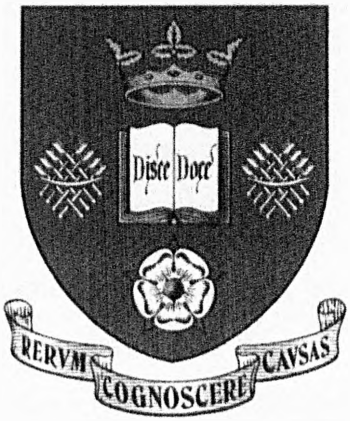
DEPARTMENT OF ELECTRONIC & ELECTRICAL ENGINEERING

UNIVERSITY OF SHEFFIELD



Inter-Vehicular Communication using wireless Ad-hoc Networks

By
Mr. Raúl Aquino Santos

SUPERVISORS

Dr. Robert M. Edwards
Dr. Luke Seed

TUTOR
Dr. Peter I. Rockett

Date
05/11/2004

ABSTRACT

This thesis proposes a new routing algorithm to allow communication in highly mobile, wireless ad-hoc networks, which in nature are wireless and infrastructureless. In motorway environments, the topology of the network changes frequently and unpredictable due to the mobility of the nodes.

We investigate a new reactive routing algorithm based in location information in the context of inter-vehicular communication. In such a scenario, the originator of the communication does not know the position of its communication partner in advance. Rapid topology changes and scarce bandwidth prevent the nodes from exchanging positions regularly throughout the network. Therefore, we focus on reactive algorithms and explore several mechanisms limiting the flooding of discoveries location packets.

The originator of a message uses scoped and controlled flooding to reach the destination. The receivers of the flooded message use their knowledge of the local environment to decide whether they can reach the intended destination of the message or retransmit the message to their neighbours.

To evaluate our communication algorithm, we first validate it in a small scale network with the results of a test bed. Then for large scale networks, our protocol is compared with the models of two prominent reactive routing algorithms: Ad-Hoc On-Demand Distance Vector (AODV) and Dynamic Source Routing (DSR) on a multi-lane rectangular and circular dual carriageway representative of city and motorway driving. Finally, our algorithm is analysed on a multi-lane circular carriageway representative of a six lane motorway driving with one location-based routing algorithm: Greedy Perimeter Stateless Routing (GPSR).

The mobility of the vehicles on a Motorway using a Microscopic traffic model developed in OPNET has been used to evaluate the performance of each protocol in terms of: Route Discovery Time (RDT), End to End Delay (EED), Routing Overhead (RO), Overhead (O), Routing Load (RL) and Delivery Ratio (DR).

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Robert Edwards, for his valuable support and guidance through the realisation of my doctoral studies. He took a chance by accepting me as his student, and patiently guided me throughout my research career at the University of Sheffield. He always found time for me whenever I needed his input. Also, I wish to thank my second supervisor, Dr. Luke Seed, for his valuable comments, and feedback, and my tutor, Dr. Peter Rockett, for constantly providing his guidance and support.

I want to thank Arthur Edwards for his continuous help in reviewing my thesis writing.

I also wish to express my sincerest gratitude to department colleges, as well as a special "thank you" to Salvador Villarreal for his help in the final process of my thesis.

Last but not least, I would like to thank my family, especially my parents for teaching me to believe in myself and follow my dreams, and my two sisters and brother for their love and support.

To my wife Tania who gave me two daughters: Iritzi and Dafne my greatest treasures.

LIST OF PUBLICATIONS

Papers in conferences:

R. A. Santos, R. M. Edwards, N. L. Seed. Using the Cluster-Based Location Routing (CBLR) Algorithm for Exchanging Information on a Motorway. Proceeding of the Fourth IEEE Conference on Mobile and Wireless Communication Networks (MWCN). September 2002. Stockholm, Sweden.

R. A. Santos, R. M. Edwards, N. L. Seed. First steps towards inter-vehicular traffic of data on Motorways using ad-hoc networks over IEEE 802.11b. Postgraduate Research Conference in Electronics, Photonics, Communications and Software (PREP2003). April 2003. Exeter, U.K.

R. A. Santos, R. M. Edwards, N. L. Seed. Cluster-Based Location Routing algorithm and its application on Motorways. IASTED International Conference, Computer Science and Technology (CST 2003). May 2003. Cancun, Mexico.

R. A. Santos, R. M. Edwards, N. L. Seed. Inter-Vehicular Data Exchange between fast moving road traffic using an Ad-Hoc Cluster-Based Location Routing algorithm and 802.11b Direct Sequence Spread Spectrum Radio. PostGraduate Networking Conference (PGNET 2003). June 2003. Liverpool, U.K.

R. A. Santos, R. M. Edwards, N. L. Seed. Using a Short-Term Predictive Algorithm to Improve the Communication's Speed in Fast Mobile Ad-Hoc Networks. London Communications Symposium (LCS 2003). September 2003.London, U.K.

R. A. Santos, R. M. Edwards, N. L. Seed. Supporting Inter-Vehicular and Vehicle-Roadside Communications over a Cluster-Based Wireless Ad-Hoc Routing Algorithm. Winter International Symposium on Information and Communication Technologies (WISICT 2004). January 2004. Cancún, México.

R. A. Santos, R. M. Edwards, A. Edwards. A Novell Cluster-Based Location Routing Algorithm for Inter-Vehicular Communication. The 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2004). September 5 – 8, 2004, Barcelona, Spain.

R. A. Santos, R. M. Edwards, and A. Edwards. Cluster-Based Location Routing Algorithm for Vehicle to Vehicle Communication. IEEE Radio &Wireless Conference: Latest in Wireless & RF Technology (RAWCON 2004). September 19 – 22, 2004, Atlanta, GA USA.

R. A. Santos, R. M. Edwards, and A. Edwards. Cluster-Based Location Routing Algorithm for Inter-Vehicle Communication. IEEE Vehicular Technology Conference (VTC-2004 Fall). September 26 – 29, Los Angeles, CA USA.

R. A. Santos, R. M. Edwards, N. L. Seed and A. Edwards. A Location-Based Routing Algorithm for Vehicle to Vehicle Communication. The 13th IEEE International Conference on Computer Communication and Networks (ICCCN 2004). October 11-15, Chicago IL, USA.

Paper in Journals:

R. A. Santos, R. M. Edwards, N. L. Seed and A. Edwards. A Reactive Location Routing Algorithm with Cluster-Based Flooding for Inter-Vehicle Communication. Submitted to IEEE Transaction on Mobile Computing (TMC).

R. A. Santos, R. M. Edwards, N. L. Seed and A. Edwards. Exchanging information in Vehicular Ad-hoc Networks (VANET). Submitted to the International Journal of Ad Hoc and Ubiquitous Computing.

R. A. Santos, R. M. Edwards and A. Edwards. Inter-vehicular Communication using Wireless Ad-hoc Networks. Submitted to IEEE Transactions on Vehicular Technology.

LIST OF CONTENTS

Appendices

LIST OF ABBREVIATIONS

ABR	Associativity-Based Routing Protocol
AICC	Autonomous Intelligent Cruise Control
AODV	Ad-Hoc On-Demand Distance Vector
AP	Access Point
BSA	Basic Service Area
BSS	Basic Service Set
CCK	Complementary Code Keying
CDC	Centralized Distance Control
CF	Contention Free
CFP	Contention Free Period
CP	Contention Period
CPC	Centralized Platoon Control
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Clear to Send
DARPA	Defense Advanced Research Project Agency
DBPSK	Differential Binary Phase Shift Keying
DQPSK	Differential Quadrature Phase Shift Keying
DCF	Distribute Coordination Function
DLS	DREAM Location Service
DREAM	Distance Routing Effect Algorithm for Mobility
DSSS	Direct Sequence Spread Spectrum
DSR	Dynamic Source Routing
EED	End-to-End Delay
FCC	Federal Communication Commission
FHSS	Frequency Hopping Spread Spectrum
FSP	Free Space Propagation
GFSK	Gaussian Frequency Shift Keying
GLS	Grid Location Service
GPS	Global Position System
GPSR	Greedy Perimeter Stateless Routing
HR/DSSS	High Rate Direct Sequence Spread Spectrum
IARP	Intra-zone Routing Protocol
IERP	Inter-zone Routing Protocol
ILG	Intelligent Lateral Control
ITS	Intelligent Transportation Systems
IVC	Inter-Vehicle Communication
IVHS	Intelligent Vehicle Highway System
ISA	Intelligent Speed Limit Device
ISM	Industrial, Scientific and Medical
LAN	Local Area Networks
LAR	Location-Aided Routing
LCA	Link Cluster Architecture
LG	Lane Guidance
LORA-CBF	Location Routing Algorithm with Cluster-Based Flooding

LREQ	Location Request Packets
LREP	Location Reply Packet
LPR	Low-cost Packet Radio
LSA	Link State Advertisement
LSU	Link State Update
MAC	Medium Access Control
MANET	Mobile Ad-Hoc Networks
MFR	Most Forward within Radius
MPDU	MAC Protocol Data Unit
MPR	Multipoint Relay
NAV	Network Allocation Vector
NIST	National Institute of Standard and Technology
NTDR	Near-Term Digital Radio
OFDM	Orthogonal Frequency Division Multiplexing
OLSR	Optimized Link State Routing
OSR	Optimal Spine Routing
PC	Point Coordinator
PCF	Point Coordination Function
PDU	Protocol Data Unit
PLCP	Physical Layer Convergence Procedure
PRNET	Packet Radio Network
PSDU	PLCP Service Data Unit
PSR	Partial-knowledge Spine Routing
RD	Route Discovery
RF	Radio Frequency
RLS	Reactive Location Service
RREQ	Route Request Packet
RREP	Route Reply Packet
RTS	Request to Send
SLS	Simple Location Service
SOM	System Operating Margin
SSL	Static Speed Limit
SURAN	Survivable Radio Network
TBRPF	Topology Dissemination Based on Reverse Path Forwarding
TDMA	Time Division Multiple Access
TND	TBRPF Neighbour Discovery
VANET	Vehicular Ad-hoc Networks
WLANs	Wireless Local Area Networks
ZRP	Zone Routing Protocol

List of Figures

LIST OF GRAPHS

LIST OF TABLES

Chapter 1: INTRODUCTION

1.1 INTRODUCTION TO WIRELESS LOCAL AREA NETWORKS (WLANS)

The requirements of data communication, beyond the physical link, has resulted in the necessity of wireless networks, which have been fuelled by fabrication improvements of digital and RF circuits, new large-scale circuit integration, and other miniaturization technologies that make portable radio equipment smaller, cheaper, and more reliable.

Wireless Local Area Networks (WLANs) represent flexible data communications systems that can be implemented as an extension to, or as an alternative for, a wired LAN. Using a form of electromagnetic radiation as the network medium, most commonly in the form of radio waves, wireless LANs transmit and receive data over air, minimising the need for wired connections (cables). Thus, wireless LANs combine data connectivity with user mobility. By combining mobile devices with wireless communications technologies, the vision of being connected at anytime and anywhere will soon become a reality.

Whereas today's expensive wireless infrastructure depends on centrally deployed hub and one hop stations, mobile ad hoc networks consist of devices that are autonomously self-organizing in networks. In ad-hoc networks, the devices themselves comprise the network. This advantage permits seamless communication, at low cost, in a self-organized fashion and with easy deployment. The large degree of freedom and the self-organizing capabilities make possible ad-hoc networks completely different from any other networking solution. For the first time, users have the opportunity to create their own networks, which can be deployed easily and inexpensively.

Ad-hoc networks are a key step in the evolution of wireless networks. They inherits the traditional problems of wireless and mobile communications such as bandwidth optimization, power control and transmission quality enhancements. There are presently two WLAN standards that define protocol architectures and network topologies: Bluetooth, and IEEE 802.11b [1].

1.2 BLUETOOTH TECHNOLOGY

The main characteristics for Bluetooth technology are described below:

- The Bluetooth system operates in the 2.4 GHz, industrial, scientific, and medical (ISM) band.
- This technology is optimized for short-range communications, low power consumption and low cost.
- The higher layer reuses transport and applications protocols already developed for similar domains, such as those used with infrared wireless communications.

Bluetooth specifies a 10m-radio range and supports up to 7 devices per piconet (Figure 1.1) **[2][3]**.

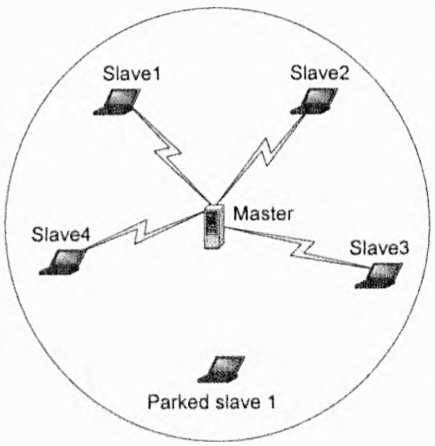


Figure 1.1: Basic topology of a simple piconet.

As noted above, a piconet can include up to seven active slaves and many more parked slaves. In active mode, a slave essentially always listens for transmissions from the master. Active slaves receive packets that enable them to remain synchronized with the master and inform it when they can transmit packets back to the master. In parked mode, a slave maintains synchronization with the master, but is no longer considered active. Since there can be only seven active slaves in a piconet at one time, this is due to three–bit node addressing inside piconet. The parked mode allows the master to orchestrate communication within a piconet of more than seven devices by exchanging active and parked slaves to maintain up to seven active connections while the remaining slaves in the piconet are parked. Figure 1.2 shows the general view of a piconet.

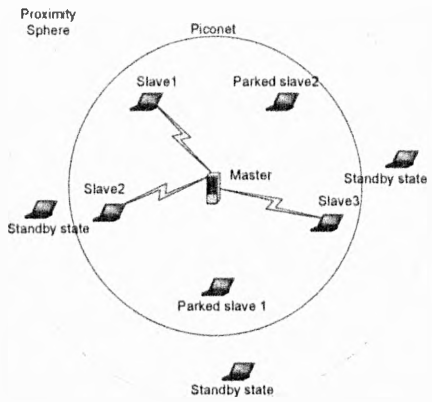


Figure 1.2: General Bluetooth piconet including active, parked slaves and standby devices.

As described and illustrated above, a device may be an active or parked participant in a piconet, or it may not be part of any piconet. In addition, it is possible for a device to take part in more than one piconet. When two or more piconets at least partially overlap in time and space, a scatternet is formed (Figure 1.3). All of the same principles of piconets also apply for scatternets; each piconet has a single master and a set of slaves, which may be active or parked. Each piconet has its own hopping pattern, which is determined by its master. A slave can participate in multiple piconets by, in turn, establishing connections with and synchronizing to different masters in proximity. In fact, a single device might act as a slave in one piconet, but assume the master role in another piconet. The scatternet topology shown in 2.3 illustrates the flexibility by which devices can maintain multiple connections, which is especially useful for mobile devices that frequently move into and out of proximity to other devices.

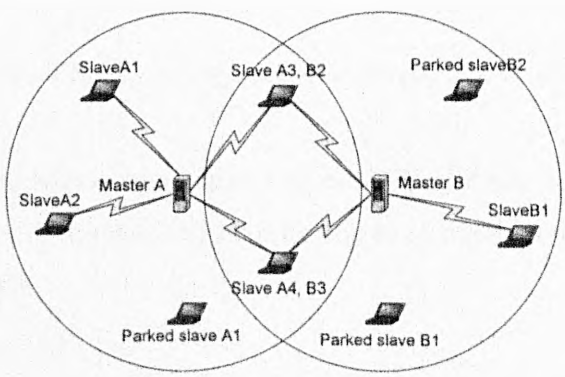


Figure 1.3: Scatternet topology.

1.3 IEEE 802.11 WLAN ARCHITECTURE

The IEEE 802.11 was the first international standard for WLANs **[4].** The basic service set (BSS) is the fundamental building block of the IEEE 802.11 architecture. A BSS is defined as a group of stations that are under the direct control of a single coordination function (e.g. Direct Coordination Function (DCF) or Point Coordination Function (PCF)), which is defined below.

The geographical area covered by the BSS is known as the basic service area (BSA), which is analogous to a cell in a cellular communication network. Conceptually, all stations in a BSS can communicate directly with all other stations in a BSS.

An ad-hoc network is a deliberate grouping of stations into a single BSS for the purposes of inter-networked communications, without the aid of an infrastructure network. Figure 1.4 provides an illustration of an infrastructure and independent BSS. The independent BSS is the formal name of an ad-hoc network in the IEEE 802.11 standard.

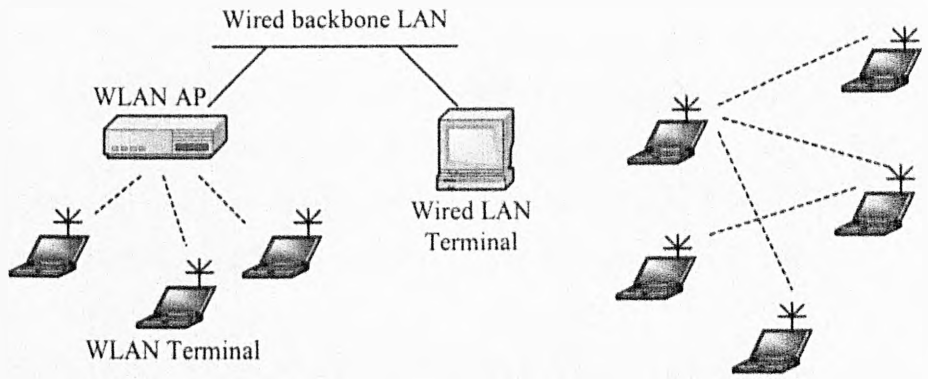


Figure 1-4: Basic WLAN topologies: infrastructure and ad-hoc.

In an ad-hoc network, any station can establish a direct communication session with any other station in the BSS, without having to channel all traffic through a centralized access point (AP).

1.3.1 Physical Layer

The IEEE specification calls for three different physical-layer implementations: Frequency Hopping Spread Spectrum (FHSS), Direct Sequence Spread Spectrum (DSSS), and Infrared. The FHSS utilizes the 2.4 GHz Industrial, Scientific, and Medical (ISM) band (e.g. 2.4- 2.4835 GHz). In the United States, a maximum of 79 channels are specified in the hopping set. The first channel has a centre frequency of 2.402 GHz, and all subsequent channels are spaced 1 MHz apart. The 1 MHz separation is mandated by the FCC for the 2.4 GHz ISM band. The channel separation corresponds to 1 Mb/s of instantaneous bandwidth. Three different hopping sequence sets are established with 26 hopping sequences per set. Different hopping sequences enable multiple BSSs to coexist in the same geographical area, which may become necessary to alleviate congestion and maximize the total throughput in a single BSS. The minimum hop rate permitted is 2.5 hops/seconds. The basic access rate of 1 Mb/s uses two-level Gaussian frequency shift keying (GFSK). The enhanced access rate of 2 Mb/s uses four-level GFSK (Figure 1.5). The DSSS also uses the 2.4 GHz ISM frequency band, where the 1 Mb/s basic rate is encoded using differential binary phase shift keying (DBPSK), and a 2 Mb/s enhanced rate uses differential quadrature phase shift keying (DQPSK). The spreading is done by dividing the available bandwidth into 11 sub-channels, each 11 MHz wide, and using an 11-chip Barker sequence to spread each data symbol. The maximum channel capacity is therefore (11 chips/symbol)/ (11 MHz) = 1 Mb/s if DBPSK is used.

In October 1997, the IEEE 802 Executive Committee approved two extensions for higher data rate transmission. The first extension, IEEE 802.11a, defines requirements for a PHY operating in the 5.0 GHz frequency and data rate transmission ranging from 6 Mbps to 54 Mbps. The second extension, IEEE 802.11b, defines a set of PHY specifications operating in the 2.4 GHz frequency band up to 11 Mbps. Both PHY are defined to operate with the existing MAC.

The IEEE 802.11a PHY is one of the physical layer extensions of IEEE 802.11 and is referred to as orthogonal frequency division multiplexing (OFDM) and the IEEE 802.11b is referred to as high rate direct sequence spread spectrum (HR/DSSS). The HR/DSSS PHY provides two functions. First, the HR/DSSS extends the PSDU data rates to 5.5 and 11 Mbps using an enhanced modulation technique, called

Complementary Code Keying (CCK). Secondly, the HR/DSSS PHY provides a rate shift mechanism, which allows 11 Mbps networks to fall back to 1 and 2 Mbps and interoperates with the legacy IEEE 802.11 standard. Finally the IEEE 802.11g was approved in June 2003 **[5]**. The IEEE 802.11g standard provides optional data rates transmission of up to 54 Mbps, and requires backward compatibility with 802.11b devices to protect the substantial investments in today's WLAN installations. The 802.11g standard includes mandatory and optional components. It specifies OFDM and CCK as the mandatory modulation schemes with 24 Mbps as the maximum mandatory data rates, but it also provides for optional higher data rates of 36, 48 and 54 Mbps.

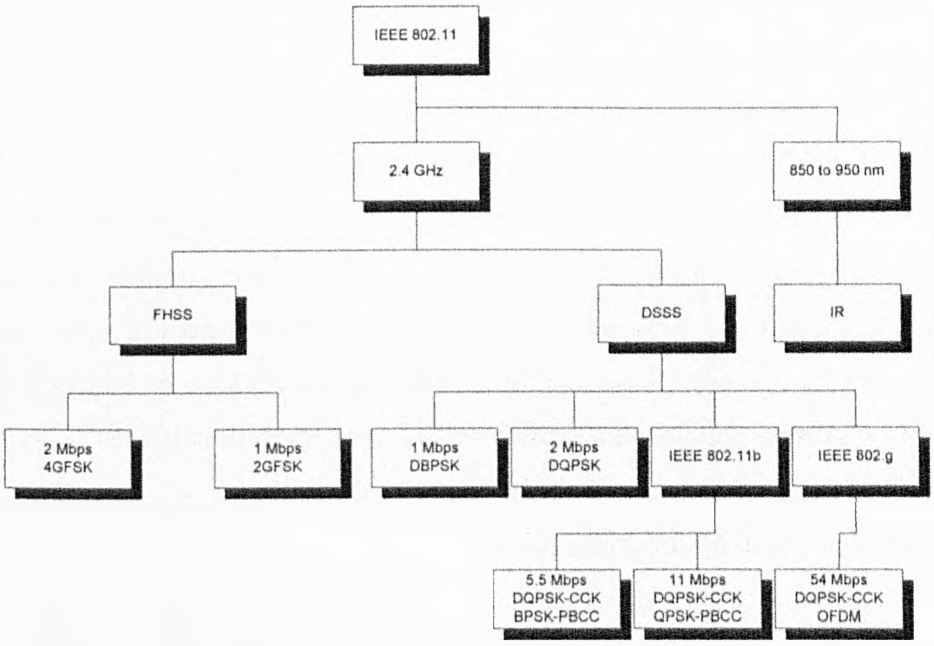


Figure 1.5: IEEE 802.11 architecture.

