. Leong, B. Liskov, and R. Morris, "Geographic routing without planarization," in *NSDI'06: Proceedings of the 3rd conference on 3rd Symposium on Networked Systems Design & Implementation*, (Berkeley, CA, USA), p. 25, USENIX Association, 2006.

[49] *SPEED: a stateless protocol for real-time communication in sensor networks*, May 2003.

[50] J. Zhao and G. Cao, "VADD: Vehicle-Assisted Data Delivery in Vehicular Ad Hoc Networks," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pp. 1–12, IEEE, April 2006.

[51] *GeOpps: Geographical Opportunistic Routing for Vehicular Networks*, 2007.

[52] P.-C. Cheng, K. Lee, M. Gerla, and J. Härri, "GeoDTN+Nav: Geographic DTN Routing with Navigator Prediction for Urban Vehicular Environments," *Mobile Networks and Applications*.

[53] Y.-B. Ko and N. H. Vaidya, "Geotora: a protocol for geocasting in mobile ad hoc networks," in *Proceedings of the 2000 International Conference on Network Protocols*, ICNP '00, (Washington, DC, USA), pp. 240–, IEEE Computer Society, 2000.

[54] T. Imieliński and J. C. Navas, "Gps-based geographic addressing, routing, and resource discovery," *Commun. ACM*, vol. 42, pp. 86–92, April 1999.

[55] C. Maihöfer, "A survey of geocast routing protocols," *IEEE Communications Surveys and Tutorials*, vol. 6, no. 1-4, pp. 32–42, 2004.

[56] Y.-C. Tseng, S. yao Ni, and E. yu Shih, "Adaptive approaches to relieving broadcast storms in a wireless multihop mobile ad hoc network," *IEEE Transactions on Computers*, vol. 52, pp. 545–557, 2002.

[57] B. Williams and T. Camp, "Comparison of broadcasting techniques for mobile ad hoc networks," in *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, (New York, NY, USA), pp. 194–205, ACM, 2002.

[58] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic, "Impact of radio irregularity on wireless sensor networks," in *in MobiSYS '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pp. 125–138, ACM Press, 2004.

[59] L. Briesemeister, L. Schafers, G. Hommel, and D. Ag, "Disseminating messages among highly mobile hosts based on inter-vehicle communication," in *In IEEE Intelligent Vehicles Symposium*, pp. 522–527, 2000.

[60] A. Bachir and A. Benslimane, "A multicast protocol in ad hoc networks inter-vehicle geocast," in *Proc. 57th IEEE Semiannual Vehicular Technology Conference*, vol. 4, pp. 2456–2460, 2003.

[61] C. Maihöfer and R. Eberhardt, "Geocast in Vehicular Environments: Caching and Transmission Range Control for Improved Efficiency," in *IEEE Intelligent Vehicles Symposium (IV)*, pp. 951–956, 2004.

[62] G. Korkmaz, E. Ekici, F. Özgüner, and U. Özgüner, "Urban multi-hop broadcast protocol for inter-vehicle communication systems," in *VANET '04: Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*, (New York, NY, USA), pp. 76–85, ACM, 2004.

[63] C. Maihöfer, T. Leinmüller, and E. Schoch, "Abiding geocast: time–stable geocast for ad hoc networks," in *VANET '05: Proceedings of the 2nd ACM international workshop on Vehicular ad hoc networks*, (New York, NY, USA), pp. 20–29, ACM, 2005.

[64] *Locating nodes with EASE: last encounter routing in ad hoc networks through mobility diffusion*, vol. 3, 2003.

[65] M. Grossglauser and M. Vetterli, "Locating Mobile Nodes with EASE: Learning Efficient Routes from Encounter Histories Alone," *IEEE/ACM Transactions on Networking*, vol. 14, no. 3, pp. 457–469, 2006.

[66] U. Lee, J.-S. Park, E. Amir, and M. Gerla, "Fleanet: A virtual market place on vehicular networks," *Mobile and Ubiquitous Systems, Annual International Conference on*, vol. 0, pp. 1–8, 2006.

[67] J. A. Davis, A. H. Fagg, and B. N. Levine, "Wearable computers as packet transport mechanisms in highly-partitioned ad-hoc networks," in *Proceedings of the 5th IEEE International Symposium on Wearable Computers*, ISWC '01, (Washington, DC, USA), pp. 141–, IEEE Computer Society, 2001.

[68] H. Füßler, M. Mauve, H. Hartenstein, C. Lochert, D. Vollmer, D. Herrmann, and W. Franz, "Position-based routing in ad-hoc wireless networks," in *Inter-Vehicle-Communications Based on Ad Hoc Networking Principles —The FleetNet Project* (W. Franz, H. Hartenstein, and M. Mauve, eds.), pp. 117–143, Karlsruhe, Germany: Universitätsverlag Göttingen, Nov 2005.