1.3.2 Medium Access Control Sub-layer

The MAC sub-layer is responsible for channel allocation procedures, protocol data unit (PDU) addressing, frame formatting, error checking, and fragmentation and reassembly.

The transmission medium can operate in the contention mode exclusively, requiring all stations to contend for access to the channel for each packet transmitted. The medium can also alternate between the contention mode, known as the contention

period (CP) under the Distribute Coordination Function (DCF), and a contention-free period (CFP) under the Point Coordination Function (PCF). During the CFP, medium usage is controlled (or mediated) by the AP, thereby eliminating the need for stations to contend for channel access.

The DCF is the fundamental access method used to support asynchronous data transfer on a best effort basis. The DCF operates exclusively in ad-hoc networks and is based on carrier sense multiple access with collision avoidance (CSMA/CA). In IEEE 802.11, carrier sensing is performed at both the air interface, referred to as physical carrier sensing and at the MAC sub-layer, referred to as virtual carrier sensing. Physical carrier sensing detects the presence of other IEEE 802.11 WLAN users by analyzing all detected packets and also detecting activity in the channel via relative signal strength from other sources.

A source station performs virtual carrier sensing by sending MPDU duration information in the header of request to send (RTS), clear to send (CTS), and data frames. The duration field indicates the amount of time (in microseconds) after the end of the present frame. The channel will then be utilized to complete the successful transmission of the data or management frame. Stations in the BSS use the information in the duration field to adjust their network allocation vector (NAV), which indicates the amount of time that must elapse until the current transmission session is complete and the channel can be sampled again for idle status. The channel is marked busy if either the physical or virtual carrier sensing mechanisms indicate the channel is busy.

On the other hand, the PCF is an optional capability, which is connection-oriented, and provides contention-free (CF) frame transfer. The PCF relies on the point coordinator (PC) to perform polling, enabling polled stations to transmit without contending for the channel. The function of the PC is performed by the AP within each BSS.

1.4 ROUTING ALGORITHMS FOR WIRED NETWORKS

The most popular dynamic routing algorithms in modern computer networks are: *distance vector and link state routing* **[6].**

Distance vector routing algorithms require each router to simply inform its neighbours of its routing table. These tables are updated by exchanging information with the

neighbours. For each neighbour path, the receiving routers pick the neighbour advertising the lowest cost, then add this entry into its routing table for re-advertisement. The lowest cost is defined in the following terms: number of hops, time delay in milliseconds, and total number of packets queued along the path.

The router is assumed to know the "distance" to each of its neighbours. If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure it directly with special "Hello" packets that the receiver just timestamps and sends back as fast as it can.

On the other hand, link state routing algorithms require each router maintain at least a partial map of the network, and must:

1. Discover its neighbours and learn their network address.
2. Measure the delay or cost to each of its neighbours.
3. Construct a packet for all it has just learned.
4. Send this packet to all other routers.
5. Compute the shortest path to every other router.

When a network link changes state (up to down, or vice versa), a notification, called a link state advertisement (LSA), is flooded throughout the network. All the routers note the change and re-compute their routes accordingly. The routing algorithms mentioned previously are not suitable for wireless networks, because they generate too much traffic and can only accept a few topology changes per minute.

1.5 ORIGINS OF AD-HOC WIRELESS NETWORKS

Historically, mobile ad-hoc networks have primarily been used for tactical network-related applications to improve battlefield communications and survivability. The dynamic nature of military applications means it is not possible to rely on access to a fixed pre-placed communication infrastructure on the battlefield. The Packet Radio Network (PRNET), under the sponsorship of the Defense Advanced Research Project Agency (DARPA), is considered the precursor of mobile wireless ad-hoc networks (MANET) [7], and its main features include:

1. Flow control over a wireless multi-hop communication route.
2. Error control over wireless links.

3. Deriving and maintaining network topology information.
4. Deriving accurate routing information.
5. Mechanisms to handle router mobility.
6. Shared channel access by multiple users.
7. Processing capability of terminals.
8. Size and power requirements.

PRNET was the first implementation of ad-hoc wireless networks with mobile nodes. This was primarily inspired by the efficiency of packet switching technology, such as bandwidth sharing and store-and-forward routing and its possible applications in mobile wireless environments.

Survivable Radio Networks (SURANs) were deployed by DARPA in 1983 to address open issues in PRNET in the areas of network scalability, security, processing capability, and energy management. The main objectives of this effort were to develop network algorithms to support networks that can scale to tens of thousands of nodes and to resist security attacks, as well as use small, low cost, low-power radio that could support more sophisticated packet radio protocols. This effort resulted in the design of Low-cost Packet Radio (LPR) technology in 1987, which featured a digitally controlled DS spread spectrum radio with an integrated Intel 8086 microprocessor-based packet switch.

Although early MANET application and deployments were military oriented, non-military applications have grown substantially since then and have become the main focus today. This has been the case the last few years, due to rapid advances in mobile ad-hoc networking research; mobile ad-hoc networks have attracted considerable attention and interest from the commercial sector as well as the standards community. The introduction of new technologies such as Bluetooth and IEEE 802.11 greatly facilitate the deployment of ad-hoc technology outside of the military domain. As a result, many new ad-hoc networking applications have since been conceived to help enable new commercial and personal communication beyond the tactical networks domain, including personal area networking, home networking, law enforcement operations, research-and-rescue operations, commercial and educational applications,

sensor networks, and so on. Table 1.1 shows the classification of present and future applications as well as the example services they provide.

Applications	Description/services
Tactical networks	Military communication, operations automated Battlefields
Sensor networks	Collection of embedded sensor devices used to collect real-time data to automate everyday functions. Data highly correlated in time and space, e.g. remote sensors for weather, earth activities, sensors for manufacturing equipment
Emergency services	Search-and-rescue operations as well as disaster recovery; early retrieval and transmission of patient data (record, status, diagnosis) to or from the hospital
Commercial environments	E-commerce: electronic payments from anywhere (i.e. in a taxi) Vehicular services: transmission of news, road conditions, weather, music, local ad-hoc network with nearby vehicles for road/accident guidance
Home and enterprise networking	Home/office wireless networking (WLAN), e.g. shared whiteboard application, use PDA to print anywhere
Educational applications	Set up virtual classrooms or conference rooms, set up ad-hoc communication during conferences, meetings or lectures
Entertainment	Multi-user games, Robotic pets, outdoor Internet access
Locate-aware services	Follow-on services, e.g. automatic call forwarding, transmission of the actual workspace to the current location

Table 1.1: Mobile Ad-hoc Network Applications.

1.6 PEOPLE INVOLVED IN AD-HOC NETWORK RESEARCH

Many people are involved actively in the area of ad-hoc networks. Here, we will mention the most well-known researchers.

Charles E. Perkins is a Research Fellow at Nokia Research Center investigating mobile wireless networking and dynamic configuration protocols. He is working on projects involving Mobile-IP, Ad-hoc networking, IPv6 and AAA protocols. He also has worked on projects involving miniaturization of operating systems, parallel processing and many Unix administration systems.

C-K Toh chairs the IEEE Technical Subcommittee on Ad-hoc Mobile Wireless Networks, and is Director of the Ad Hoc Wireless Networking Consortium. His research interests include: Next Generation High Speed Networks, Broadband Wireless Multimedia Networks, Satellite and Hybrid Networks, Ad-hoc Mobile Networking, Pervasive Networks, Communication Protocols and Architectures, Power Management and Power-efficient protocols, Service and resource discovery, Flow and congestion control, application of control theory to networks, Internet Protocols, Quality of Service Assurance and Adaptation, Location and Context Aware Systems, Mobile Computing and Network Security.

Mohammad Ilyas is a professor of computer science and engineering at Florida Atlantic University, Boca Raton, Florida. His doctoral research was about switching and flow control techniques in computer communication networks. He has conducted successful research in various areas including traffic management and congestion control in broadband/high-speed communication networks, traffic characterization, wireless communication networks, performance modelling and simulation.

Giuseppe Anastasi is currently an associate professor of Computer Engineering at the Department of Information Engineering of the University of Pisa, Italy. His research interests include architectures and protocols for mobile computing, energy management, QoS in mobile networks, and ad-hoc networks.

Elizabeth M. Belding-Royer is an assistant professor in the Department of Computer Science at the University of California, Santa Barbara. Her research focuses on mobile networking, specifically routing protocols, security, scalability and adaptability.

Luciano Bononi is a researcher at the Department of Computer Science of the University of Bologna. His research interests include wireless and mobile ad-hoc networks, network protocols, power saving, modelling and simulation of wireless systems, discrete-event simulation, and parallel and distributed simulations.

Azzedine Boukerche is Canada Research chair and an associate professor of Computer Sciences at the School of Information Technology and Engineering (SITE), University of Ottawa, Canada. His current research interests include ad-hoc networks, mobile computing, wireless networks, parallel simulation, distributed computing and large-scale distributed interactive simulation.

Raffaele Bruno is currently a junior researcher at the IIT Institute of the Italian National Research Council (CNR). His research interests are in the area of wireless and mobile networks with emphasis on efficient wireless MAC protocols, scheduling and scatternet formation algorithms for Bluetooth networks.

Sajal K. Das is a professor of Computer Science and Engineering and also the founding director of the Center for Research in Wireless Mobility and Networking (CReWMaN) at the University of Texas, Arlington (UTA). Dr. Das' current research includes resource and mobility management in wireless networks, mobile and pervasive computing, wireless multimedia and QoS provisioning, sensor networks, mobile internet architectures and protocols, parallel processing, grid computing, performance modelling and simulation.

András Faragó is a professor of Computer Science at the University of Texas, Dallas. His main research interests include the development and analysis of algorithms, network protocols and modelling of communication networks.

Laura Marie Feeney has been a member of the Computer and Network Architecture Laboratory at the Swedish Institute of Computer Science in Kista, Sweden, since 1999. Her research includes topics in energy efficient routing and quality of service for wireless networks, especially ad-hoc and sensor networks.

Enrico Gregori is the deputy director of the Institute for Informatics and Telematics (IIT) of the CNR (Italian National Research Council). His current research interests include ad-hoc networks, sensor networks, wireless LANs, quality of service in packet-switching networks and evolution of TCP/IP protocols.

Xiang-Yang Li has been an assistant professor of Computer Science at the Illinois Institute of Technology since August 2000. His research interests span computational geometry, wireless ad-hoc networks, optical networks and algorithmic mechanism design.

Joseph P. Macker is a senior communication system and network research scientist at the Naval Research Laboratory in Washington, D.C. His primary research interests are adaptive network protocol and architecture design, multicast technology and data reliability, mobile wireless networking and routing, network protocol simulation and analysis, Quality of Service (QoS) networking, multimedia networking and adaptive sensor networking.

Ram Ramanathan is a division scientist at BBN technologies. His research interests are in the area of wireless ad-hoc networks, in particular, routing, medium-access control, and directional antennas.

Andreas Savvides is an assistant professor in Electrical Engineering and Computer Science at Yale University. His research interests are in sensor networks, embedded systems and ubiquitous computing.

Mani Srivastava is a professor of electrical Engineering at UCLA. His current research spans all aspects of wireless, embedded and low power systems, with a particular focus on systems issues and applications in wireless sensor and actuator networks.

Violet R. Syrotiuk is an assistant professor of computer science and Engineering at Arizona State University. Her areas of research include many aspects of medium-access control for mobile ad-hoc networks, such as dynamic adaptation, quality of service, energy awareness, and topology transparency.

Jie Wu is a professor in the department of Computer Science and Engineering, Florida Atlantic University. He researches the areas of mobile computing, routing protocols, fault-tolerant computing, and interconnection networks.

George V. Záruba is an assistant professor of Computer Science and Engineering at the University of Texas, Arlington. Dr. Záruba's research interests include wireless networks, algorithms, and protocols, and performance evaluation concentrating on the medium-access control layer and current wireless technologies.

1.7 INTER-VEHICLE AND VEHICLE TO ROADSIDE COMMUNICATION

The last decade has witnessed an increased interest in inter-vehicle and vehicle to roadside communication, in part, because of the proliferation of wireless networks. Most research in this area has concentrated on vehicle-roadside communication, also called beacon-vehicle communication **[8] [9] [10] [11] [12] [13] [14]**, in which vehicles share the medium by accessing different time slots (Time Division Multiple Access, TDMA), the beacon (down-link direction) and the vehicles (up-link direction). The beacon arranges up-link time slots (so called windows allocations) for the vehicles; the vehicles are not allowed to access the medium without a window allocation sent to them by the beacon. The beacon, as primary station, offers two different types of windows to the vehicles: public and private windows. A public window is a time slot, which may be accessed by every vehicle within the communication zone. A private window allocation reserves a time slot for one specific vehicle, thus protecting it against data collisions.

A typical communication process between a beacon and a vehicle can be divided into two phases: connection establishing phase and transaction phase. When a vehicle enters the beacon's communication zone, the vehicle address is not known to the beacon, so the beacon periodically broadcasts a request for identification and offers a public up-link window (connection establishing phase). After the response of the vehicle has been received, the vehicle address is known and individual addressing becomes possible. The beacon then opens private up-link windows to a specific vehicle for data exchange during the transaction phase.

Due to the limited communication zones (less than 60 meters), some applications for vehicle to roadside communications are mentioned: Automatic Payment, Route Guidance, Cooperative Driving, parking management, etc. However, with the introduction of IEEE 802.11, wireless ad-hoc networks and location-based routing algorithms have made vehicle-to-vehicle communication possible **[15] [16] [17] [18] [19] [20] [21]**. Applications for inter-vehicle communication can be Intelligent Cruise Control, Intelligent Maneuvering Control, Lane Access, Emergency Warning, etc. In **[16]**, the authors propose using Grid **[33]**, which employs a geographic forwarding and scalable distributed location service to route packets from car to car, without flooding the network. The authors in **[17]** propose relaying messages in low traffic densities,

using two components in their simulation environments: a microscopic traffic simulator to produce accurate movement traces of vehicles travelling on a highway, and a network simulator to model the transport of messages among the vehicles. The highway geometry is a straight highway segment with two directions, each composed of one or more lanes. The messages are propagated greedily each time step by hopping to the neighbour closest to the destination. The authors in [18] compare a topology-based approach and a location-based routing scheme. The authors chose GPSR [34] as the location-based routing scheme and DSR [28] as the topology-based approach. The simulator used in [18] is called FARSI, which is a macroscopic traffic model. In [20], the authors compare two topology-based routing approaches, DSR and AODV [27], versus one position-based routing scheme, GPSR, in an urban environment. Finally, in [21], the authors employ a geo-cast routing protocol that is based on AODV.

In inter-vehicle communication, vehicles are equipped with on-board computers and wireless networks, allowing them to contact other similarity equipped vehicles in their vicinity. By exchanging information, in the near future, vehicles may obtain knowledge about the local traffic situation, which may improve comfort and safety in driving.

In this work, we will focus on inter-vehicle communication because vehicle-roadside communication has been already proposed for standardisation in Europe (CEN TC 278 WG 9) [22] and North America (IVHS) [23].

1.8 ISSUES CONCERNING INTER VEHICLE COMMUNICATION USING WIRELESS AD-HOC NETWORKS

Future developments in automobile manufacturing will also include new communication technologies. The major goals are to provide increased automotive safety, to achieve smooth traffic flow on the roads, and to improve passenger convenience by providing them with information and entertainment. In order to avoid communication costs and guarantee the low delays required for the exchange of safety related data between cars, inter-vehicle communication (IVC) systems based on wireless ad-hoc networks represent a promising solution for future road communication scenarios. IVC allows Vehicles to organize themselves locally in ad-hoc networks without any pre-installed infrastructure. Communication in future IVC systems will not be restricted to neighboured vehicles travelling within the radio transmission range, as

in typical wireless scenarios, the IVC system will also provide multi-hop communication capabilities by using "relaying" vehicles that are travelling between the sender and receiver. Figure 1.6 illustrates this basic idea. In this particular example, the source vehicle is still able to communicate with destination vehicle, although the destination vehicle is not in source's vehicle immediate communication range. Vehicles between source-destination act as intermediates vehicles, relaying the data to the receiver. As a result, the multi-hop capability of the IVC system significantly increases the virtual communication range, as it enables communication with more distance vehicles.

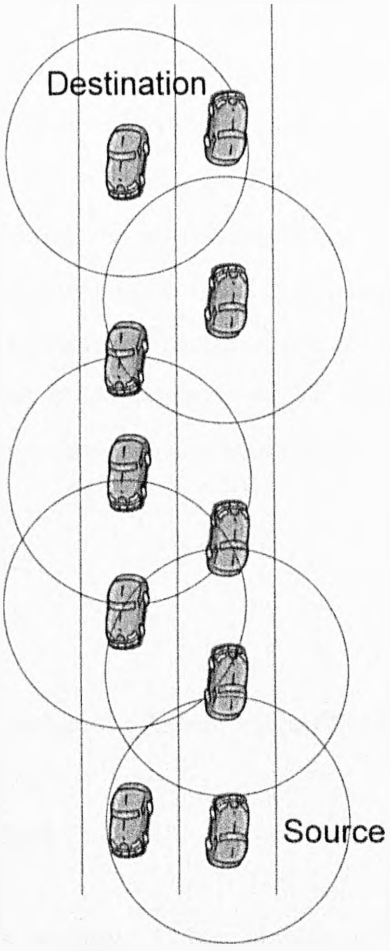


Figure 1.6: Inter-Vehicular Communication System using Wireless Ad-hoc Networks

1.9 PEOPLE INVOLVED IN INTER-VEHICLE COMMUNICATION RESEARCH

Basically, there are three different projects focused on inter-vehicle communication: Fleetnet [19], CarTALK 2000 [24] and CarNet [16]. The objectives of Fleetnet are to develop a communication platform for inter-vehicle communications, to implement demonstrator applications, to develop promising introduction strategies and to standardize the solutions found in order to improve drivers' and passengers' safety and comfort. This project involves several investigations by different companies and universities: DaimlerChrysler AG, Fraunhofer FOKUS, NEC Europe Ltd, Robert Bosh GmbH, Siemens AG, TEMIC Telefunken microelectronic GmbH and the Universities of Mannheim, Hamgurg-Harburg, Karlsruhe, and Hannover. The European project CarTALK 2000 is focussing on new driver assistance systems which are based upon inter-vehicle communication. The main objectives are the development of co-operative driver assistance systems and the development of a self-organizing ad-hoc radio network as the basis for communication with the aim of preparing a future standard.

Under the co-ordination of DaimlerChrysler, the CarTALK 2000 consortium pools the knowledge of European car manufacturers, the IT industry and suppliers and expert knowledge of the following companies and institutions:

1. DaimlerChrysler AG
2. Centro Ricerche Fiat
3. Robert Bosh GmbH
4. Siemens
5. Netherlands Organisation for Applied Scientific Research (TNO)
6. University of Cologne
7. University of Stuttgart

CarNet is an application for large ad-hoc mobile networks systems that scales well without requiring a fixed network infrastructure to route messages. CarNet places radio nodes in cars, which communicate using Grid, a novel scalable routing system. Grid uses geographic forwarding and scalable distributed location service to route packets from car to car without flooding the network. CarNet will support IP connectivity as well as applications such as cooperative highway congestion monitoring, fleet tracking

and discovery of nearby point of interest. The CarNet project has been developed by the MIT Laboratory for Computer Science.

Fleetnet is based on UTRA-TDD (UMTS (Universal Mobile Telephone Service) Terrestrial Radio Access Time Division Duplex. However, because UTRA-TDD is a cellular communication system with centralized organization, its air-interface has to be changed and adapted towards an ad-hoc mode to make it a suitable system for Fleetnet. On the other hand, in CarTALK and CarNet, a wireless system based on IEEE 802.11b has been considered as a possible candidate for the realization of demonstrator prototypes.

In this thesis, we have considered the IEEE 802.11b standard because of its higher market penetration. More than 95 % of today's WLAN infrastructure includes 802.11b products **[5]**. In addition, only the DCF mechanism has been utilized because our work is based on wireless ad-hoc networks. Besides, using the lower ISO OSI layer communication hardware, our main task is to develop a routing protocol that can cope with scalability, high mobility and is completely decentralized.

1.10 SCOPE AND OBJECTIVES OF THE THESIS

The research presented in this Thesis studies the requirements for the provision of inter-vehicular communication on a motorway using ad-hoc wireless networks.

We use intelligent and scoped flooding to fulfil the new routing requirements for highly mobile and dense networks. We extend the relatively new ideas of controlling message dissemination in cluster-based networks.

The first objective of this research is to validate the proposed algorithm with the results of a test-bed on a small scale and on a large scale. Our proposed algorithm is compared with the models of two prominent reactive algorithms: Ad-Hoc On-Demand Distance Vector (AODV) and Dynamic Source Routing (DSR).

The second objective is to compare our proposed algorithm with a popular location-based routing algorithm: Greedy Perimeter Stateless Routing (GPSR).

Finally, the final objective is the statistical evaluation of the communication system and the development, in OPNET, of a Microscopic Traffic model to represent the mobility of the vehicles on a Motorway.

1.11 OUTLINE OF THE THESIS

This dissertation is organised as follows. First, we describe in Chapter 2 an introduction to related work in wireless ad-hoc routing. The first part considers proactive and reactive algorithms. The second part analyses algorithms based on location information. The last part describes cluster-based algorithms. As indicated in this chapter, the unicast problem in highly mobile environments requires knowledge of the destination partner, before sending the first data packet. Chapter 3 will analyse a reactive routing strategy based on location information in the context of inter-vehicular communication. We again discuss related work, now in the area of cluster-based flooding. Then, we derive a predictive algorithm that supports highly mobile nodes. In Chapter 4, we present results of measurements for the justification of wireless ad-hoc networks for inter-vehicle communication. We have divided the chapter in two sections. The first section analyses large-scale models and the second section analyses small-scale models. In large-scale models, three aspects are considered: free-space propagation modes, path loss and system operating margin. On the other hand, in small-scale models, the impact of Doppler shift is considered. The chapter finishes with some experiments that allow us to determine the interval of packets received between communication partners in the worst case scenario, when the vehicles are travelling in opposite directions. Chapter 5 will focus on microscopic traffic models. The chapter starts with an introduction to vehicular traffic theory and the most important models that describe the human driver behaviour for car-following models. In addition, the chapter analyses the most popular microscopic traffic models and its suitability for simulation. Chapter 6 contains the validation and verification of the simulated models. The analysis of Dynamic Source Routing (DSR) and Ad-Hoc On-Demand Distance Vector (AODV) is first considered. For small-scale networks, the validation is executed with the results of a test-bed. Then, for large scale-networks, the proposed algorithm is compared with the models of two prominent ad-hoc routing algorithms: DSR and AODV on a multi-lane rectangular and circular dual carriageway representative of city and motorway driving. Finally, our protocol is compared on a multi-lane circular carriageway representative of a six lane motorway driving with one location-based routing algorithm: Greedy Perimeter Stateless Routing (GPSR). Chapter 7 shows the conclusion and provides suggestion for future investigation.

CHAPTER 2: ROUTING ALGORITHMS IN WIRELESS AD-HOC NETWORKS

2.1 INTRODUCTION

There are two classes of routing algorithms for wired networks: *distance vector routing* and *link state routing* [6]. Distance vector algorithms are based on the exchange of only a small amount of information, where each node keeps a routing database with one entry of the previous information, for every possible destination in the system. On the other hand, in link state routing, a node floods throughout the network the list of its neighbours.

In addition to the challenges presented by routing in traditional wired networks, routing algorithms for wireless environments with mobility are very different than those for wired networks with static nodes. Major problems in mobile ad-hoc networks include limited bandwidth and a high rate of topological change.

Topological change implies the limited lifetime of topology information held at any time due to the movements of the nodes. This implies that unless the information is updated regularly, it becomes invalid. The more frequently the information is updated, the greater the node mobility that can be handled correctly and efficiently.

Traditional routing algorithms based on non-positional information are suitable for scenarios with low node mobility. However, these routing algorithms are unsuitable in scenarios with high mobility. For this reason, we have developed an algorithm that includes topological information inside of its routing mechanism.

The wireless nature of the medium implies limited available bandwidth, which is even further reduced because of the high bit error rate in radio transmission.

On one hand, mobility requires topological information be sent more frequently to maintain the nodes informed of the changes. On the other hand, the bandwidth of the wireless medium is reduced with the constant communication between nodes. Therefore, the compromise is to manage the mobility of the nodes using the minimum of the bandwidth resources.

Routing algorithms in mobile ad-hoc wireless networks can be categorised into two different categories: non-positional algorithms and positional algorithms (Figure 2.1).

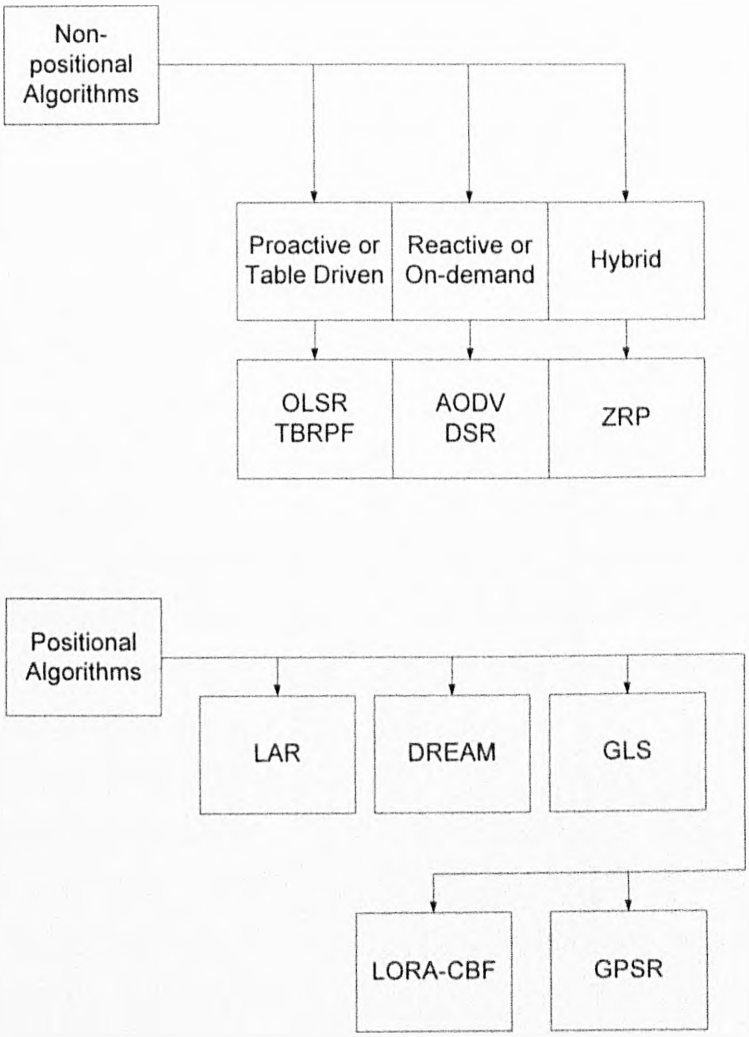


Figure 2.1: Routing algorithms in mobile wireless ad-hoc networks.

There are three ways to classify non-positional algorithms: proactive or table-driven; reactive or on-demand; or hybrid. Proactive, or table-driven algorithms, periodically update the reachability information in the nodes' routing tables, making routes immediately available when needed. However, these algorithms require additional bandwidth to periodically transmit control traffic. Each node maintains the necessary routing information and each node is responsible for propagating topology updates in response to instantaneous connectivity changes in the network **[25]**. Examples of such

protocols include Optimized Link State Routing (OLSR) protocol [26] and Topology Dissemination Based on Reverse Path Forwarding (TBRPF) [27]. These two protocols keep track of routes for all destinations in the ad hoc network, and have the advantage that communications with arbitrary destinations experience minimal initial delay (latency). Such protocols are called proactive because they store route information even before it is needed, and are table driven because the information can be seen as available as part of well-maintained table.

On the other hand, on-demand, or reactive protocols, have emerged to acquire routing information only when it is needed. Reactive routing protocols often use less bandwidth for maintaining route tables, but the Route Discovery (RD) time for many applications will increase. Most applications are likely to suffer a delay when they start because a route to the destination will have to be acquired before the communications can begin.

As a result, a route discovery process is realized before the first data packet can be sent. These types of protocols reduce control traffic overhead at the cost of increased latency in finding the route to a destination [28]. Examples of reactive or on-demand protocols include Ad-Hoc On-Demand Distance Vector (AODV) routing [29], and Dynamic Source Routing (DSR) algorithm [30].

A protocol that combines both proactive and reactive approaches is called hybrid [31]. The most popular protocol in this category is the Zone Routing Protocol (ZRP). ZRP divides the network into overlapping routing zones, allowing for the use of independent protocols within and between the zones. ZRP is called hybrid because it combines proactive and reactive approaches to maintain valid routing tables without much overhead. Communication inside of the zone is realized by the Intra-zone Routing Protocol (IARP) that provides direct neighbour discovery (proactive routing). On the other hand, the communication between the different zones is realized by the Inter-zone Routing Protocol (IERP) and provides routing capabilities among nodes between zones (reactive routing).

The main disadvantages of pure proactive and reactive routing algorithms in highly mobility environments are their poor route convergence. They also have a very low communication throughput because of the high number of retransmissions. [32].

To overcome these limitations, several new types of routing algorithms have been developed using geographic position information: Location-Aided Routing (LAR) [33],

Distance Routing Effect Algorithm for Mobility (DREAM) [34], Grid Location Service (GLS) [35] and Greedy Perimeter Stateless Routing for Wireless Networks (GPSR) [36]. Trends suggest that, in the near future, a broad variety of location dependent services will become feasible using to the use of Global Position System (GPS). In vehicular technology inter and intra vehicular communication can be improved with the help of wireless ad-hoc networks and GPS [37].

The type of control traffic that is generated to manage the mobility of the nodes in a network is the information that a node declares about its relative movement: its new position, or the new neighbourhood of the node which is declaring the information. In these situations, we are not concerned about the information propagating across the entire network to every node. But in many cases, not only the immediate neighbour of the declaring node, but other nodes further away may also need to know the topological changes occurring anywhere in the network. In these situations, a lot of messages must pass through the network to keep the information consistent and valid at each node, by regularly announcing the changes due to the mobility, or failure of a link, etc.

These announcements about the changes are propagated to each node of the network. But in most cases, all the nodes of the network are not in radio range of each other and cannot communicate directly. So there must be a mechanism to reach distant nodes to keep them informed of changes. The concept of intermediate nodes, which serve as relays to pass the messages between the source and the destination, arises from this need.

When a message is for a specific destination, all nodes which form the path from the source up to the destination become intermediate nodes.

An interesting problem arises when the packet is to be broadcast. Now, the task of determining the intermediate nodes to forward the packet is less straight forward. A simple solution is for each node re-transmits the message when it receives it the first time. Figure 2.2 shows an example where a packet, originated by node *S*, is diffused up to 3-hops with 49 retransmissions. In the figure, we have shown the propagation of the message up to 3 hops. The packet is retransmitted by all the nodes, including the leaf nodes (or boundary nodes) in order to be diffused across the whole network. This is called "pure flooding". It consumes a large amount of bandwidth due to redundant retransmissions.

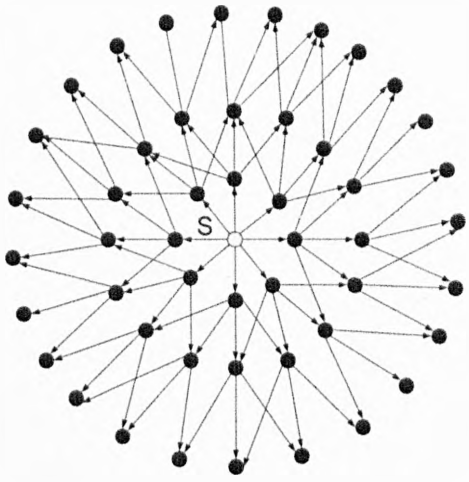


Figure 2.2: Diffusion of broadcast message using pure flooding.

Several protocols use some kind of flooding of control messages because it is very advantageous to optimize the bandwidth. There are many techniques described in the literature to limit flooding of broadcasting traffic [26 [27].

2.2 PROACTIVE ROUTING ALGORITHMS FOR AD-HOC NETWORKS

There are two main proactive algorithms that still remain within the MANET working group [38], that we will consider here for our study: Optimised Link State Routing Protocol (OLSR) and Topology Dissemination based on Reverse-Path Forwarding (TBRPF).

2.2.1 Optimised Link State Routing Protocol (OLSR) for Ad-Hoc Networks

The OLSR protocol is an optimisation of the pure Link State Algorithm, which was developed for mobile ad hoc networks by Thomas Clausen, et al [39]. The OLSR protocol operates as a table driven or proactive protocol. The key concept used in OLSR is the use of Multipoint Relays (MPRs) (Figure 2.3). The Multipoint Relay of node *n* is selected between the other neighbours and considers a simple heuristic; the Multipoint

Relay is the neighbour which covers the maximum number of two hop neighbours **[40] [41] [42]**. MPRs are selected nodes, which forward broadcast messages during the flooding process.

This technique reduces message overhead as compared to pure flooding mechanism, where every node retransmits each message when it receives the first copy of the packet. In OLSR, only nodes elected as MPRs generate link state information. The protocol inherits the stability of a link state algorithm and has the advantage of having routes immediately available when needed due to its proactive nature. OLSR minimizes the overhead from flooding of control traffic by using only selected nodes, called MPRs, to retransmit control messages. Furthermore, a second optimisation can be achieved by minimizing the number of control messages flooding the network, and as a third optimisation, an MPR node may choose to report only links between itself and its MPR selectors **[41]**. Hence, contrary to the classical link state algorithm, only partial link state information is distributed in the network. This information is then used by the OLSR protocol for route calculation. OLSR provides optimal routes (in terms of number of hops).

The protocol is particularly suitable for large and dense networks as the technique of MPRs works well in this context. As OLSR continually maintains routes to all destinations in the network, any link change will cause OLSR sent packets to be flooded across the entire network, immediately causing excessive routing overhead, which can be very significant in large, highly mobile networks **[40]**.

Results from simulations in OLSR, carried out by Floyd and Van Jacobson **[43]**, have shown that the periodic transmission of control messages from individual nodes can become synchronized. For example, neighbouring nodes would, at the same time, attempt to transmit a TC message, causing both messages to be lost due to collisions. Enforcing jitter (a small random delay) significantly improves the performance **[40] [44]**. This can be explained by the reduction in the collision rate for the broadcast traffic.

An extension for fast mobility in OLSR has also been proposed **[45] [46]**. Here, a higher Hello frequency is proposed to accommodate node mobility. A node in a Fast-Moving mode sends Fast-Hello messages at high frequency. The concerns include higher mobility and higher overhead. Also, results have shown that the packet loss increases as a function of speed **[45]**.

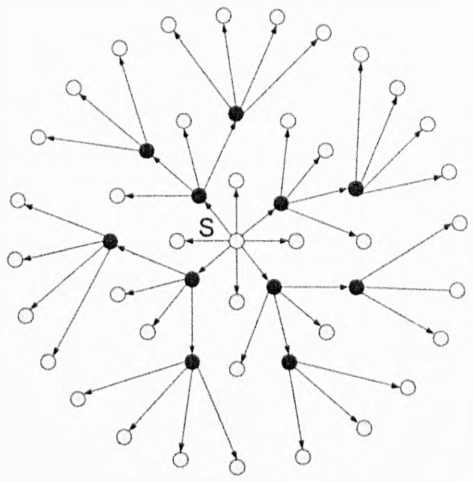


Figure 2.3: Multipoint relays (MPRs).

2.2.2 Topology Dissemination based on Reverse-Path Forwarding (TBRPF)

Topology Dissemination, based on Reverse-Path Forwarding, is a proactive link state routing protocol designed for mobile ad hoc networks by Richard Ogier, et al. [47]. TBRPF consists of two main modules: the Neighbour Discovery module, and the Routing module (which performs topology discovery and route computation). The Neighbour Discovery module uses differential Hello messages, which include only the Ids of new neighbours and recently lost neighbours. In the Routing module, each node reports only a portion of its source tree to all neighbours using periodic updates (e.g. every 5 or 6 seconds) or differential updates (e.g. every 1 or 2 seconds).

TBRPF provides hop-by-hop routing along the shortest paths to each destination. The key feature of TBRPF Neighbour Discovery (TND) protocol is that it uses "differential" Hello messages which report only neighbours whose state has recently changed. This results in Hello messages that are much smaller than those of other link-state routing protocols such as OSPF can be sent more frequently, thus allowing faster detection of topology changes.

Each node running TBRPF computes a source tree (providing paths to all reachable nodes) based on partial topology information stored in its topology table (Figure 2.4). To minimize overhead, each node reports only "part" of its source tree to neighbours.

TBRPF uses a combination of periodic and differential updates to keep all neighbours informed of the reportable part of its source tree. Each node also has the option of reporting additional topology information (up to the full topology), to provide improved robustness in highly mobile networks.

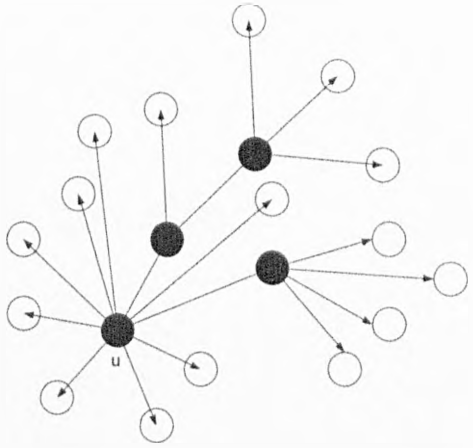


Figure 2.4: Example illustrating TBRPF source tree.

Proactive algorithms keep the information of all the nodes in the wireless ad-hoc network, requiring additional control traffic to continually update route entries. Consequently, proactive protocols suffer the disadvantage of requiring additional bandwidth resources [48] [49].

2.3 ON-DEMAND ROUTING ALGORITHMS FOR AD-HOC NETWORKS

2.3.1 Ad-Hoc On-demand Distance Vector Routing (AODV)

Ad Hoc On-demand Distance Vector Routing was developed by C. Perkins, et al [50]. It minimises the number of broadcasts by creating routes on-demand. AODV uses traditional routing tables to maintain routing information in one entry per destination [25]. The source broadcasts a route request packet, to find the route to a destination. The neighbours' nodes also propagate the route request packet to their neighbours until a packet reaches an intermediate node that has a recent route for the specific destination, or until it reaches the destination (Figure 2.5).

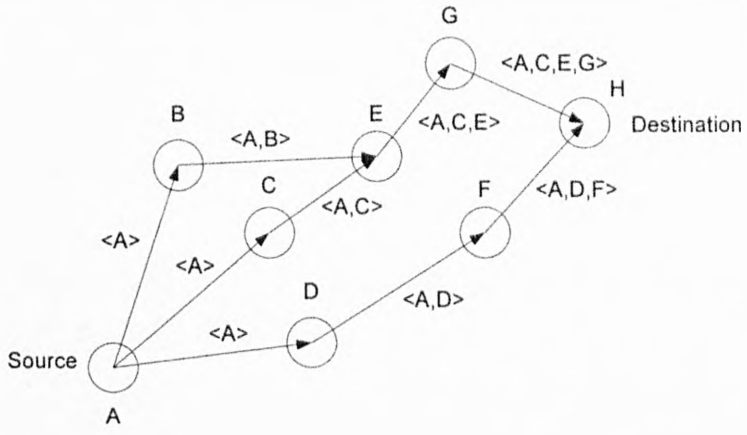


Figure 2.5: Route Request Packet in AODV.

Figure 2.5 shows that the source node broadcasts a Route Request Packet (RREQ). When the RREQ packet reaches node B, C and D, it is registered in each node. A node discards a route request packet that it has already seen. The route request packet uses a sequence of numbers to ensure that the routes are loop free. Before a node forwards a route request packet to its neighbours, it records the node information in its table. This information is used to construct the reverse route for the route reply packet (Figure 2.6).

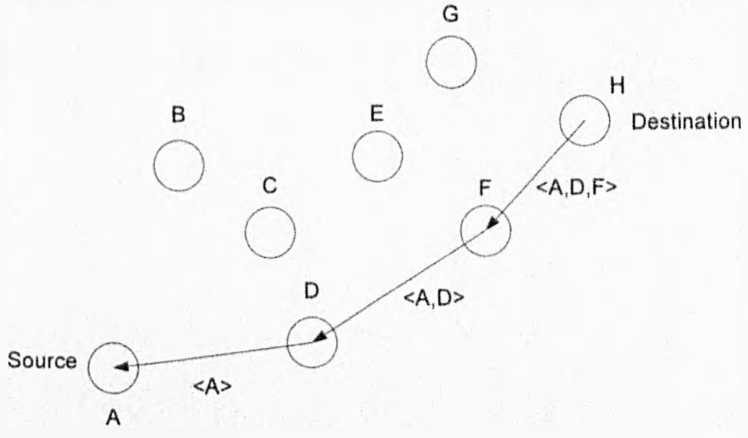


Figure 2.6: Route Reply Packet in AODV.

The recent specification of AODV **[51]** includes an optimization technique to control the Route Request packets (RREQ) flood in the route discovery process. It uses an *expanding ring* search initially to discover routes to an unknown destination.

A drawback with AODV is that it requires the dissemination of Route Error (RERR) packets when a link break in an active route is detected. A RERR message is used to

notify other nodes that the loss of that link has occurred. The RERR message is sent whenever a link break causes one or more destinations to become unreachable from some of the node's neighbours. Since link loss is common in mobile environments AODV will use RERR packets with greater frequency.

2.3.2 The Dynamic Source Routing Protocol for Mobile Ad-Hoc Networks (DSR)

The Dynamic Source Routing Protocol was developed by David B. Johnson, et al **[52]**. It is an on-demand routing protocol based on source routing. DSR requires no periodic packets of any kind at any layer within the network. For example, DSR does not use any periodic routing advertisement, link status sensing, or neighbour detection packets, and does not rely on these functions for any underlying protocols in the network. A node maintains routes containing the source routes to every node in the network. The protocol consists of two major phases: route discovery and route maintenance **[53]**. When the source node wants to send a packet to a destination, it looks up its route cache to determine if it already contains a route to the destination. If it finds that an existing unexpired route to the destination, it will use this route to send the packet. But if the node does not have a route, it will initiate the route discovery process by broadcasting a route request packet similar to AODV (Figure 2.7).