[69] "Cartalk2000 website."

[70] P. Morsink, R. Hallouzi, I. Dagli, C. Cseh, L. Schafers, and M. Nelisse, "Cartalk2000: Development of a co-operative adas based on vehicle-to-vehicle communication," in *10th World Congress and Exhibition on Intelligent Transport Systems and Services*, November 2003.

[71] S. Tsugawa, "An introduction to demo 2000: The cooperative driving scenario," *IEEE Intelligent Systems*, vol. 15, pp. 78–79, 2000.

[72] Y. Shiraki, T. Ohyama, S. Nakabayashi, and K. Tokuda, "Development of an inter-vehicle communications system," *Special Edition on ITS*, vol. 68, pp. 11–13, September 2001.

[73] S. Tsugawa, "Issues and recent trends in vehicle safety communication systems," Tech. Rep. 29(1): 7-15, LATSS Research, 2005.

[74] P. Farradyne, "Vehicle infrastructure integration (vii) architecture and functional requirements," Tech. Rep. 1.1, ITS Joint Program Office, http://www.intellidriveusa.org/documents/27-VIIARC.PDF, 2005.

[75] Y. Ma, M. Chowdhury, A. Sadek, and M. Jeihani, "Real-time highway traffic condition assessment framework using vehicle-infrastructure integration (vii) with artificial intelligence (ai)," *Trans. Intell. Transport. Sys.*, vol. 10, pp. 615–627, December 2009.

[76] F. M. Aziz, "Implementation and analysis of wireless local area networks for high-mobility telematics," Master's thesis, Virginia Polytechnic Institute and State University, 2003.

[77] J. Hao, K. M. Hou, J.-J. Li, J.-P. Chanet, C. D. Vaulx, H. Zhou, and G. D. Sousa, "Capacity and packets delivery analysis of manet on road," in *ICWN*, pp. 516–522, 2005.

[78] M. Kara, *Wireless Sensor Networks: Study for developing a Low Cost Differential GPS receiver*. PhD thesis, Blaise Pascal University – Clermont-Ferrand II, 11 2009.

[79] GlobalSat Technology Corporation, *Product User Manual, Gps Engine Board, ET-301*.

[80] X. Diao, H. Zhou, K. M. Hou, and J.-J. Li, "An embedded system dedicated to inter-vehicle communication applications," *EURASIP Journal on Embedded Systems*, 2010.

[81] D. Gelernter, "Generative communication in linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, 1985.

[82] S. Ahuja, N. Carriero, and D. Gelernter, "Linda and friends," *IEEE Computer*, vol. 19, no. 8, pp. 26–34, 1986.

[83] D. Gelernter and N. Carriero, "Coordination languages and their significance," *Commun. ACM*, vol. 35, no. 2, p. 96, 1992.

[84] Sun Microsystems, *JavaSpaces Specification (V 1.0)*, 1999.

[85] M. Fontoura, T. Lehman, D. Nelson, and T. Truong, "Tspaces services suite: Automating the development and management of web services," in *In Proceedings of the 12th International World Wide Web Conference*, 2003.

[86] G. Wells, "Coordination languages: Back to the future with linda," in *Proceedings of WCAT'05*, pp. 87–98, 2005.

[87] A. L. Murphy, G. P. Picco, and G. catalin Roman, "Lime: A coordination model and middleware supporting mobility of hosts and agents," *ACM Transactions on Software Engineering and Methodology*, vol. 15, p. 2006, 2006.

[88] C. Scholliers, E. G. Boix, and W. D. Meuter, "Totam: Scoped tuples for the ambient," *ECEASST*, vol. 19, 2009.

[89] W. Leler, "Linda meets unix," *Computer*, vol. 23, no. 2, pp. 43–54, 1990.

[90] K. Raatikainen, "Operating system issues in wireless ad-hoc networks (keynote speech)," in *International Workshop on Wireless Ad-hoc Networks*, May 2005.

[91] H. Zhou, K. M. Hou, and C. D. Vaulx, "Sdream: A super-small distributed real-time microkernel dedicated to wireless sensors," *International Journal of Pervasive Computing and Communications*, vol. 12(4), pp. 398–410, 2007.

[92] P. Levis, S. Madden, J. Polastre, R. Szewczyk, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "Tinyos: An operating system for sensor networks," in *in Ambient Intelligence*, Springer Verlag, 2004.