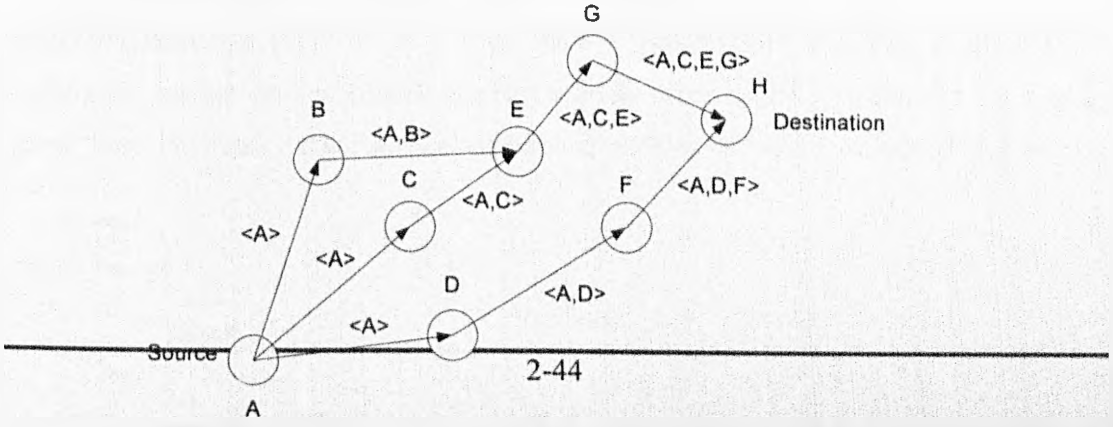


Figure 2.7: Route Request Packet in DSR.

The route request packet contains the address of the source and the destination, and a unique identification number. Each intermediate node checks whether it knows the route to the destination. If it does not, it includes its address in the route record of the packet and forwards the packet to its neighbours. To limit the number of route requests propagated, a node processes the route request packet only if it has not already seen the packet and its address is not present in the route record of the packet.

A route reply packet is generated when either the destination or an intermediate node with current information about the destination receives the route request packet. The route record of a route request packet contains the sequence of hops from the source to the destination (Figure 2.8).

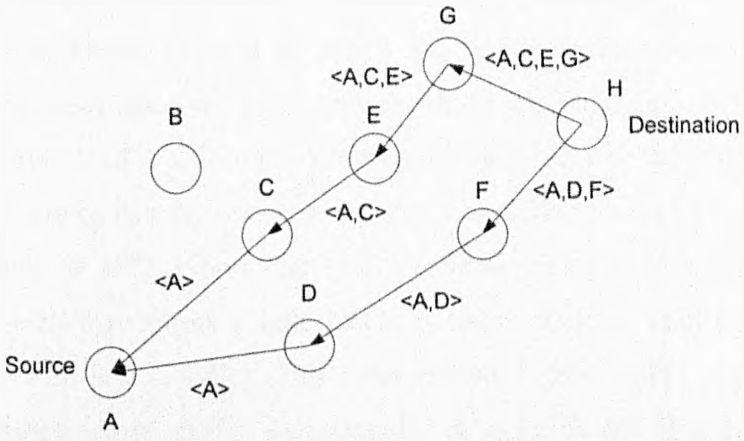


Figure 2.8: Route Reply Packet.

The algorithm uses two types of packets for route maintenance: Route Error and Acknowledgements **[54]**. When a node finds a transmission problem, it generates a route error packet. When a node receives a route error packet, it removes the hop in error from its route cache. Acknowledgment packets are used to verify the correct

operation of the route links. Two types of acknowledgments are specified in DSR: link-level acknowledgment and passive acknowledgment [54].

The main drawback of DSR is that it is designed for mobile ad hoc networks of up to about two hundred nodes, because it sends the entire route from the source to the destination in every data packet. In environments with high mobility, it is difficult to guarantee the path from source to destination, so the protocol will send Route Error packets more frequently, causing possible network congestion.

2.3.3 Performance comparison between AODV and DSR

Performance comparison between AODV and DSR, focusing on overhead, routing load, delivery ratio and end-to-end delay has been reported in the literature. Authors in [49], report that AODV requires about 5 times the overhead of DSR when there is constant node mobility, meaning that nodes involved in the communication are not stationary. DSR limits the scope and overhead of Route Request packets by using caching from forwarded and promiscuously over-heard packets and using non-propagation Route Request. DSR always has a lower routing load than AODV [55] [56] [57]. DSR makes effective use of route caching and the destination replies to all route discovery messages because of which the source learns about multiple path(s). In AODV, the destination replies to only the first RREQ and ignores the rest. A relatively stable routing load is a desirable property for scalability of the protocols. AODV has a slightly worse packet delivery performance than DSR because of higher drop rates [56]. The authors in [57], report that DSR performs better in less demanding situations. Finally, AODV performs a little better in terms of delay than DSR [56]. The delay increases with low mobility. This phenomenon is less visible in more highly mobile environments where traffic automatically is more evenly distributed due to source movements [57].

2.4 ALGORITHMS BASED ON POSITION INFORMATION (GEOGRAPHIC COORDINATES)

In inter-vehicle communications, the vehicles form a mobile ad-hoc network which consists of mobile hosts that communicate via a wireless link. Due to mobility, the topology of the network changes continuously and wireless links establish and break down frequently. Moreover, the ad-hoc network operates in the absence of fixed infrastructure, forcing the hosts to organize the exchange of information in a distributed manner. We consider here the most representative algorithms that might be suitable for highly mobile environments. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks, Distance Routing Effect Algorithm for Mobility (DREAM), Grid Location Service (GLS) and Greedy Perimeter Stateless Routing for Wireless Networks (GPSR).

2.4.1 Basic Principles and problems of position-based routing algorithms

Before a packet can be sent, it is necessary to determine the position of its destination. Typically, a location service is responsible for this task. Existing location services can be classified according to how many nodes host the service. This can be either some specific nodes or all nodes of the network. Furthermore, each location server may maintain the position of some specific or all nodes in the network. We abbreviate the four possible combinations as some-for-some, some-for-all, all-for-some and all-for-all.

In position-based routing, the forwarding decision by a node is primarily based on the position of a packet's destination and the position of the node's immediate one-hop neighbour. The position of the destination is contained in the header of the packet. If a node has a more accurate position of the destination, it may choose to update the position in the packet before forwarding it. The position of the neighbours is typically learned through a one-hop broadcast beacon. These beacons are sent periodically by all nodes and contain the position of the sending node.

We can distinguish three main packet-forwarding strategies for position-based routing: greedy forwarding, restricted directional flooding, and hierarchical approaches. For the first two, a node forwards a given packet to one (greedy forwarding) or more (restricted directional flooding) one-hop neighbours that are located closer to the destination than the forwarding node itself. The selection of the neighbour in the greedy case depends on the optimization criteria of the algorithm. The third forwarding strategy is to form a hierarchy in order to scale to a large number of mobile nodes. We now consider each of

the previous position-based routing algorithms mentioned for suitability to our proposed network of highly mobile vehicles.

2.4.2 Location-Aided Routing (LAR) in Mobile Ad-Hoc Networks

The Location-Aided Routing (LAR) algorithm was developed by Young-Bae Ko, et al **[58]**. LAR does not, in fact, define a location-based routing protocol. Instead, it proposes using of position information to enhance the route discovery phase of reactive ad-hoc routing approaches. Reactive ad-hoc routing protocols frequently use flooding as a means of route discovery. Under the assumption that nodes have information about other nodes' positions, this position information can be used by LAR to restrict the flooding to a certain area.

The LAR algorithm has two schemes:

LAR scheme 1:

The first scheme uses a request zone that is rectangular in shape (Figure 2.9). In this scheme, the request zone is defined as the smallest rectangle that includes the current location of *S* and the expected zone. When a node receives a route request, it discards the request if the node is not within the rectangle specified by the four corners included in the route request.

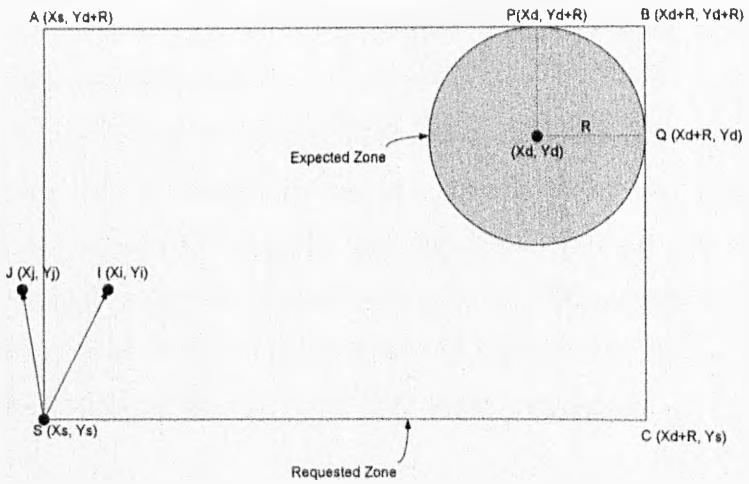


Figure 2.9: LAR scheme 1.

LAR scheme 2:

In LAR scheme 1, source *S* explicitly specifies the request zone in its route request message. In scheme 2, node *S* includes two pieces of information with its route request:

- Assuming that node *S* knows the location (Xd, Yd) of node *D* at some time previous to the moment in which route discovery is initiated, node *S* calculates its distance from the location (Xd, Yd), referred to as DISTs, and includes this distance with the route request message (Figure 2.10).
- The coordinates (Xd, Yd) are also included with the route request.

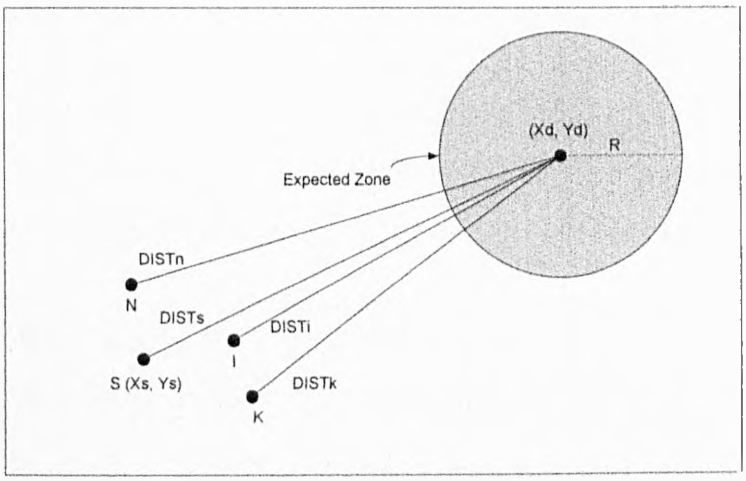


Figure 2.10: LAR scheme 2.

LAR does not define a location-based routing protocol. Instead, it uses position information to enhance the route discovery phase. This is one of the directional flooding-based routing methods **[59]**.

A drawback in LAR is that the source node assumes the location and average speed of the destination node. If the sender has no information about the location of the destination node, the algorithm behaves like unlimited flooding and spreads the message in all directions to find the destination node. After the request has reached the destination, the discovered route is sent back to the initiator. In a second phase, the originator sends the data along the discovered path to the destination.

2.4.3 A Distance Routing Effect Algorithm for Mobility (DREAM)

The Distance Routing Effect Algorithm for Mobility was developed by Stefano Basagni, et al **[34]**. The DREAM algorithm can be considered proactive and it introduces two concepts: distance effect and mobility rate.

In DREAM, each node maintains a position database that stores position information about each other node that is part of the network. It can therefore be classified as an all-for-all approach. An entry in the position database includes a node identifier, the direction and the distance to the node, as well as a time value that indicates when this information was generated. The accuracy of such entry depends on its age. Each node regularly floods packets to update the position information maintained by the other nodes. A node can control the accuracy of its position information available to other nodes by:

- sending position updates (temporal resolution) and
- indicating how far a position update may travel before it is discarded (spatial resolution).

The temporal resolution of sending updates is coupled with the mobility rate of a node. The mobility rate refers to the movement of nodes and states that the faster a node moves, the more often it must communicate its location. This allows each node to self optimize its dissemination frequency, thus transmitting location information only when needed and without sacrificing route accuracy.

Spatial resolution is used to provide accurate position information in the direct neighborhood of a node and less accurate information for nodes farther away. The cost associated with accurate position information at very remote nodes can be reduced as a consequence of the distance effect. The term distance effect refers to the distance separating two nodes and states that the greater the distance separating two nodes, the slower they appear to be moving with respect to each other. Thus, nodes that are far apart, need to update each other's locations less frequently than nodes closer together. This is realized by associating an "age" with each control message which corresponds to how far from the sender that message has travelled. An example of this distance effect is given in Figure 2.11.

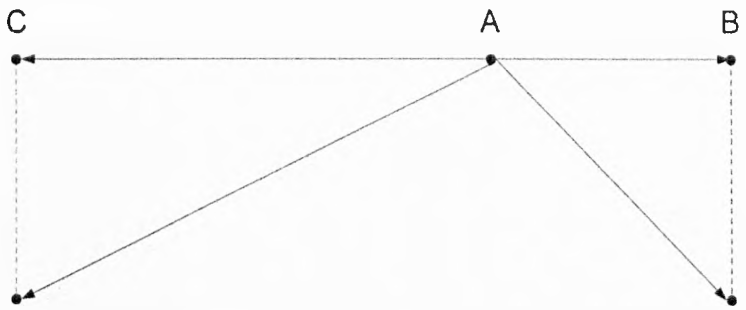


Figure 2.11: Distance Effect in DREAM.

The diagram in Figure 2.11 assumes that node A is stationary, while nodes B and C are moving in the same direction at the same speed. From node A's perspective, the change in direction will be greater for node B than for node C. The distance effect allows having a low spatial resolution in areas that are far away from the target node, provided that intermediate hops are able to update the position information contained in the packet.

In DREAM, the sender, *S*, of a packet to the destination *D* will forward the packet to all one-hop neighbours that lie in the direction of *D*. In order to determine this direction, a node calculates the region that is likely to contain *D*, called the expected region. As depicted in figure 2.11, the expected region is a circle around the position of *D*, as it is known to *S*. Since this position information may be outdated, the radius *r* of the expected region is set to $(t_1 - t_0)\ v_{max}$, where t_1 is the current time, t_0 is the timestamp of the position information that *S* has about *D*, and v_{max} is the maximum speed that the node may travel in the ad-hoc network. Given the expected region, the 'direction toward *D*' for the example given in figure 2.12 is defined by the line between *S* and *D* and the angle @. The neighbouring hops repeat this procedure using their information on *D*'s position.

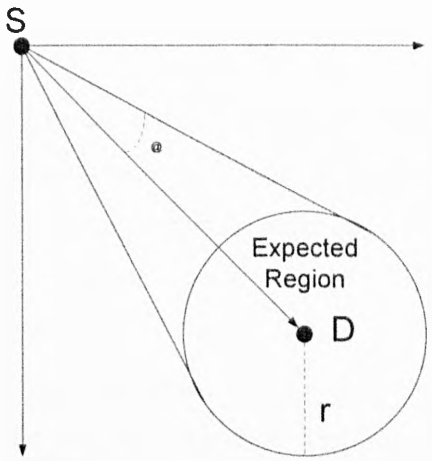


Figure 2.12: The expected region in DREAM.

DREAM is proactive in nature and may have issues relating to its scalability and may not therefore be appropriate for large scale networks [32].

2.4.4 Grid Location Service (GLS)

Grid Location Service was developed by Jinyang Li, et al [35]. Grid Location Service (GLS) divides the area that contains the ad hoc network into a hierarchy of squares. In this hierarchy, n-order squares contain exactly four (n-1)-order squares, forming a so called quad tree. Each node maintains a table of all other nodes within the local first order square. The table is constructed with the help of periodic position broadcasts which are scoped to the area of first-order square (Figure 2.13).

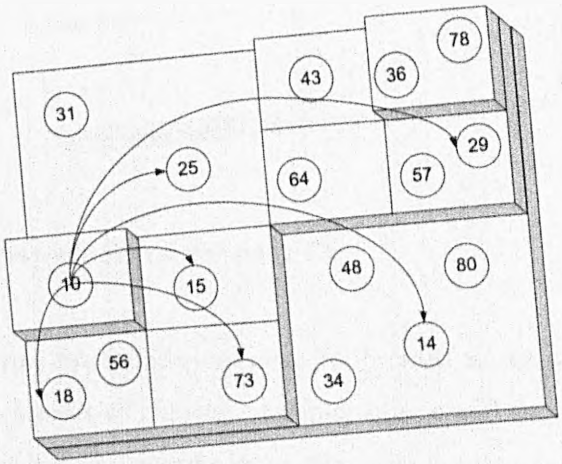


Figure 2.13: Grid Location Service (GLS) Algorithm.

Figure 2.13, shows the mechanism used to store position information. GLS establishes a notion of near node Ids, defined as the least ID greater than a node's own ID. When node 10 in the example wants to distribute its position information, it sends position updates to the respective node with the nearest ID in each of the three surrounding first-order squares. Thus, the position information is available at the nodes 15, 18, 73 and at all nodes that are in the same first-order square as 10 itself. In the surrounding three second-order squares, again the nodes with the nearest ID are chosen to host the node's position. In figure 2.14, for example, these are nodes 14, 25 and 29. This process is repeated until the area of the ad-hoc network has been covered.

Assume now that node 78 wants to obtain the position of node 10. In order to do this, it must locate a nearby node that knows the position of node 10. In Figure 2.14, this is node 29. But despite node 78 not knowing that node 29 holds the required position, it can discover this information. To see how this process works, it is useful to take a look at the position servers for node 29. Its position is stored in the three surrounding first-order squares at nodes 36, 43, and 64.

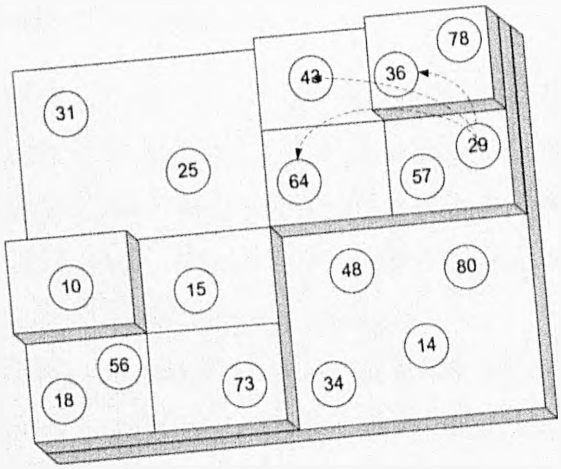


Figure 2.14: Position Server for node 29.

Position queries for a node can now be directed to the node with the nearest ID the querying node knows of. In our example, this would be node 36. The node with the nearest ID does not necessarily know the node being sought, but it will know a node

with nearer node ID (node 29, which is already the sought-after position server). The process continues until a node that has the position information available is found.

Since GLS requires that all nodes store the information on some other nodes, it can be classified as an all-for-some approach. GLS, although hierarchical, is also proactive and is thought not to be efficient in highly dynamic environments [32].

2.4.5 GREEDY PERIMETER STATELESS ROUTING FOR WIRELESS NETWORKS (GPSR)

The Greedy Perimeter Stateless Routing algorithm (GPSR) was developed by Karp and Kung [36]. In GPSR, routers decide where to forward a packet using the information about immediate neighbours. By keeping information only about the local topology, GPSR scales well with the number of network nodes and a frequently changing network topology. The GPRS algorithm solves the task of geographic routing using two methods for forwarding packets: Greedy Forwarding and perimeter forwarding.

2.4.5.1 Greedy Forwarding

In greedy forwarding, a forwarding node can make a locally optimal, greedy choice in choosing a packet's next hop. This is done if a node knows its radio neighbour's position. The locally optimal choice of next hop is the neighbour geographically closest to the packet's destination. Forwarding in this regime follows successively closer geographic hops, until the destination is reached.

A simple beaconing algorithm provides all nodes with their neighbours' positions. Periodically, each node transmits a beacon to the broadcast MAC address, containing only its own identifier (e.g. IP address) and position.

2.4.5.2 Perimeter Forwarding

Perimeter forwarding is called the right-hand rule. This rule states that when arriving at node x from node y, the next edge traversed is the next one sequentially, counter clockwise edge order, which, in this case, is the triangle bounded by the edges between nodes x, y and z, in the order ($y \rightarrow x \rightarrow z \rightarrow y$) (Figure 2.15). The rule traverses an

exterior region, which, in this case, is the region outside the same triangle, in counter clockwise edge order.

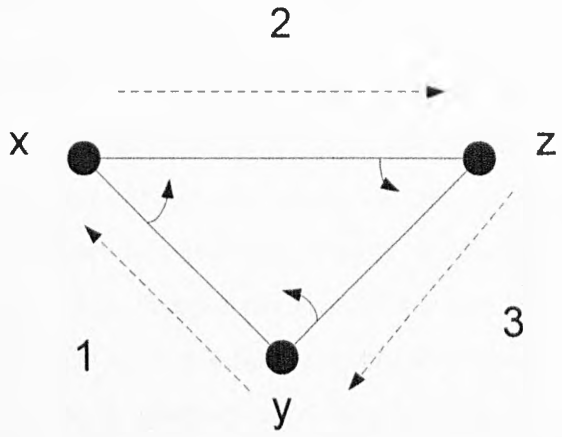


Figure 2.15: The right hand rule.

2.5 CLUSTER-BASED ROUTING ALGORITHMS FOR AD-HOC NETWORKS

Proactive and reactive protocols have limited scalability, due to their route discovery and maintenance procedures [60]. One alternative to these protocols for improving scalability is clustering, or hierarchical routing protocols. Hierarchical protocols place nodes into groups, often called clusters. These groups may have some sort of cluster leader that is responsible for route maintenance within its cluster and between other clusters. Basically, cluster-based routing algorithms can be widely classified into two different categories: clustering for transmission management, and clustering for backbone formation [25].

2.5.1 Clustering for Transmission Management

One problem associated with wireless mediums is the shared nature of the channel. For example, on the physical layer, when one node transmits information to a different node, all neighbouring nodes will receive the message. Nodes can reduce the possibility of interference by separating transmissions in time, space, frequency, or via spreading codes.

2.5.1.1 Link-Clustered Architecture

The Link-clustered architecture reduces interference in a multiple-access broadcast environment by forming distinct clusters of nodes in which transmission can be scheduled in a contention-free manner [61]. Transmission in adjoining clusters can be isolated through spread-spectrum multiple access by using different spreading codes in each cluster. Formation of clusters is realized as follows: each cluster contains a cluster-head, one or more gateways, and zero or more ordinary nodes that are neither cluster-heads nor gateways. The cluster-head schedules a transmission and allocates resources within the clusters. The main rule for a gateway node is to establish connection between two adjacent clusters. A gateway may directly connect two clusters by acting as a member of both, or it may indirectly connect two clusters by acting as a member of one and forming a link to a member of the other. Hence, the link-clustered architecture accommodates both overlapping and disjointed clusters (Figure 2.16).

At least two cluster-head election algorithms have been proposed for the link-clustered architecture: *identifier-based clustering and connectivity-based clustering* [62]. The node with the lowest or highest numbered identifier (identifier-based clustering) or with the largest number of neighbours (connectivity-based clustering) is chosen as a cluster-head.

The link-clustered architecture provides a natural routing backbone consisting of cluster-heads and gateways and the links between them. A drawback of this algorithm is that by obliging all traffic to traverse cluster-heads, network throughput and robustness may be reduced. By employing Cluster-heads as the point of traffic, each cluster head can potentially become congested, thus representing a failure point for communication across its cluster.

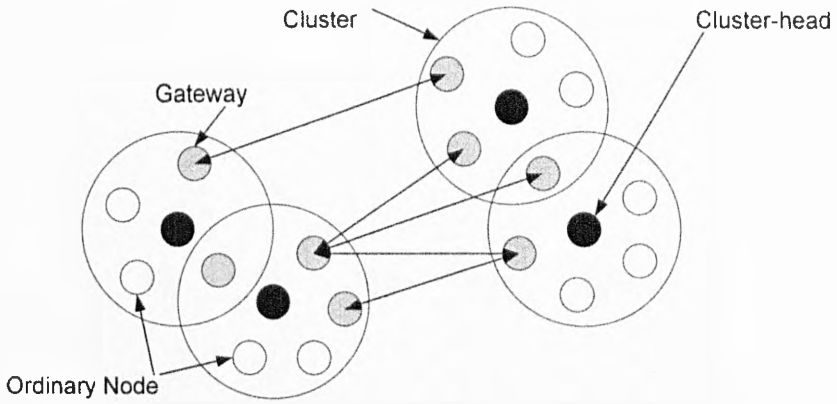


Figure 2.16: The Link-Clustered Architecture.

2.5.1.2 Cluster-Based Routing Protocol (CBRP)

Cluster-Based Routing Protocol (CBRP) was developed by Mingliang Jiang, et al **[63]**. It is a hybrid protocol, using a two-level hierarchical routing mechanism. It groups the network nodes into clusters (Figure 2.17), having a cluster-head which has the knowledge of cluster members. The cluster-head also knows the inter-cluster link state topology through the gateway nodes. CBRP employs a proactive scheme inside the cluster, and uses a reactive protocol for inter-cluster routing, with route optimizations by cluster-heads. Grouping the nodes into clusters provides an optimization while discovering routes on-demand, as it produces less flooding traffic. CBRP is suitable for middle to large networks with slow node movements. In CBRP, each node maintains a Neighbour Table in which it keeps information about its neighbours, a Cluster Adjacency Table in which it keeps information about how to reach adjacent clusters, and a Two-hop Topology Database in which it registers all its two-hop neighbours. Each node periodically sends a Hello message, containing its neighbours. When a node starts up, it is neither a cluster member nor a cluster head. It discovers, through its Hello messages, one or more cluster-heads which are its neighbours. It then selects one cluster-head, and registers itself in the cluster. If a node does not find any cluster-head, and has some neighbours with a bidirectional link, it declares itself as a cluster-head.

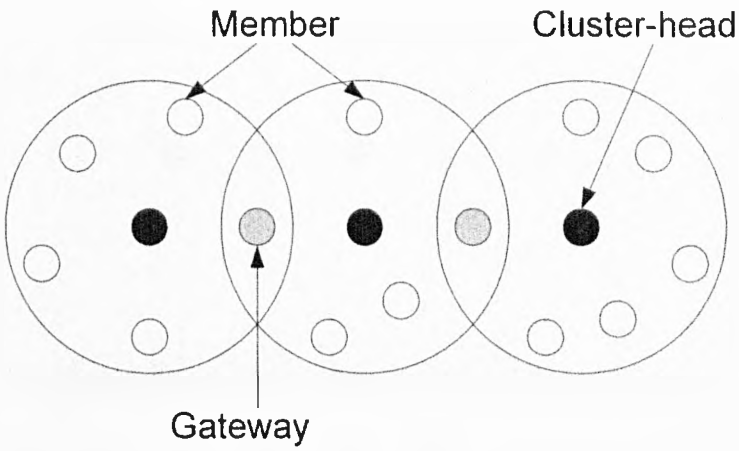


Figure 2.17: Clusters' Formation.

Route discovery in CBRP uses the flooding method. An RREQ (Route Request) message is sent by the source node to its cluster-heads, and all the known adjacent cluster-heads. If the destination is not within two hops of the cluster-head, receiving the RREQ, it adds its address in the header and forwards the RREQ to all adjacent cluster-heads. If the destination is within two hops of a cluster-head, it sends the RREQ to the destination (Figure 2.18). The destination then inverts the route contained in the header of the RREQ and replies to the source with a RREP message using this reverse route (Figure 2.19). A route is hence, essentially, composed of a sequence of cluster-heads and the gateway nodes (between two clusters). When RREP passes through a cluster-head, it may optimize the route using its two-hop knowledge. If a route can be shortened, the cluster-head removes itself from the route by linking two member nodes together in the route.

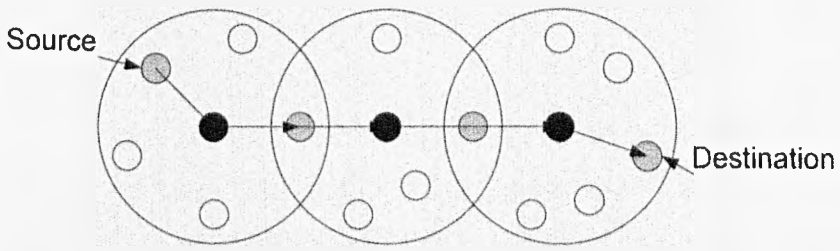


Figure 2.18: Route Request Packet (RREQ) in CBRP.

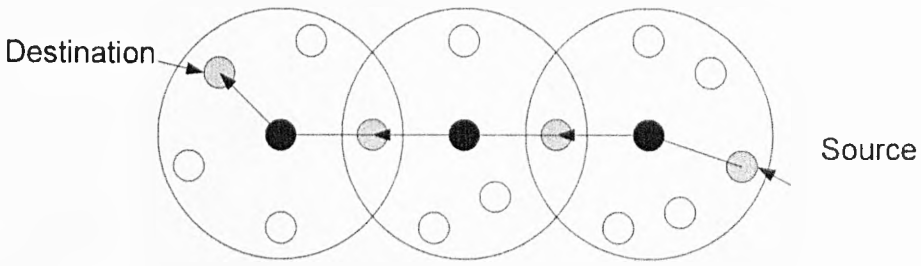


Figure 2.19: Route Reply Packet (RREP) in CBRP.

2.5.2 Clustering for Backbone Formation

The aim in this type of architecture is to reduce the number of hops, thereby diminishing the end-to-end delay and providing high speed connectivity between distant nodes in large networks. The most popular architecture is the Near-Term Digital Radio Network.

2.5.2.1 Near-Term Digital Radio Network

The Near-Term Digital Radio Network was developed by R. Ruppe, et al [25]. The NTDR produces a set of clusters, each of which contains a cluster-head. When they are linked together, they form a routing backbone, as shown in Figure 2.20. The main difference with the link-clustered architecture is that the communication between clusters is only realized by cluster-heads. The cluster formation in NTDR is as follows: an NTDR node elects itself as cluster-head if it does not detect any other cluster-head in its vicinity. The NTDR architecture restricts direct inter-cluster communication to cluster-heads, only thus permitting the cluster-heads to function as the gateways.

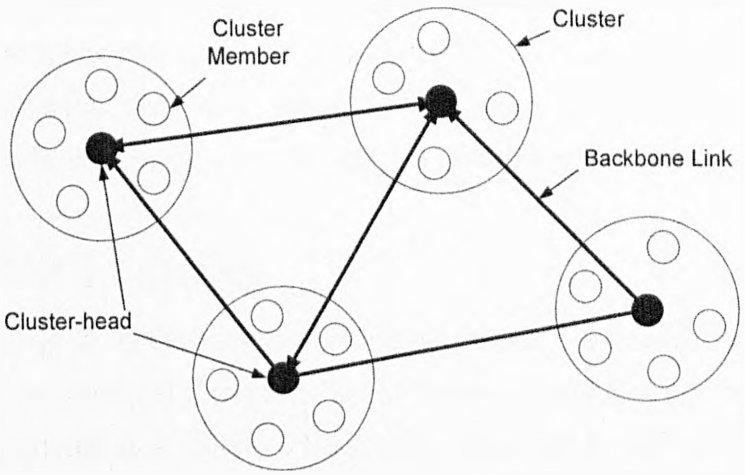


Figure 2.20: The NTDR Network Architecture.

2.6 Special Issues relating to inter-vehicle ad-hoc routing protocols

One of the most obvious issues relating to inter-vehicle ad-hoc routing protocols is the velocity of the mobile devices. One of the effects of velocity is to make the signals very variable. Chu and Stark demonstrated in [64] that the fading signals are in function of velocity. Based on simulations, they observed that the best performance is at lower velocities because the signals vary more slowly, thus creating little channel attenuation from bit to bit.

Another important aspect of the inter-vehicular ad-hoc routing protocols is the environment in which the vehicles are moving. It is known that different environments lead to different performances. For example, in an urban scenario, we will have more multi-path respect to a freeway scenario because of the presence of buildings and obstacles. Moreover, we must also consider the different velocity of different scenarios: in a freeway scenario, the vehicles will have a higher velocity and be more distant from each other than in a urban scenario; distance and relative velocity are very important because they influence the communication in a very significant manner. For example, in a urban scenario, the distances between vehicles are very small for a long periods of time, so they can exchange more data than in a freeway scenario, where the distances are greater. It is important to recall that in a peer to peer communication, the distance between peers must be small enough for the entire duration of the communication.

These facts lead us to consider the dynamic change of the traffic and important variables, including changes in the distance between vehicles, their relative velocity and their disposition as they move along to streets or roadways. Consequently, different models must be developed to predict vehicular movement in real world scenarios.

2.7 CONCLUSIONS

The challenge in wireless ad-hoc routing algorithms is the limited bandwidth and the high rate of geographical changes in highly mobile environments. In these environments, frequent broadcasting of routing information is necessary to maintain the routing tables updated. Limiting this distribution across the network is essential in the scalability of wireless ad-hoc routing algorithms. On one hand, proactive algorithms keep the information of all the nodes in the wireless ad-hoc network, requiring additional control traffic to continually update route entries. Consequently, proactive protocols suffer the disadvantage of requiring significantly greater of bandwidth resources. On the other hand, reactive algorithms reduce control traffic overhead at the cost of increased latency in finding the route to a destination. Existing pure proactive and reactive algorithms are not suitable in high mobility environments, as they result in poor route convergence and very low communication throughput. To minimize this limitation, positional algorithms have emerged add additional information in their routing information.

Nevertheless, positional algorithms are insufficient to cope with dense and large networks with high mobility if they do not consider any hierarchical or clustering combination. To date, the mechanism of sending information in cluster-based networks is inefficient because it constrains all traffic to traverse cluster-heads, thereby augmenting the possibility of congestion due to traffic concentration. Each cluster-head may become a single point of failure for communication across its cluster, causing the reduction of both throughput and robustness of the network.

CHAPTER 3: REACTIVE LOCATION-BASED ROUTING ALGORITHM WITH CLUSTER-BASED FLOODING

3.1 INTRODUCTION

The idea of using geographical information in highly mobile nodes is in order to improve packet forwarding decisions **[65]**. Before forwarding a packet, a node using location information only needs information about the immediate neighbourhood and the position of the destination. However, as an essential prerequisite for location-based routing, a location service is needed from which a node can learn the current position of its desired communication partner. To preserve location information on other nodes in the network, each mobile node maintains a location table. This table contains an entry for every node in the network whose location information is known, including the node's own location information. Three location services have been described in **[66]**: DREAM Location Service (DLS), Simple Location Service (SLS) and Reactive Location Service (RLS).

3.1.1 DREAM Location Service (DLS)

In the DREAM location service, each location packet updates the location tables containing the coordinates of the source node based on some reference system, the source node's position, and the time the location packet was transmitted. Each mobile node in the ad hoc network transmits a location packet to nearby nodes at a given rate and to faraway nodes at another, lower rate. The rate a mobile node transmits location packets adapts according to when the mobile node has moved a specified distance from its last update location. Since faraway nodes appear to move more slowly than nearby mobiles nodes, it is not necessary for a mobile node to maintain up-to-date location information on faraway nodes. Thus, by differentiating between nearby and faraway nodes, the overhead of location packets is limited.

3.1.2 Simple Location Service (SLS)

A node using the Simple Location Service will transmit a location packet to its neighbours at a given rate. The rate a mobile node transmits location packets adapts according to location change, via a similar procedure used for nearby nodes in DLS. Each location packet in SLS contains up to E entries from the node's location table and the E entries are chosen from the table in a round robin fashion. As multiple location packets are transmitted, all the location information a node knows will be shared with its neighbours.

A node using SLS will also periodically receive a location packet from one of its neighbours. The node will then update its location table, based on the received table entries, such that location information with the most recent time is maintained.

3.1.3 Reactive Location Service (RLS)

In the Reactive Location Service (RLS), when a mobile node requires a location for another node and the location information is either unknown or expired, the requesting node will first ask its neighbours for the requested location information. If the node's neighbours do not respond to the requested location information within a timeout period, then the node will flood a location request packet in the entire network.

When a node receives a location request packet and does not know the requested location information, the node propagates the flood location request. If, however, a node receives a location request packet and the node's location table contains the requested location information, the node returns a location reply packet via the reverse source route obtained in the location request packet. In other words, each location request packet carries the full route (a sequence list of nodes) that a location reply packet should be able to traverse in its header.

3.2 ROUTING OF PACKETS USING LOCATION INFORMATION

The advantage of using positional information in vehicular ad-hoc wireless networks (VANET) is that routing can be fully dynamic and distributed and the number of

flooding packets observed in dynamic ad-hoc networks compared to non-positional algorithms is reduced.

In non-positional based routing algorithms, link failures or link changes may lead to the transmission of route error packets. In this case, either local route error correction is required or the route error has to travel back to the source. The source then sends a fresh route request looking for a new route to the destination. If the nodes in the network are mobile with constantly changing links, the probability of these failures increases. This may lead to the source not being able to send data packets to the destination. However, with positional-based routing algorithms, routing can be done without the use of route error packets. A source node using geographic forwarding includes the location of its destination in each packet. The packet moves, hop by hop, through the network, forwarded along via cooperating intermediates nodes. At each node, a purely local decision is made to forward the packet to the neighbour that is geographically closest to the destination. The fact that forwarding does not involve any global information helps geographic routing scale and cope well with highly mobile nodes.

In this chapter, we propose a reactive location routing algorithm with cluster-based flooding for inter-vehicle communication (LORA-CBF) due to the increasing number of vehicles in circulation [66]. In 1950 the number of vehicles registered in United Sates was around 50 million and for 2000 it was around 150 million. This tendency motivates us to design an algorithm that copes with a very dense and highly mobile network.

3.3 REACTIVE LOCATION ROUTING ALGORITHM WITH CLUSTER-BASED FLOODING (LORA-CBF)

This algorithm inherits the properties of reactive and hierarchical routing algorithms and has the advantages of acquiring routing information only when a route is needed. First of all, LORA-CBF improves the traditional routing algorithms based on non-positional routing by making use of location information provided by GPS, and secondly, it minimizes flooding of its Location Request (LREQ) packets. Flooding, therefore, is directive for traffic control by using only the selected nodes, called gateway nodes, to diffuse LREQ messages. The purpose of gateway nodes is to minimize the flooding of broadcast messages in the network by reducing duplicate retransmission in the same region.

Member nodes are converted into gateways when they receive messages from more than one cluster-head. All the members in the cluster read and process the packet but do not retransmit the broadcast message. This technique significantly reduces the number of retransmissions in a flooding or broadcast procedure in dense networks. Therefore, only the gateway nodes retransmit packets between clusters (hierarchical organization). Moreover, gateways only retransmit a packet from one gateway to another in order to minimize unnecessary retransmissions, and only if the gateway belongs to a different cluster-head. To avoid synchronization of neighbours' transmissions, as observed in **[38] [41] [43]**, we have delayed each packet transmission randomly.

Some Cluster-Based Flooding strategies for routing in wireless ad-hoc networks have been reported in the literature **[68] [69] [70] [71] [72]**. The main contribution in this work is the re-broadcast and gateway selection mechanism. The cluster formation in **[68]** is based on the Link Cluster Algorithm (LCA) **[58]** and the algorithm is based on Link State Routing protocol, where all nodes in the cluster are expected to acknowledge the Link State Update (LSU). If one of the nodes does not send an acknowledgement, the cluster-head retransmits the LSU to that particular node. Flooding is transmitted from source to a destination via cluster-heads and gateways. In **[69]** and **[70]**, the re-broadcast is done only by the boundary nodes. Nodes other than boundary nodes just listen and update their tables. In **[71]**, there are two types of routing strategies: Optimal Spine Routing (OSR) and Partial-knowledge Spine Routing (PSR). OSR uses full and up-to-date knowledge of the network topology, thus the source can determine the route to the destination. On the other hand, PSR uses partial knowledge of the network topology, and takes a greedy approach to compute the shortest path from the source to the destination. In **[72]**, the authors use a cluster-head controlled token protocol (like polling) to allocate the channel among competing nodes. We have implemented a re-broadcast strategy, where only gateways that belong to a different cluster-head re-broadcast the location request packets, improving the routing overhead in dense networks.

Apart from normal Hello messages, LORA-CBF does not generate extra control traffic in response to link failures and additions. Thus, it is suitable for networks with high rates of geographical changes. As the protocol keeps only the location information

of the [source, destination] pairs in the network, the protocol is particularly suitable for large and dense networks with very high mobility. LORA-CBF is designed to work in a completely distributed manner and thus does not depend upon any central entity. In addition, it does not require a reliable transmission for its control messages. Each node sends its control messages periodically, and can therefore sustain some packet losses. This is, of course, important in radio networks like the one being considered here, where deep fades are possible.

LORA-CBF does not operate in a source routing manner **[30]**. Instead it performs hop by hop routing: each node uses its most recent location information of its neighbour nodes to route a packet. Hence, when a node is moving, its position should be registered in a routing table such that the movements can be predicted to correctly route the packets to the next hop to the destination.

3.3.1 Protocol functioning

The Reactive Location Routing Algorithm with Cluster-Based Flooding (LORA-CBF) carries out different functions that are required to perform the task of routing. Here we discuss some functionalities of the protocol.

3.3.1.1 Neighbour sensing

Each node must detect the neighbouring nodes with which it has a direct link. To accomplish this, each node periodically broadcasts a Hello message, containing its location information, address and its status. These control messages are transmitted in broadcast mode and are received by all one-hop neighbours, but they are not relayed to further nodes. A Hello message contains:

- Node Address.
- Type of node (Undecided, Member, Gateway or Cluster-head).
- Location (Latitude and Longitude).

3.3.1.2 Operation of Reactive Location Routing Algorithm with Cluster-Based Flooding (LORA-CBF)

LORA-CBF is formed with one cluster-head, zero or more members in every cluster and one or more gateways to communicate with other cluster-heads. Each cluster-head maintains a "Cluster Table." A "Cluster Table" is defined as a table that contains the addresses and geographic locations of the member and gateway nodes. We have assumed that all nodes can gather their positions via GPS or some local coordinate system.

When a source attempts to send data to a destination, it first checks its routing table to determine if it knows the location of the destination. If it does, it sends the packet to the closest neighbour to the destination (Figure 3.1). Otherwise, the source stores the data packet in its buffer, starts a timer and broadcasts Location Request (LREQ) packets. Only gateways and cluster-heads can retransmit the LREQ packet. Gateways only retransmit a packet from one gateway to another in order to minimize unnecessary retransmissions, and only if the gateway belongs to a different cluster-head.

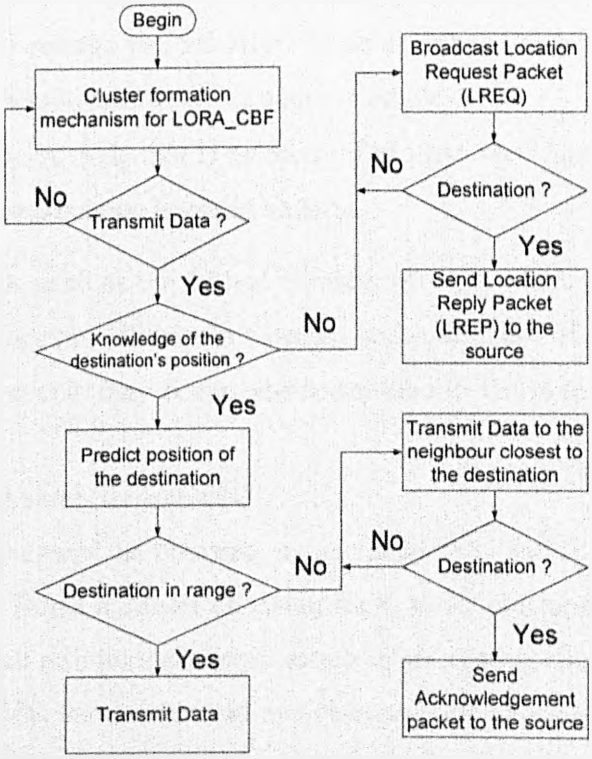


Figure 3.1: Flow diagram for LORA-CBF.