[93] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, (Washington, DC, USA), pp. 455–462, IEEE Computer Society, 2004.

[94] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, "A dynamic operating system for sensor nodes," in *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, (New York, NY, USA), pp. 163–176, ACM, 2005.

[95] C. L. Liu and J. W. Layland, *Readings in hardware/software co-design*, ch. Scheduling algorithms for multiprogramming in a hard-real-time environment, pp. 179–194. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[96] I. Systems, *ARM IAR C,C++ compiler, Reference guide*, 2006.

[97] M. Moubarak and M. K. Watfa, "Embedded operating systems in wireless sensor networks," in *Guide to Wireless Sensor Networks* (S. C. Misra, I. Woungang, and S. Misra, eds.), Computer Communications and Networks, pp. 323–346, Springer London, 2009.

[98] W. Maurer, *The Scientist and Engineer's Guide to TinyOS Programming*. University of California-Berkeley, http://tinyos.org, 2004.

[99] K. Hou, G. De Sousa, H. Zhou, J. Chanet, M. Kara, A. Amamra, C. De Vaulx, J. Li, and A. Jacquot, "Livenode: Limos versatile embedded wireless sensor node," 2007.

[100] H.-Y. Zhou, G. de Sousa, J.-P. Chanet, K.-M. Hou, J.-J. Li, C. de Vaulx, and M. Kara, "An intelligent wireless bus-station system dedicated to disabled, wheelchair and blind passengers," in *Wireless, Mobile and Multimedia Networks, 2006 IET International Conference*, 2006.

[101] Atmel Corporation, *AT91SAM7S Series Preliminary*, 2007.

[102] Digi International Inc., *XBee/XBee-Pro RF Modules*, 2009.

[103] N. Samama, *Global Positioning: Technologies and Performance (Wiley Survival Guides in Engineering and Science)*. New York, NY, USA: Wiley-Interscience, 2008.

[104] Z. Sahinoglu, S. Gezici, and I. Güvenc, *Ultra-wideband Positioning Systems: Theoretical Limits, Ranging Algorithms, and Protocols*. Cambridge University Press, October 2008.

[105] I. C. Society, *IEEE Std 802.15.4-2003, Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Computer Society, New York, USA, October 2003.

[106] J. Tourrilhes, "Robust broadcast: Improving the reliability of broadcast transmissions on CSMA/CA," in *The Ninth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, vol. 3, (New York, NY, USA), pp. 1111–5, IEEE, 1998.

[107] Atmel Corporation, *Application Note 93 (Benchmarking with ARMulator)*, March 2002.

[108] Atmel Corporation, *ARM7TDMI Data Sheet (ARM DDI 0029E)*, August 1995.

[109] M. U. Mahfuz and K. M. Ahmed, "A review of micro-nano-scale wireless sensor networks for environmental protection: Prospects and challenges," *Science and Technology of Advanced Materials*, vol. 6, no. 3-4, pp. 302 – 306, 2005. International Conference on Nanotechnology in Environmental Protection and Pollution.

[110] D. D. Coleman and D. A. Westcott, *CWNA: Certified Wireless Network Administrator, Study Guide*. Wiley Publishing, Inc., 2006.

[111] A. Kröller, D. Pfisterer, C. Buschmann, S. P. Fekete, and S. Fischer, "Shawn: A new approach to simulating wireless sensor networks," in *Proceedings of the Design, Analysis, and Simulation of Distributed Systems Symposium 2005 (DASD' 05)*, pp. 117–124, Apr. 2005.

[112] S. P. Fekete, A. Kröller, S. Fischer, and D. Pfisterer, "Shawn: The fast, highly customizable sensor network simulator," in *Proceedings of the Fourth International Conference on Networked Sensing Systems (INSS 2007)*, June 2007.

[113] S. Mccanne, S. Floyd, and K. Fall, "ns2 (network simulator 2)." http://www-nrg.ee.lbl.gov/ns/.

[114] *Arm-elf-gcc Documents*. http://www.gnuarm.com/.

[115] R. S. Last, R. M. Stallman, and S. Contents, "Using the gnu compiler collection," *M.I.T. Artificial Intelligence Laboratory*, 2003.

[116] W. R. Heinzelman, A. Chandrakasan, H. Balakrishnan, W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*, (Washington, DC, USA), IEEE Computer Society, 2000.

[117] N. Vlajic and D. Xia, "Wireless sensor networks: To cluster or not to cluster?," in *WOWMOM '06: Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, (Washington, DC, USA), pp. 258–268, IEEE Computer Society, 2006.