Upon receiving a location request, each cluster-head checks to see if the destination is a member of its cluster. Success triggers a Location Reply (LREP) packet that returns to the sender using geographic routing, because each node knows the position of the source and the closest neighbour based on the information from the LREQ received and the Simple Location Service (SLS). Failure triggers retransmissions by the cluster-head to adjacent cluster-heads (Reactive Location Service, RLS). The destination address is recorded in the packet. Cluster-heads and gateways, therefore, discard a request packet that they have already seen.

Once the source receives the location of the destination, it retrieves the data packet from its buffer and sends it to the closest neighbour to the destination.

A node in LORA-CBF can be in any of the following four states:

- Undecided: A node is in this transitional state when it is in search of a cluster-head. The node is in this state when it initially enters the network or wakes up.
- Member: A node that is a member of some cluster assigned to a cluster-head. A Member in LORA-CBF can not retransmit a Location Request Packet (LREQ).
- Cluster-head: A node that is responsible for all the nodes in its cluster and sends a Hello message periodically. The cluster-head also maintains the cluster table of the member and gateway nodes in its cluster.
- Gateway: A node that is member of at least two cluster-heads that can be used for communication between clusters.

For a network, such as the ad-hoc wireless network where there are not static nodes, there will be frequent link breaks between nodes, because of the dynamic nature of the network. Four events may occur which can lead to the make or break of links in the network:

1. A new node enters the network

When a node enters the network, it can follow two paths; it either joins an existing cluster-head or forms a cluster choosing itself as the cluster-head. Being in Undecided state, it sends out a Hello message in search of an existing cluster-head and if it receives a response, it joins the cluster-head and changes from undecided to member. Otherwise, it elects itself as the cluster-head and sends out future Hello message noting itself as a cluster-head.

2. An existing node leaves the network

When a node leaves the network due to failure or some other reason, the node is said to be turned off. In this case, the cluster-head will wait for a response from the failed node. If there is no response within a particular time period, the node is either considered off or has changed its location out of the range of its cluster-head. In this case, the cluster-head updates its cluster table removing the details of that node.

In case the node that has switched off is a cluster-head, this results in all the nodes of the cluster going to the undecided state. The nodes then search for a different cluster-head. In case they find a cluster-head, they change their status to a member of that cluster-head.

In case the node that has switched off is a bridge node between two existing cluster-heads, the cluster-head will update its Cluster Table when the cluster-head does not receive further messages from the gateway.

3. A node enters the vicinity of an existing cluster

When a node having an existing cluster-head enters the vicinity of another cluster-head, its status is that of a member. It changes its status to gateway before rebroadcasting the Hello message. In this case, when the cluster-heads receive the Hello message from the gateway, they simply update their routing table with the actual information.

4. A node leaves the vicinity of an existing cluster

In case the node moves out of the vicinity of an existing cluster *X* and makes its way to a new cluster *Y*, the cluster head of *X* updates its routing table so that the node is no longer its member and the cluster-head of *Y* updates its routing table, making the node its member. Transmitting Hello messages between cluster-head *Y* and the node does this. In case the node moves out of a cluster and cannot find a new cluster-head, it elects itself as the cluster-head.

Basically, the algorithm consists of four stages:

1. Cluster formation.
2. Location discovery (LREQ and LREP).
3. Routing of data packets.
4. Maintenance of location information.

3.3.1.3 Cluster formation

To enable cluster formation and maintenance, all nodes keep the information about their neighbours in the neighbour table.

Let *t* be the period of time between the Hello broadcasts. When a node first switches on, it first listens to Hello packets on the broadcast channel. If any other node on the broadcast channel is already advertising itself as a cluster-head (status of node = cluster-head), the new node saves the heard cluster-head ID in its cluster-head ID field and changes its status to member. At any point in time, a node in the mobile network associates itself with a cluster-head. The cluster-heads are identified by the cluster-head ID. Otherwise, the new node becomes cluster-head. The cluster-head is responsible for the cluster and periodically sending a Hello Message (Figure 3.2).

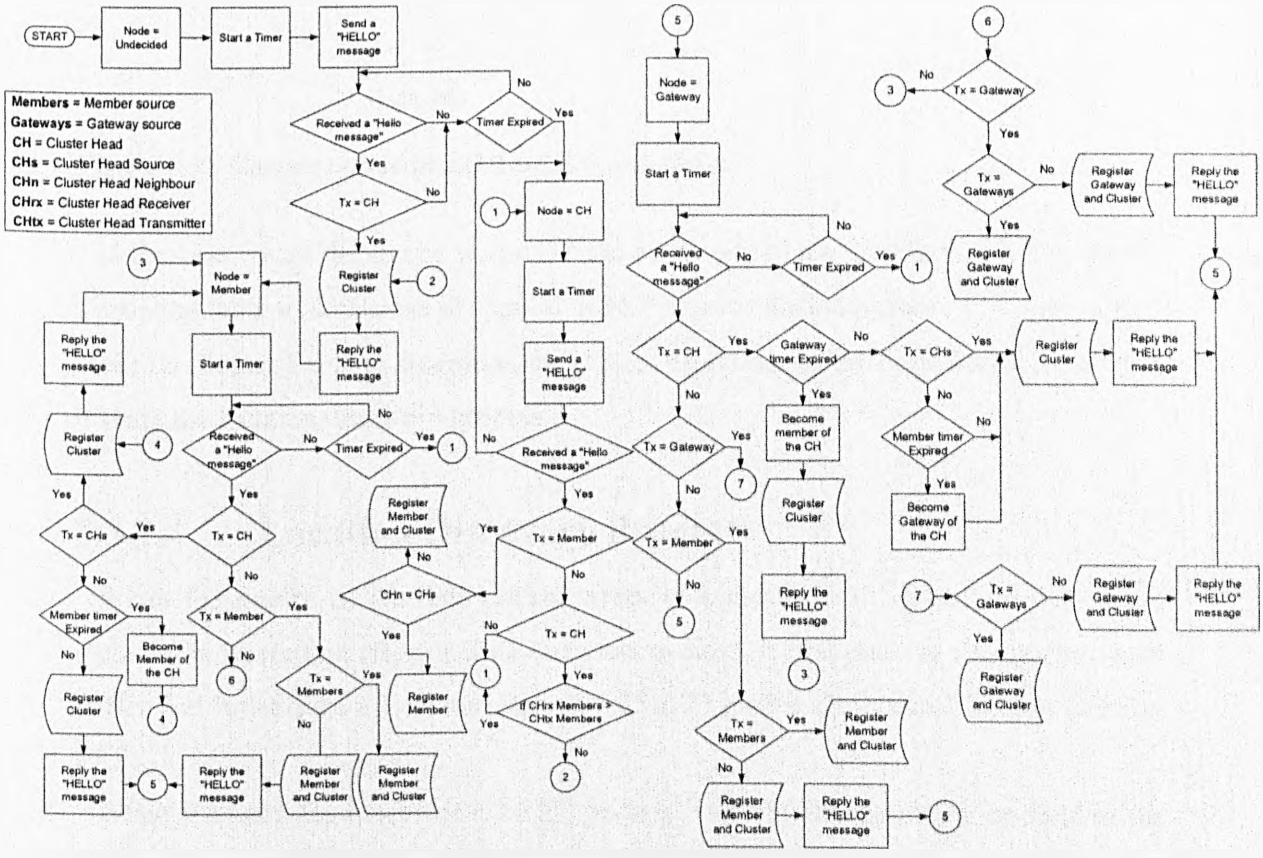


Figure 3.2: Cluster formation mechanism for Reactive Location Routing Algorithm with Cluster-Based Flooding (LORA-CBF).

When one member receives the Hello message, it registers the cluster-head and responds with a reply Hello message. The cluster-head then updates the Cluster Table with the address and position (longitude and latitude) of every member in the cluster.

When a member receives a Hello packet from a different cluster-head, it first registers the cluster-head, but the member does not modify its cluster-head ID until the expiration time for the field has expired. Before the member rebroadcasts the new information, it changes its status to a gateway. After receiving the Hello packet, the cluster-heads update the Cluster Table with the information about the new gateway (Figure 3.3).

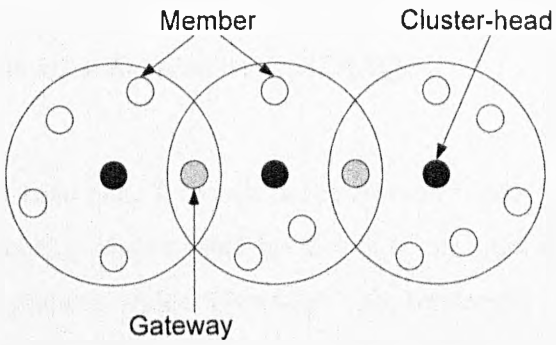


Figure 3.3: Gateway node in LORA-CBF algorithm.

In the case where the source wants to send a message to the destination, it first checks its routing table to determine if it has a "fresh" route to the destination. If it does, it first seeks its Cluster Table to determine the closest neighbour to the destination. Otherwise, it starts the location discovery process.

3.3.1.4 Location Discovery Process

When the source of the data packet wants to transmit to a destination that is not included in its routing table or if its route has expired, it first puts the data packet in its buffer and broadcasts a Location Request (LREQ) packet (Reactive Location Service, RLS).

When a cluster-head receives a LREQ packet, it checks the identification field of the packet to determine if it has previously seen the LREQ packet. If it has, it discards the packet. Otherwise, if the destination node is a member of the cluster-head, it unicasts the Location Reply (LREP) packet to the source node.

If the destination node is not a member of the cluster-head, it first records the address of the LREQ packet in the list and rebroadcasts the LREQ packet to its neighbouring Cluster-heads (Figure 3.4).

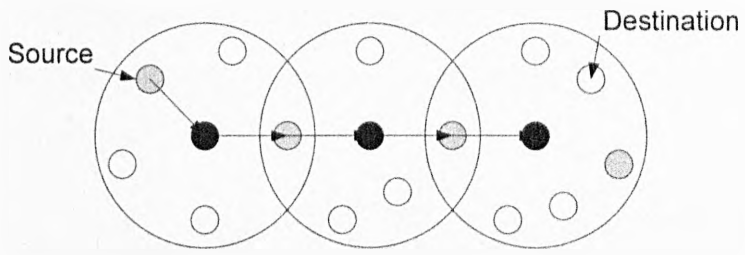


Figure 3.4: Location Request Packet (LREQ).

Each cluster-head node forwards the packet only once. The packets are broadcast only to the neighbouring cluster-head by means of an omni-directional antenna that routes them via the gateway nodes. Gateways only retransmit a packet from one gateway to another in order to minimize unnecessary retransmissions, and only if the gateway belongs to a different cluster-head. When the cluster-head destination receives the LREQ packet, it records the source address and location. From this, the destination's cluster-head knows the location of the source node. The destination then sends a LREP message back to the source via its closest neighbour (Figure 3.5).

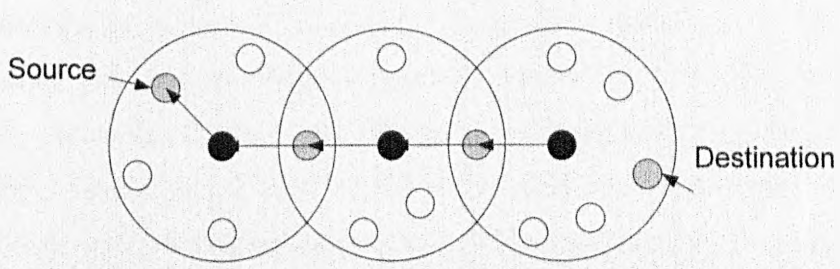


Figure 3.5: Location Reply Packet (LREP).

Finally, the packet reaches the source node that originated the request packet. If the source node does not receive any LREP after sending out a LREQ for a set period of time, it goes into an exponential back off before re-transmitting the LREQ. Hence, only one packet is transmitted back to the source node. The reply packet does not have to maintain a routing path from the source to the destination. The path is determined from

the location information given by the source node. The path traversed by the LREQ may be different from that traversed by the LREP.

3.3.1.5 Routing of Data packets

The actual routing of data packets is then based on the location of source, destination and neighbours (Figure 3.6).

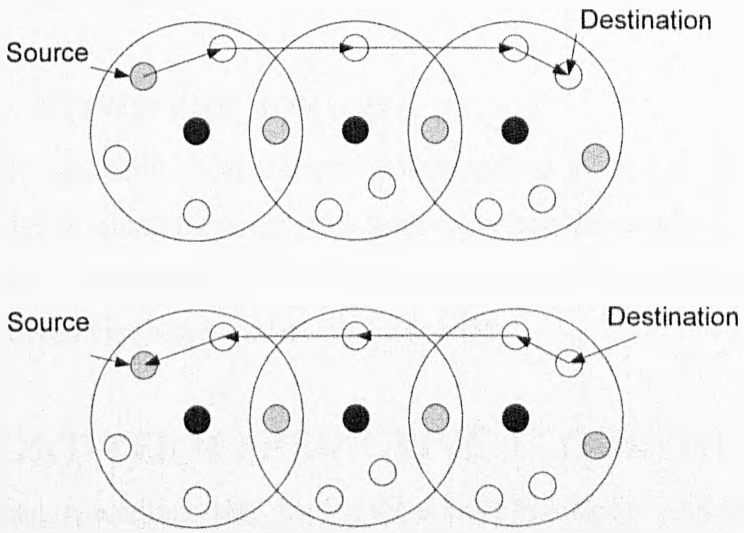


Figure 3.6: Data and Acknowledgement packets being forwarding in LORA-CBF.

Since the protocol is not based on source routing, packets travel the path from source to a destination based on locations. The packets find paths to the destinations individually each time they transmit data between the source and the destination. As the two nodes know their relative positions, packets are transmitted based on this. Moreover, since the transmission is in the direction of the destination node, the path found will be shorter than other routing mechanisms (non-positional-based). In non-positional-based routing strategies, the shortest path is measured in hops. Therefore, the path found may not be the shortest. However, the path found using location information will be significantly shorter. If the source of the data packet does not receive the acknowledgement packet before its timer expires, it will retransmit the data packet again. This situation might occur during loss of packets due to drop out or network disconnection.

3.3.1.6 Maintenance of location information

The LORA-CBF algorithm is suitable for networks with very fast mobile nodes because it maintains and updates the location information of the source and destination every time the pairs send or receive data and acknowledgment packets. The source updates its location information before sending each data packet. When the destination receives the data packet, its location information is updated and an acknowledgment packet is sent to the source.

3.3.1.7 Forwarding strategy

LORA-CBF uses MFR (Most forward within radius) as its forwarding strategy. In MFR the packet is sent to the neighbours that best reduce the distance to the destination. The advantage of this method is that it decreases the probability of collision and end-to-end delay between the source and the destination **[59]**.

3.4 SHORT-TERM PREDICTIVE ALGORITHM

In highly mobile environments, having the correct knowledge of neighbour positions is fundamental in the routing efficiency of any algorithm. LORA-CBF predicts the next position (geographical location) of every neighbour node, based on its short-term predictive algorithm **[73]**. After predicting the position of all neighbour nodes, LORA-CBF sends the packet to the neighbour node which has the best position (MFR). The best position means that it can reach the node that is closest to the destination.

Mobility and contention of the wireless media may cause the loss of packets being transferred, and this is a very important aspect to consider in the development of predictive algorithms. We address this problem, including the gap between packets being received (Figure 3.7).

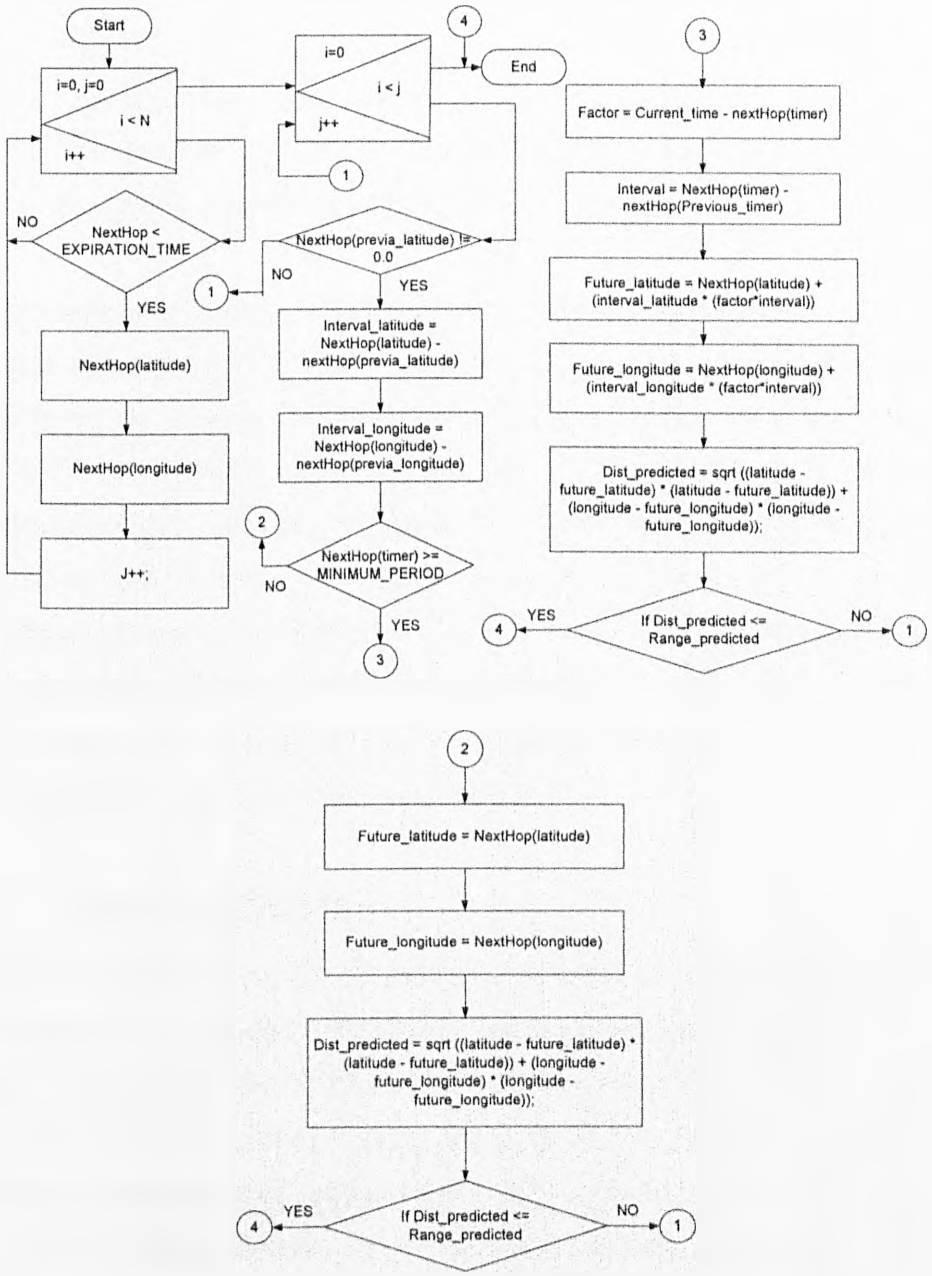


Figure 3.7: Sort-term predictive algorithm used in LORA-CBF.

The short-term predictive algorithm tries to extrapolate the position of the next hop k positions ahead in time. For example, a simple technique is to assume the data follows a linear trend.

$P_{j+k} = P_j + \Delta P * e$ ..... (3.1)

Where:

P_{j+k} future position of the next hop.

P_j current position of the next hop.

ΔP interval between current position and previous position of the next hop.

e factor indicating the gap between packets received.

The predictive algorithm is useful for networks with very high mobility and which are contention-based. In LORA-CBF, a Hello message is broadcast periodically. Every node keeps the location information of every neighbour from which it has received Hello messages. When one node receives a packet for transmission to a particular destination, it first checks its routing table to determine if it knows the location of the destination node. If it does, it triggers the short-term predictive algorithm to calculate the future position of the destination. If the node can reach the destination, it sends the packet directly. Otherwise, before retransmitting it, the node predicts the locations of the neighbour nodes, based on previous positions, and sends the packet to the closest neighbouring node to the predicted destination.

3.5 CONCLUSIONS

We have addressed several challenges in vehicular ad-hoc networks (VANET) with the Reactive Location-Based Routing Algorithm with Cluster-Based Flooding (LORA-CBF). First, using location information, we have improved packet forwarding decisions due to two location services: Simple and Reactive. For neighbouring nodes, a Simple Location Service has been implemented, and for faraway nodes, a Reactive Location Service is employed. Second, in geographic forwarding, the source node includes the location of its destination in each packet. Here, the packet moves hop by hop through the network, forwarded along via cooperating intermediates nodes. At each node, a purely local decision is made to forward the packet to the neighbour that is geographically closest to the destination. However, location information by itself does not guarantee the transmission between neighbouring nodes in vehicular ad-hoc networks. Mobility and contention of wireless media may cause loss of packets being transferred, and this is very important aspect to consider in the development of wireless routing algorithms. Here, we have addressed this problem by including a predictive algorithm in LORA-CBF. Finally, due to the increasing number of vehicles, VANET

networks are foreseen to include hundreds or thousands of vehicles in its architecture. This tendency motives us to design an algorithm that copes with a very dense and highly mobile network. What we have developed here divides the size of VANET networks into several clusters, thus improving its scalability.

CHAPTER 4: JUSTIFICATION OF IEEE 802.11b WIRELESS NETWORKS FOR INTER-VEHICLE COMMUNICATION

4.1 INTRODUCTION

There are many problems that affect communication in mobile communication environments, but the two main ones are: multi-path delay and Doppler Shift [74]. Multi-path delay produces frequency selective fading, so that signals suffer interference, and Doppler can affect shift the carrier frequencies. Fading is caused by interference between two or more versions of the transmitted signal that arrive at the receiver at slightly different times. The versions, also called multi-path waves, combine at the receiver antenna to give a resultant signal, which can vary widely in amplitude and phase, depending on the distribution of the intensity and relative propagation time of the waves and the bandwidth of the transmitted signal. The phenomenon is known as constructive or destructive interference.

In the estimation of the radio coverage area of a transmitter and receiver, two simple large-scale and small-scale propagation models can be used. Large-scale models characterise signal strength over large T-R separation distances (several hundreds or thousands of meters). Propagation models that characterise the rapid fluctuation of the received signal strength over very short travel distances (a few wavelengths) or short time duration (on the order of seconds) are called small scale, or fading models.

4.2 LARGE-SCALE FADING

As a mobile node moves away from the transmitter over much larger distances, the local average received signal will gradually decrease, and it is the local average signal level that is predicted by large-scale propagation models.

4.2.1 Free Space Propagation Model

The free space propagation model (FSP) is used to predict received signal strength when the transmitter and receiver have a clear, unobstructed line-of-sight path between them **[74]**. The FSP model can be calculated with the formula listed below, which is the transmission between two antennas, separated by a distance.

$$P_r(d) = \frac{PtGtGr\lambda^2}{(4\pi)^2 d^2} \tag{4.1}$$

Where:

Pt is the transmitted power, $P_r(d)$ is the receiver power, which is a function of the transmission – reception separation.

Gt is the transmitter antenna gain, Gr is the receiver antenna gain, d is the transmission – reception separation distance in meters and λ is the wavelength in meters.

Received power $P_r(d)$ is generally the most important parameter predicted by large-scale propagation models

4.2.2 Path Loss

Path Loss is the reciprocal of the received power and represents signal attenuation as a positive quantity measured in dB, and is defined as the difference (in dB) between the effective transmitted power and the received power, which may or may not include the effect of antenna gain. The path loss for the free space model when antenna gains are included is given by the equation below.

$$\frac{P_r(d)}{P_t} = \frac{G_t G_r \lambda^2}{(4\pi)^2 d^2} \qquad 10\log\frac{P_t}{P_r} = -10\log\left[\frac{G_t G_r \lambda^2}{(4\pi)^2 d^2}\right]$$

$$PL(dB) = -10\log\left[\frac{G_t G_r \lambda^2}{(4\pi)^2 d^2}\right] \tag{4.2}$$

4.2.3 System Operating Margin

System Operating Margin (SOM) (also referred to as Fade Margin) is defined as the difference between the received signal level (in dBm) and the receiver sensitivity (in dBm) needed for error free reception. Also, the System Operating Margin can be calculated using the formula listed below. SOM, basically, is the difference between the signals a radio is actually receiving vs. what it needs for good data recovery (i.e. receiver sensitivity).

$$SOM = \mathrm{Re}ceived\_Signal(dBm) - \mathrm{Re}ceiver\_Sensitivity(dBm) \quad (4.3)$$

The System Operating Margin predicts the area of optimal reception between the transmitter and receiver. The minimum SOM recommended is 10 dB, and 20 dB is considered to be excellent **[76]**.

4.3 SMALL-SCALE FADING

As a mobile node moves over very small distances, the instantaneous received signal strength may oscillate rapidly giving rise to small scale fading. Small-scale fading, or simple fading, is used to describe the rapid fluctuations of the amplitude, phase or multi-path delay of a radio signal over a short period of time or travel distance, so that large-scale path loss effects may be ignored. In vehicular ad-hoc wireless networks (VANET), due to the relative motion between the transmitter and receiver, each multi-path wave experiences an apparent shift in frequency.

4.3.1 Impact of Doppler Shift

We have considered the worst case scenario to evaluate the impact of Doppler shift (Figure 4.1). In the scenario shown, we have assumed an average speed of the vehicles of 42 m/s (150 km/h), with each vehicle iequipped with an IEEE 802.11b wireless card. We wish to determine the maximum speed that the vehicle can travel without being affected by Doppler shift. The relative speed in the scenario shown in Figure 4.1 is the involvement of speed of each vehicle. In this case, it is 84 m/s.

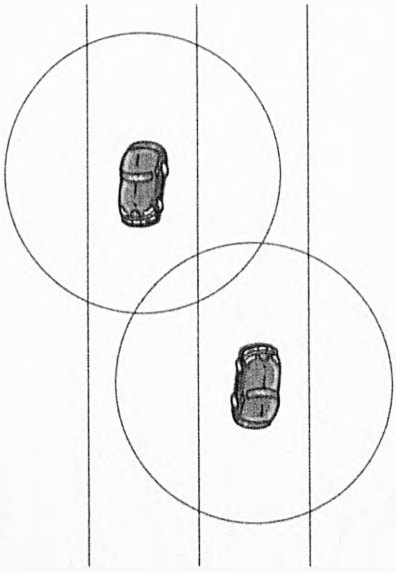


Figure 4.1: Worst case scenario to evaluate the impact of Doppler shift

There are two types of small-scale fading based on Doppler Spread: fast fading and slow fading.

4.3.1.1 Fast fading

Depending on how rapidly the transmitted base band signal changes compared to the rate of change of the channel, a channel may be classified either as a fast fading or slow fading channel. Therefore, a signal undergoes fast fading if $T_S > T_C$ and $B_S < B_D$.

Where: T_S is the reciprocal bandwidth, T_C is the coherence time, B_S is the Bandwidth, and B_D is the Doppler Spread.

The coherence time describes the time varying nature of the channel in a small-scale region and is caused by the relative motion between the vehicles.

Here, we test if our scenario is fast fading or slow fading. The signal base band in IEEE 802.11b is 1.375 MHz, so T_S = 90 ns.

$$\lambda = \frac{c}{f}, \qquad \lambda = \frac{3x10^8 m/s}{2.4x10^9 Hz} = 0.125 \frac{m/s}{Hz}, \qquad f_m = \frac{v}{\lambda} = \frac{42m/s}{0.125\frac{m/s}{Hz}} = 336\text{Hz}$$

The coherence time is defined in **[74]**, as the period of time over which the time correlation function is greater than 0.5,

$$T_C = \frac{0.423}{f_m} \qquad (4.4)$$

Where f_m is the maximum Doppler shift.

Using the equation (4.4), we obtain: $T_C = \frac{0.423}{336Hz} = 1.3\text{ms}$

$T_S = 90$ ns < 1.3ms = T_C, and B_S =1.375MHz > 336Hz = B_D. This is not a Fast fading channel.

4.3.1.2 Slow fading

In a slow fading channel, the channel may be assumed to be static over one or several reciprocal bandwidth intervals. In the frequency domain, this implies that the Doppler spread of the channel is much less than the bandwidth of the base band signals. Therefore, a signal undergoes slow fading if

$T_S << T_C$ and $B_S >> B_D$.

It should be clear that the velocity of the mobile node (or velocity of objects in the channel) and the base band signalling determines whether a signal undergoes fast fading or slow fading.

The channel in our scenario is slow fading because:

$T_S = 90$ ns << 1.3 ms = T_C and B_S = 1.375MHz >> 336Hz = B_D.

If the base band signal bandwidth is much greater than B_D, the effect of Doppler Spread are negligible at the receiver **[74]**.

Now, we are able to analytically determine the maximum speed that the vehicle can travel before it will be affected by Doppler Effect. Considering the maximum Doppler Spread of 1.375 MHz we obtain:

$v = f_m * \lambda$, $v = 1.375MHz * 0.125\frac{m/s}{Hz}$ v=618,750 km/h.

The result obtained indicates that the Doppler Effect will not affect the communication between vehicles, using the IEEE 80.11b Wireless cards.

4.4 TEST SET UP AND EXPERIMENTAL DETAILS

Very few test-beds have been deployed to evaluate the performance of wireless networks for inter-vehicle communication **[77] [78] [79]**. The authors in **[77]** used ORINOCO IEEE 802.11b WLAN cards and enhanced the range of connectivity by deploying ORINOCO omni-directional antennas on top of cars. One laptop was set up as a receiver and the other as a transmitter that streamed UDP packets. Three environments were evaluated: Sub-urban, Urban and Freeway. In the freeway environment, the speed limit was considered to be 65 miles per hour (104 km/h). Here the vehicle measurements were done considering the vehicles following and crossing each other. Authors in **[78]** have reported that, eight nodes were deployed within a 700m by 300 m site. Each node ran the Dynamic Source Routing (DSR) protocol. Every vehicle was equipped with a Lucent Wave LAN Wireless LAN radio on the roof. The ad-hoc network included five moving car-mounted nodes. In this experiment, one car was following the other at a separation of 90m. Another experiment was reported in **[79]**. Here the authors report a single inter-vehicle communication system using commercially available DS/SS wireless LAN modems with omni-directional antennas. The communication frequency is the 2.4 GHz ISM (Industrial, Scientific and Medical) band. They showed that inter-vehicle communication can be realized at low cost with existing equipment.

The experiment was realized between two vehicles. In one vehicle, a notebook computer (PC-1) was connected to a JRL 200 wireless LAN adapter via an Ethernet hub, to which an Internet camera was also connected. In the other vehicle, another notebook computer (PC-2) was connected directly to another JRL 200 wireless LAN adapter, both via a PCMCIA LAN card. Both computers and the camera were assigned an IP address. Thus a small LAN was formed, with one of the links being a wireless one between the two vehicles.

By using the UNIX command *ping*, the packet loss and round trip delay of packets transferred between computers was measured every two seconds. In addition, using the file transfer command ftp, the data rate of the transfer of some files was measured. Furthermore, images could be viewed in real time (approximately 1 picture per second) from the Internet camera on the computer PC-2.

Several experiments were undertaken on a straight stretch of road to determine if there was any effect of speed, and also whether there was any noticeable effect of Doppler shift when the vehicles approached and then moved away from each other at different speed. In all cases, good line of sight communication was achieved, and there was no noticeable performance degradation.

In our experiment, we were interested in the worst case scenario, where vehicles travel in opposite directions on opposite sides of a motorway.

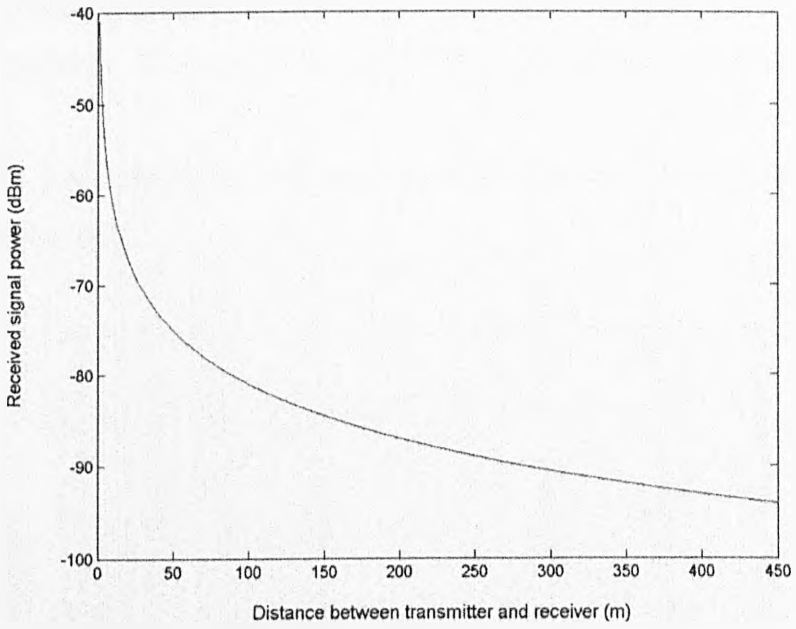
The first part of the experiment focused on determining the maximum distance of the received power between the transmitter and the receiver. To do this, we used two Enterasys' wireless cards and two omni-directional antennas. According to specifications, the antenna has a power of 15 dBm or 32 mW, and the omni-directional antennas have a gain of 5 dBi. We realized the experiment in the local airport of Colima, Mexico and repeated the test three times.

Table 4.1 indicates the values of receiver sensitivity for the Enterasys' wireless cards, according the data rates and distance between transmitter and receiver **[75]**.

	11 Mbps	5.5 Mbps	2 Mbps	1 Mbps
Distance	160	270	400	550
Receiver Sensitivity (dBm)	-82	-87	-91	-94

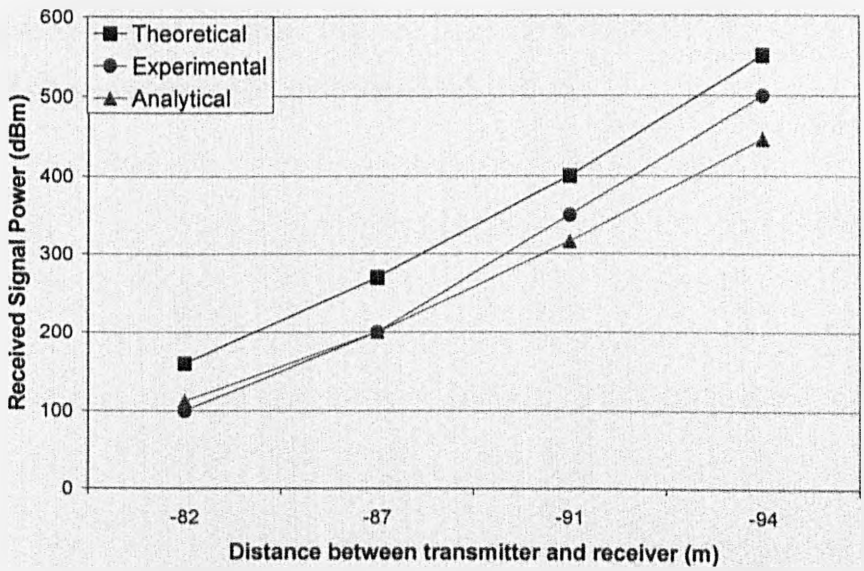
Table 4.1: Theoretical values specified for receiver sensitivity in Enterasys' wireless cards

Graph 4.1 represents the analytical result for the received signal power, and is achieved using equation (4.1).



Graph 4.1: Received signal power.

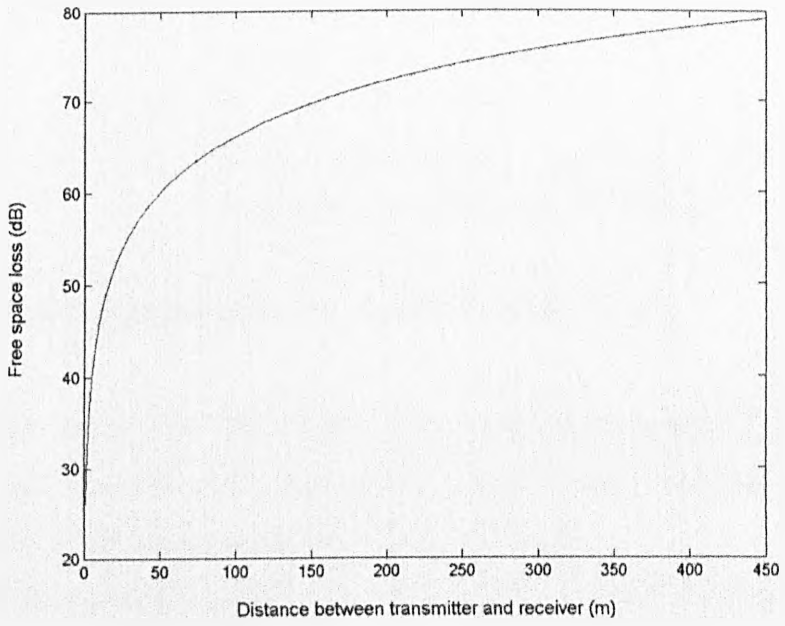
Graph 4.2 indicates theoretical, experimental and analytical results of the received signal power over a certain distance between the transmitter and receiver. The values expressed are the values shown in the data sheet of the Enterasys' Wireless cards. On the other hand, the values obtained experimentally are the same values we have used to obtain the analytical results.



Graph 4.2: Received signal power between transmitter and receiver

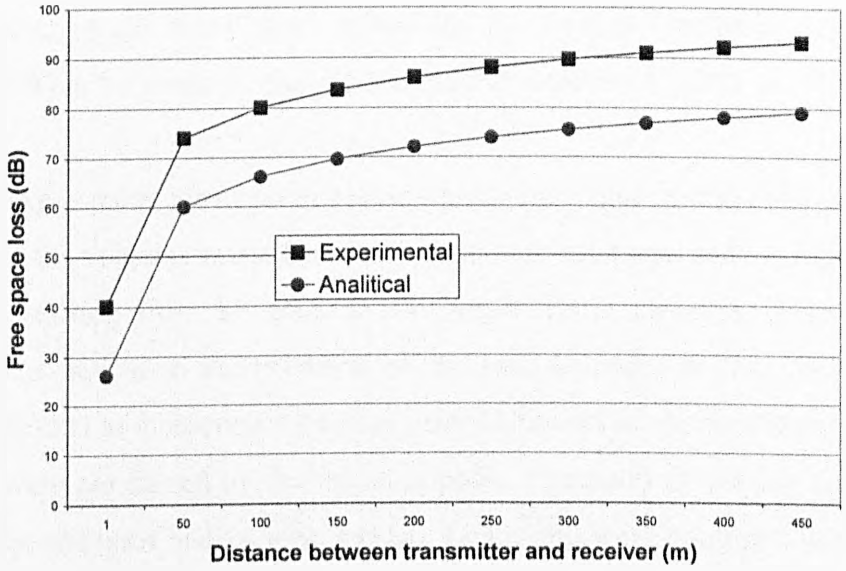
The maximum experimental distance between the transmitter and the receiver with 802.11b Enterasys' Wireless Cards and 5dBi car-mounted omni-directional antennas is 446 m.

Graph 4.3 represents the analytical result for the free space loss, and is achieved using equation (4.2).



Graph 4.3: Free space loss.

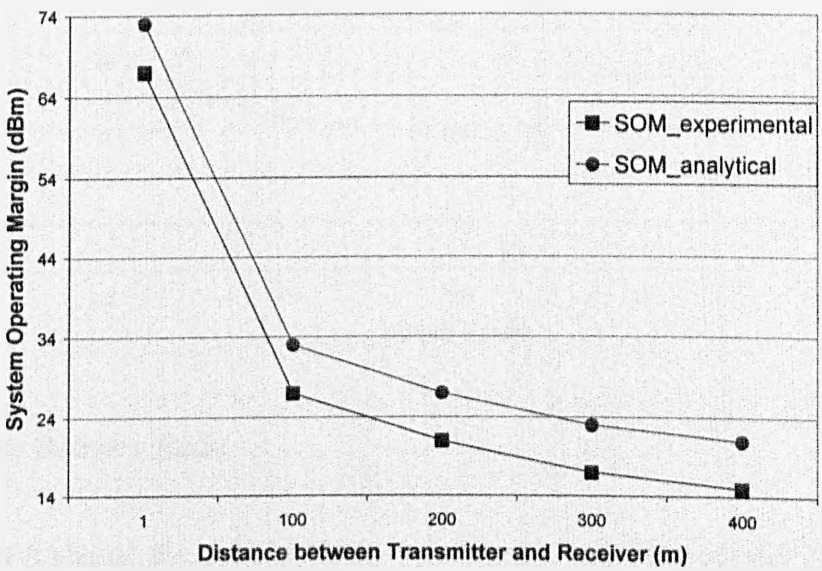
Graph 4.4 indicates experimental and analytical results of the free space loss over a certain distance between the transmitter and receiver.



Graph 4.4: Free space loss between transmitter and receiver.

Graph 4.4 shows the free space loss considering car-mounted omni-directional antennas and Enterasys' Wireless cards. The path loss increases with the distance starting with 40 dB at one meter to 93 dB at 450 meters.

The next experiment focused on determining the System Operating Margin between transmitter and receiver (Graph 4.5). The analytical result is achieved using equation (4.3).



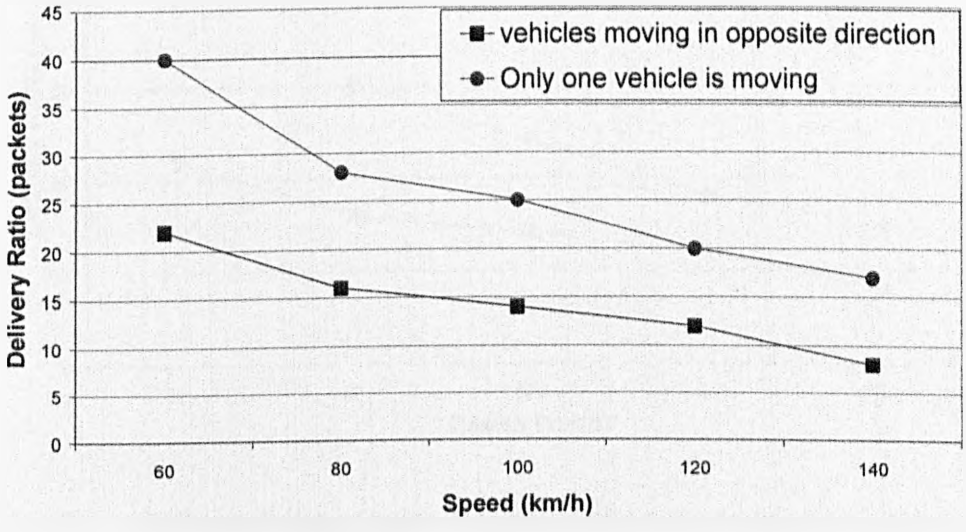
Graph 4.5: System Operating Margin.

Experimental results show good values for the System Operating Margin, with a distance of 300 m between the transmitter and the receiver. A SOM of 17dBm has been obtained.