[118] X. X. Diao, E. Lai, K. M. Hou, and H. Y. Zhou, "An auto-clustering algorithm for wireless sensor network management protocol," in *Conférence Internationale sur les NOuvelles TEchnologies de la REpartition (NOTERE'2007), Workshop on Wireless Sensor Networks*, (Marrakech-Morocco), pp. 17–22, June 2007.

[119] I. Fayberg, L. R. Gebuzda, T. Jacobson, and H.-L. Lu, "The development of the wireless intelligent network (win) and its relation to the international intelligent network standards," *Bell Labs Technical Journal*, pp. 57–80, 1997.

Appendix A

The format of CIVIC message

Fields	Size(Byte)		Data Types	Messages							
				(1) Hello		(2) Routing				(3) Application Data	
	Min	Max		HELLO_REQ	HELLO_RPY	ROUTE_REQ_SF	ROUTE_REQ_CIVIC	ROUTE_RPY_CIVIC	ROUTE_RPY_BY_PATH	DATA_SEND_BY_PATH	DATA_ACK_BY_PATH
START_MARK	-	1	*BYTE_T*	*	*	*	*	*	*	*	*
MSG_TYPE	-	1	*TYPE_T*	*	*	*	*	*	*	*	*
MSG_STATUS	-	1	*STATUS_T*	*	*	*	*	*	*	*	*
SND_ID	-	1	*ID_T*	*	*	*	*	*	*	*	*
DST_ID	-	1	*ID_T*			*	*	*	*	*	*
FWD_ID	-	1	*ID_T*				*	*			
SRC_ID	-	1	*ID_T*			*	*	*	*	*	*
DST_IDS	1	15	*IDS_T*		*						
PATH_IDS	1	7	*IDS_T*						*	*	*
PRE_IDS	1	7	*IDS_T*			*	*	*			
SND_LOC	-	19	*LOC_T*	*	*						
DST_LOC	-	19	*LOC_T*				*	*			
SRC_LOC	-	19	*LOC_T*			*(optional)		*(optional)	*(optional)		
GPS_RMC	-	11	*RMC_T*		*						
GPS_TIME	-	6	*TIME_T*	*(optional)							
APP_HEAD	-	3	*APP_HEAD_T*							*	*
APP_DATA	1	50	*APP_DATA_T*							*	
SRC_SN	-	1	*SEQ_T*			*					
SND_SN	-	1	*SEQ_T*	*	*	*	*	*	*	*	*
CRC	-	2	*BYTE_T*	*	*	*	*	*	*	*	*
END_MARK	-	1	*BYTE_T*	*	*	*	*	*	*	*	*
Network Type	**-**	**-**	**-**	**Broadcast**	**Multicast**	**Broadcast**	**Unicast**	**Unicast**	**Unicast**	**Unicast**	**Unicast**
Total Size (Min)	**-**	**-**	**-**	**27**	**39**	**12**	**31**	**31**	**11**	**15**	**14**
Total Size (Max)	**-**	**-**	**-**	**33**	**53**	**37**	**37**	**56**	**36**	**70**	**20**

Appendix B

Layer-based Modules

Module and header files	Usages
include/debug_io.h	The global defines for debug information
include/glb_conf.h	Global configurations used by all modules in the project
include/glb_type.h	Global data type definitions
include/glb_func.h	The head file of global functions
include/glb_func.c	Global functions
spcl/gcc/include/gcc_func.h	System configurations and functions related to GCC only
spcl/gcc/include/glb_opt.h	Turn on/off global options (GCC, HEROS and Debug Info.)
spcl/gcc/include/math.h	Definitions for the math floating point package
spcl/gcc/L1_board/bd_dcc.c	Read and write debug information by ARM DCC for GCC+BDI
spcl/gcc/L1_board/cstartup.s	Generic CStartup for GCC (IS_REMAPPED==1, remap vectors)
spcl/gcc/L5_ctl/heros/shell.s	ASM instructions of HEROS for GCC
spcl/gcc/miss/ieee754-df.S	Double-precision floating point support for ARM
spcl/gcc/miss/syscalls.c	System Calls for the newlib
spcl/iar/include/iar_func.h	System configurations and functions related to IAR C only
spcl/iar/include/glb_opt.h	Turn on/off global options (GCC, HEROS and Debug Info.)
spcl/iar/L1_board/... ...at91sam7s256_inc.h	Hardware register definition
spcl/iar/L1_board/Cstartup.s79	Generic CStartup for IAR
spcl/iar/L5_ctl/heros/shell.s79	ASM instructions of HEROS for IAR
include/ext/l1_bd_led.h	The external functions and variables of bd_led.c
include/ext/l1_bd_pit.h	The external functions and variables of bd_pit.c
include/ext/l1_bd_us0.h	The external functions and variables of bd_us0.c
include/ext/l1_bd_us1.h	The external functions and variables of bd_us1.c
include/ext/l2_mid_us0.h	The external head file for modules us0_to_gps.c, gps_to_civic.c, gps_func.c, and gps_lab_test.c
include/ext/l2_mid_us1.h	The external head file for modules us1_to_civic.c and civic_to_us1.c
include/ext/l2_us0_fm.h	The message format of l2_mid_us0.h and modules of "L2_mid/us0"
include/ext/l2_us1_fm.h	The message format of l2_mid_us1.h and modules of "L2_mid/us1"
include/ext/l3_civic.h	The external head file for modules in "L3/civic"
include/ext/l3_civic_fm.h	The message format of l3_civic.h and modules of "L3/civic"
include/ext/l4_app.h	The external head file for module app.c and app_task.c

Module and header files	Usages
L1_board/board.h	LiveNode board features definition file (only used in L1_board layer)
L1_board/bd_led.c	LED management on LiveNode
L1_board/bd_pit.c	PIT initialization and interrupt management
L1_board/bd_us0.c	USART0 (GPS) initialization and interrupt management
L1_board/bd_us1.c	USART1 (XBEE) initialization and interrupt management
L1_board/cstartup_sam7.c	Low level initializations written in C for Tools
L2_mid/us0/mid_us0.h	The internal head file for modules in us0_to_gps.c, gps_to_civic.c, gps_func.c, and gps_lab_test.c
L2_mid/us0/gps_conv.c	The GPS functions relate to coordinate convention
L2_mid/us0/gps_lab_test.c	This module provides preset locations and times for in-lab tests only
L2_mid/us0/gps_to_civic.c	Provides the last valid location and time to CIVIC layer
L2_mid/us0/us0_to_gps.c	Translates the raw data from USART 0 by GPS format
L2_mid/us1/civic_to_us1.c	This module contain function for outputting CIVIC messages to USART 1
L2_mid/us1/func_crc.c	CRC related functions
L2_mid/us1/us1_to_civic.c	Translates the raw data from USART 1 to CIVIC messages
L3_civic/civic.h	The internal head file for modules in "L3_civic"
L3_civic/civic_core.c	The major functions of the CIVIC protocol
L3_civic/civic_core_in.c	This module processes incmoing CIVIC messages
L3_civic/civic_core_out.c	This module generates output messages in CIVIC format
L3_civic/civic_id.c	The functions related to CIVIC ID
L3_civic/civic_func_misc.c	The extended CIVIC functions
L3_civic/civic_func_exp.c	The functions used only in CIVIC experiments
L3_civic/table_dst_loc.c	The table is to save the locations of destination nodes
L3_civic/table_hello_rpy.c	The table is to save the IDs needed by hello reply messages
L3_civic/table_neighbor.c	The table is to save the IDs and locations of one-hop neighbour nodes
L3_civic/table_routing.c	The table is to save routing paths
L3_civic/table_sf.c	The table is to avoid the redundant resend by simple flooding routing requests
L4_app/app.c	Transportation Layer (Routing interface)
L4_app/app_task.c	Application task examples

Module and header files	Usages
L5_ctl/main.c	Starting point of program
L5_ctl/l5_ctl.h	The collection of lower layers interfaces opened to "L5_ctl"
L5_ctl/rt_cntr.h	The external head file for modules kern_rt_cntr.c or loop_rt_cntr.c
L5_ctl/task_set.h	The external head file for module task_set.c opened to "L5_ctl"
L5_ctl/task_set.c	Define actions of tasks
L5_ctl/manage/manage.h	The external head file for module manage.c opened to "L5_ctl"
L5_ctl/manage/manage.c	The management panel
L5_ctl/heros/kern_ex.h	The external head file for modules in "L5_ctl/heros" opened to "L5_ctl"
L5_ctl/heros/kern_variable.h	The definitions of data structures of HEROS
L5_ctl/heros/kern_kernel.c	The system functions of HEROS managing tuple in/out, events, and threads
L5_ctl/heros/kern_rt_cntr.c	The PIT timer will active real-time task of HEROS by this module
L5_ctl/heros/kern_software.c	The system initialization and configuration of HEROS
L5_ctl/non_os/loop_ex.h	The external head file for modules in "L5_ctl/non_os" opened to "L5_ctl"
L5_ctl/non_os/loop.c	If IS_HEROS_ON==0, the looping tasks in this module is run
L5_ctl/non_os/loop_rt_cntr.c	If IS_HEROS_ON==0, the PIT timer will active real-time task by this module