The next experiment consisted in sending Hello messages in the worst case scenario. The speed of the vehicles was maintained constant in each test, and we repeated the test three times starting from 80 km/h to 140 km/h. Hello messages were periodically transmitted to announce the presence of the mobile node. In our algorithm, Hello messages are vital to disseminate location information between neighbouring nodes.

The tests were conducted by driving in opposite directions at varying speeds (Figure 4.1). The two vehicles had laptops running Linux and were equipped with Enterasys' IEEE 802.11b WLAN cards. The range of connectivity was enhanced by deploying an omni-directional antenna inside of each car.

One laptop was set up as a receiver and the other as a sender that streamed UDP packets. The wireless cards were configured to operate in broadcast ad-hoc mode. The UDP packets were of 64 bytes in length.



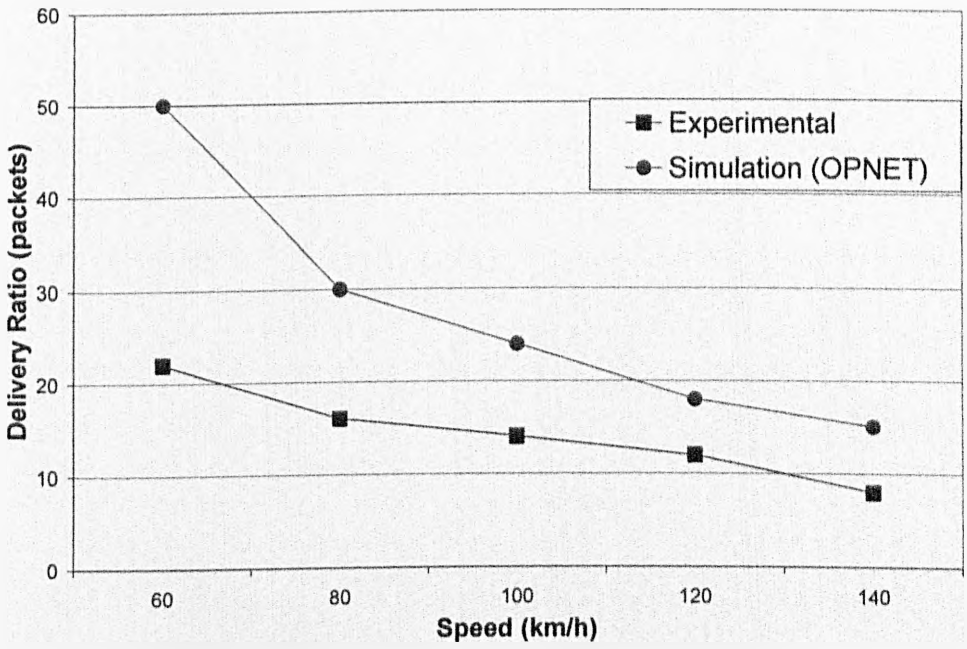
Graph 4.6: Delivery Ratio.

Graph 4.6 shows the delivery ratio between the transmitting and receiving vehicle (according to the transmission range). When one of the vehicles is moving at 140 km/h, the number of packets being received by the receiver vehicle is 17 and in the worst case

scenario, when both vehicles are travelling in opposite directions, the number of packets being received by the receiver vehicle is 8.

Graph 4.7 shows the results for delivery ratio using OPNET for simulation of the worst scenario and compares the results with those obtained experimentally. Our results are slightly different from the OPNET network simulator, because our omni-directional antennas were mounted inside of the cars, instead of on the roof of each. Our pigtail cable was too small to extend it more than 1m. Similar results are reported in **[79]**. Here, the effect of the antenna position was investigated. They found that the antenna mounted on the roof gave better results when mounted on the dashboard. When both antennas were mounted on the front dashboard, communication was more difficult.

Similar results are reported in **[80]**, where vehicles traveling in opposite directions at 140 km/h are in communication range for 12.5 seconds.



Graph 4.7: ***Delivery Ratio compared between the experimental result and results obtained in OPNET.***

4.5 CONCLUSIONS

In this chapter, we have shown that IEEE 802.11b wireless networks are suitable for inter-vehicle communication. We have supported our hypothesis with the results of two propagation models. On one hand, according to large scale models, the maximum distance between the transmitter and the receiver is 446 m. In addition, the System Operating Margin (SOM) feasible at 446 m is over 13 dB. The SOM is over the minimum margin recommended. On the other hand, we have found with the results of small scale models, that the Doppler Effect does not affect the communication between communication partners at high speed. Finally, we have realized an experiment that has allowed us to validate the former results in the worst case scenario, when the transmitter and receiver are travelling in opposite directions. Results have shown that at least 8 packets are possible when the transmitter and receiver are within communication range.

Chapter 5: Microscopic Traffic Models

5.1 INTRODUCTION

This chapter focuses on vehicular traffic models to be used with the previously described algorithm. These models will simulate realistic traffic movement and assess the most suitable traffic model for describing the movement of vehicles on a motorway.

According to **[81, 82]**, vehicular traffic models may be categorised into four level-of-detail classifications, namely sub-microscopic, microscopic, mesoscopic and macroscopic. Sub-microscopic models describe the characteristics of individual vehicles in traffic flow and the operation of specific parts (sub-units) of the vehicle (e.g. changing gears, breaks, etc). Microscopic models simulate driver behaviour and interaction among drivers. Mesoscopic models represent the transportation systems and analyse groups of drivers with homogeneous behaviour. Finally, macroscopic models describe traffic at a high level of aggregation as a flow without distinguishing its basic parts **[83] [84] [85] [86] [87] [88]**, such as in the situation of the traffic flow in a road network using entities such as density, flow and average speed. For simulation of large road networks, the family of macroscopic flow models is the common choice. This is because microscopic models are more often used for studying the traffic flow in smaller areas in great detail, because this type of model consists of sub-models that describe particular driver behaviour. Important behaviour models include: *gap-acceptance, speed adaptation, lane changing,, overtaking, and car-following.* The gap-acceptance model determines minimum acceptable distances from surrounding vehicles in the context of intersections and merging. Speed adaptation refers to the adaptation to the road design speed at a vehicle's current position in the network. Lane-changing models describe driver behaviour when deciding whether to change lane or not on a multi-lane road link, e.g. when travelling on a motorway. Analogously, on two-lane rural roads the overtake model controls drivers' overtaking behaviour. Finally, there is the car-following model, which describes the interactions with preceding vehicles in the same lane. Of these, this thesis concentrates on car-following. Modelling of car-following is needed when interactions between individual vehicles are taken into account. Most previous research

on driving behaviour modelling has been focused on car-following, and there are numerous papers on this topic. However, very few qualitative comparisons and descriptions of car-following models have been made **[89]**.Therefore, only microscopic traffic models are of interest within the context of this work. In addition, car following models are used in cases where one is interested in the dynamics of the traffic system or if information of microscopic traffic measures is needed. Typical applications of a microscopic traffic model include analysis of the impacts of different network designs, evaluation of Intelligent Transportation Systems (ITS) applications and emission modelling, all of which require detailed information on driving course of events.

5.2 INTRODUCTION TO VEHICULAR TRAFFIC THEORY

In vehicular traffic theory **[90]**, freeway traffic is described by three elementary parameters: *traffic density* ρ_{veh} in [veh/km], *traffic flow* q in [veh/s] and the *net time gap* τ in [s]. These quantities can be related together by their average values as shown in equation (5.1):

$$d_m = \frac{300}{\rho_{veh}} - l_m \qquad \tau_m = \frac{d_m}{v_m} = \frac{1}{v_m} \cdot \left(\frac{300}{\rho_{veh}} - l_m \right) \qquad q = \frac{1}{\tau_m} = v_m \cdot \left(\frac{1}{\frac{300}{\rho_{veh}} - l_m} \right) \tag{5.1}$$

Herein l_m is the average length of vehicles, d_m the average distance between vehicles and v_m the average speed in [m/s] of vehicles. From equation (5.1), the values are calculated to the maximum transmission range of 300 m. Also, we can calculate a possible maximum velocity v_p, if the average time gap τ_m and the traffic density ρ_{veh} are given. Real average velocities, the so-called average free velocities $v_{m,free}$, are always equal or below this limit:

$$v_{m,free} \le v_p = \frac{1}{\tau_m} \cdot \left(\frac{300}{\rho_{veh}} - l_m \right) \tag{5.2}$$

If traffic density is low, vehicles are assumed to drive at their free velocity $v_{m,free}$. Additional vehicles do not diminish the average driven velocity $v_{m,free}$, but only v_p. Thus, additional vehicles result in an increase of the traffic flow q and shorter time gaps τ_m.

This traffic state is called undisturbed traffic. If traffic density increases, so that it is not longer possible to drive by $v_{m,free}$, the driven velocity will reduce to v_p and the traffic state is called disturbed traffic.

Macro-simulation models describe the traffic flow in a road network using variables such as density, flow and average speed, whereas micro-simulation models consider each individual driver vehicle unit and its interactions with other vehicles in the network.

5.3 CAR FOLLOWING MODELS

All microscopic traffic models use a very large number of variables that have to be calibrated in order to achieve representative results. The calibration work increases proportionally with the number of parameters. It is therefore desirable to keep the number of variables as small as possible. A car-following model must however contain parameters that allow both the reaction time and the magnitude of the reaction to be adjusted.

A car-following model represents driver behaviour with respect to the preceding vehicle in the same lane **[91]**. We will now discuss three types of car-following models, namely safe-distance models, stimulus-response models and psycho-physiological models.

5.3.1 Safe-distance models

Safe-distance car-following models describe the mobility of a single vehicle in relation to its predecessor, and they are based on the assumption that the follower always keeps a safe distance to the vehicle in front. A simple model can be established allowing for at least one car length for every ten miles an hour (16.1 km/hr). Using this driving rule, is possible determine the minimum distance headway D of the vehicle n driving with velocity v with respect to vehicle $n-1$, where L_n represents the length of vehicle n.

$$D_n(v) = L_n(1+(v/16.1)) \qquad (5.3)$$

Equation (5.3) is called Pipe's model and shows that the minimal safe distance increases linearly with the velocity v of the vehicle. There is a more refined model that

describes the spacing of vehicles in traffic flow, in which the overall reaction time T consists of:

- *Perception time* (the time needed for the driver to recognise that there is an obstacle);
- *Decision time* (the time needed to make decision to decelerate), and
- *Braking time* (the time needed to apply the brakes).

The total *safety distance model* assumes that the drivers consider braking distances sufficient enough to permit them to brake to a stop without causing a rear-end collision with the preceding vehicles if the latter vehicles come to stop instantaneously. The corresponding safe distance headway equals:

$$D_n(v) = L_n + Tv + v^2/(2\mu g) \qquad (5.4)$$

where:

μ Denotes the friction with the road surface, and

g describes the acceleration due to gravity.

Finally, the gross-distance headway $GD_n(v)$ effectively occupied by vehicle n driving with velocity v is a function of the vehicle's length L_n, a constant minimal distance between the vehicles $d_{\min}$, the reaction time T and a speed risk factor F:

$$GD_n(v) = (L_n + d_{\min}) + v(T + vF) \qquad (5.5)$$

5.3.2 Stimulus-response car-following models

A second model is based on the fact that drivers try to follow the behaviour of the vehicle just ahead **[92]**. This model is based in the following principle:

$$Response = sensitivity \times stimulus \qquad (5.6)$$

In general, the response is the braking or the acceleration of the following vehicle, delayed by an overall reaction time T:

$$a_n(t+T) = \gamma(v_{n-1}(t) - v_n(t)) \qquad (5.7)$$

where:

$v_n(t)$ denotes the velocity of the vehicle n at time t,

$a_n(t)$ describes the acceleration of the vehicle n at time t,

γ defines the driver's sensitivity.

Thus, the stimulus is defined by the velocity difference between leader and follower. The following expression has been proposed for driver's sensitivity γ.

$$\gamma = c \cdot (v_n(t+T))^m / (x_{n-1}(t) - x_n(t))^l \qquad (5.8)$$

In equation (5.8), we see that the following vehicle adjusts its velocity $v_n(t)$ in proportion to both distance and speed differences with delay T. The extent to which this occurs depends on the training values of c, l and m which are calibrated during the simulation or by measurement.

5.3.3 Psycho-Physiological Car Following Models

In this model the reaction thresholds that distinguish different regions of driver behaviour have been considered. Here in x space, the space between vehicles is dictated by the speed of the regarded vehicle v_1, the speed of the leading vehicle v_2, and the distance $h = x_2 - x_1$. The perception threshold is defined to be a distance at which the driver realizes positive or negative differences of speed between himself and the vehicle in immediately in front.

$H_1 : v_2 - v_1 = -k_-(h - h_0)^2 - v_0$ perception threshold (negative)

$H_2 : v_2 - v_1 = -k_+(h - h_0)^2 + v_0$ perception threshold (positive)

These thresholds separate the regimes of reaction and no reaction of the driver. Here, $h_0 = 1/\rho_m$ is the distance at speed 0. Typical values for the physiological parameters v_0 and $k_\pm$ are:

v_0=0.3 m/s $k_- = 4 \cdot 10^{-4} m^{-1} s^{-1}$ $k_+ = 8 \cdot 10^{-4} m^{-1} s^{-1}$

Besides these perception thresholds, other thresholds for slowing down and acceleration procedures can be introduced so that

$H_3 : h = h_0 + T_r v_2$ Risky distance

$H_4 : h = h_0 + T_s v_1$ Safe distance

$H_5 : h = h_0 + T_d v_1$ Desired distance

Typical values are:

$T_r = 0.45s$ $T_s = 0.9s$ $T_d = 1.5s$

where T_r, T_s and T_d are the risky, safe and desired times respectively.

5.4 Microscopic Traffic Simulators

These models distinguish and trace single cars and their drivers. From driver behaviour and vehicle characteristics, position, speed and acceleration of each car are calculated for each time step.

A good review of the more popular models in use is given in **[90] [93]**. MIMIC, INTEGRATION, AIMSUM, MITSIM, VISSIM and SIMONE 2000.

5.4.1 The Traffic Simulator MIMIC

MIMIC is a microscopic traffic simulator that is based on stimulus-response car-following model as discussed in 5.2.2 **[91]**. In MIMIC, each driver maintains a preferred speed. When forced to deviate from this speed, the driver will attempt to resume it as soon as possible. The preferred speed is obtained from a set of normal distributions functions, based on observed traffic data. The normal distribution function can be expressed as:

$$f(t) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} \cdot e^{\frac{(t-\mu)^2}{2\sigma^2}} \tag{5.9}$$

Equation (5.9) represents the normal probability density function for preferred speed, based on observed traffic data, with $\mu = 80.030$ and $\sigma = 8.778$.

In MIMIC, the time gap distribution is expressed:

$$f(t) = \frac{1}{t \cdot \sigma \cdot \sqrt{2 \cdot \pi}} \cdot e^{\frac{(\ln t-\mu)^2}{2\sigma^2}} \tag{5.10}$$

Where:

$$\mu = \frac{1}{n} \cdot \sum \ln t \qquad \sigma^2 = \frac{1}{n-1} \cdot \sum (\ln t - \mu)^2$$

Equation (5.10) describes the time gap distribution using a log-normal distribution. It is based on an observed traffic volume of 917 vehicles/h, with $\sigma^2 = 0.661$ and $\mu = 0.945$.

The traffic simulation model MIMIC can be expressed as:

$$\alpha_i = \gamma \cdot (\Delta g - \Delta gp_i) \cdot e^{-a\left(1-\frac{1}{\Delta g}\right)} + \rho \cdot (vp_i - v_i) \cdot e^{-\beta\left(\frac{1}{\Delta g}\right)} \quad (5.11)$$

where:

$$\Delta g = x_{i-1}(t - t_r) - x_i(t - t_r) - L_{i-1} \quad (5.12)$$

$$\Delta gp_i = v(t - t_r) \cdot \Delta tp_i + a_i(t - t_r) \cdot \frac{\Delta tp_i^2}{2} \quad (5.13)$$

in which

t_r is the reaction time,

a_i is the rate of acceleration of vehicle i,

v_i is the actual rate of speed of vehicle i,

x_i is the position of vehicle i along the lane,

L_i is the length of vehicle i,

Δtp_i by vehicle i preferred time gap,

vp_i by vehicle i preferred rate of speed,

Δg is the actual space gap between two vehicles,

Δgp_i by vehicle i preferred space gap, and

γ, ρ, a, β are calibration parameters.

The authors in **[91]** do not mention how to obtain the values of the reaction time and the calibration parameters, however this is not important in the case of our concerned in reaching a high level of precision.

5.4.2 The Traffic Simulator Integration

The INTEGRATION model was conceived during the mid 1980s as an integrated simulation and traffic assignment model **[94]**. The model assumes a basic relationship between spatial headway (space between vehicles) and speed for each link, whose mathematical expression is:

$$h = c_1 + [c_2/(v_0 - v)] + c_3 \cdot v \qquad (5.14)$$

where:

$$c_2 = (1/d_m) \cdot [1/(\beta + 1/v_0)]$$

$$c_1 = \beta \cdot c_2$$

$$c_3 = [-c_1 + v_0/C - (c_2/(v_0 - v_{cr}))]/v_{cr}$$

$$\beta = (2 \cdot v_{cr} - v_0)/(v_0 - v_{cr})^2$$

with:

c_1, c_2 and c_3, parameters to be calibrated;

h, headway between two vehicles;

v, current speed of each vehicle;

v_0, free speed on the link;

v_{cr}, critical speed on the link;

C, capacity;

d_m jam (or maximum) density.

Parameters c_1, c_2 and c_3 are obtained for each link, through the attribution of specific values to free speed (v_0), critical speed (v_{cr}), capacity (C), jam density (d_m), as shown in table 5.1 **[82]**.

Link Classes	Link type	C (veh/h)	v_0 (km/h)	v_{cr} (km/h)	d_m (veh/km*lane)
1	Motorway	4000*	130	90	130
2	Acceleration lane	4950**	125	85	130
3	On/off ramp	1000	50	30	130
4	Merge	3500*	120	80	130
5	Toll station	-	35	20	130

* two-lane link, ** three-lane link

Table 5.1: Link categories and estimated values of the model parameters.

The traffic simulation model Integration assumes a basic relationship between the spatial headway and the current speed of each vehicle. Similar to MIMIC the model tries to maintain a free speed of the vehicle, based on several parameters to be calibrated during the simulation.

5.4.3 The traffic Simulator AIMSUN

The car-following model used in AIMSUN is a safety distance model, as previously discussed in 5.2.1. Here, vehicles are classified as free or constrained by the vehicle in front. When constrained by the vehicle in front, the follower tries to adjust its speed in order to obtain safe space headway to its leader. On the other hand, when free, the vehicle's speed is constrained by its desired speed and its maximum acceleration.

The speed during the time interval $[t, t+T]$, is chosen as:

$$v_n(t+T) = \min\left\{v_n^a(t+T), v_n^b(t+T)\right\} \tag{5.15}$$

The maximum speed a vehicle can accelerate during one time step is given by:

$$v_n^a(t+T) = v_n(t) + 2.5 \cdot a_n^{\max} \cdot T \cdot \left(1 - \frac{v_n(t)}{v_n^{desired}}\right) \cdot \sqrt{0.025 + \frac{v_n(t)}{v_n^{desired}}} \tag{5.16}$$

and the maximum safe speed for vehicle n with respect to the vehicle in front at time t is calculated as:

$$v_n^b(t+T) = d_n^{\max} \cdot T + \sqrt{\left[\left(d_n^{\max} \cdot T\right)^2 - d_n^{\max} \cdot \left[2\left\{x_{n-1}(t) - s_{n-1} - x_n(t)\right\} - v_n(t) \cdot T - \frac{v_{n-1}(t)^2}{\hat{d}_{n-1}}\right]\right]} \tag{5.17}$$

$$\hat{d}_{n-1} = \frac{d_n + d_{n-1}}{2} \tag{5.18}$$

where:

$a_n^{\max}$ maximum desired acceleration of the vehicle n, $(\frac{m}{s^2})$,

$d_n^{\max}$ maximum desired deceleration of the vehicle n, $(\frac{m}{s^2})$,

$\hat{d}_{n-1}$ estimation of maximum deceleration desired by vehicle n-1, $(\frac{m}{s^2})$,

s_{n+1} effective length of a vehicle, and consists of the vehicles length and the user specified parameter minimun distance between vehicles.

Typical values for AIMSUN model are showed in table 5.2

Parameter	Description	Value
$a_n^{\max}$	Maximum acceleration of the vehicle n	$2.5\ \frac{m}{s^2}$
$d_n^{\max}$	Maximum desired deceleration of vehicle n	$2\ \frac{m}{s^2}$
T	Reaction Time	0.7 seconds

Table 5.2: Typical values used in AIMSUN model.

The Traffic Simulation Model AIMSUM bases its performance on a safety distance model. Therefore, the maximum speed that the vehicle can accelerate in the next interval is determined according to the maximum safe distance between vehicles. The main drawback of the AIMSUM model is that it does not define clearly how to estimate the maximum deceleration of the vehicle n-1.

5.4.4 MITSIM Traffic Simulation Model

The car-following model used in MITSIM incorporates three regimes with different follower behaviour: free driving, following and emergency deceleration.

5.4.4.1 Free driving:

In this domain, the vehicle objective is to achieve its current desired speed. If the current speed is higher than the desired speed, the vehicle uses the normal deceleration rate to slow down to the desired speed. On the other hand, if the speed is lower than the desired speed, the vehicle uses its maximum acceleration rate to reach the desired speed as fast as possible. In MITSIM, the normal deceleration and the maximum acceleration rates are parameters, which are functions of vehicle type and the current speed. The acceleration rate of the vehicle n may be expressed as

$$a_n = \begin{cases} a_n^+ & v_n < v_n^{desired} \\ 0 & v_n = v_n^{desired} \\ a_n^- & v_n > v_n^{desired} \end{cases}, \tag{5.19}$$

Where a_n^+ is the maximum acceleration rate and a_n^- is the normal deceleration rate, both measured in ms^{-2}

5.4.4.2 Car-following:

In the car-following regime, the acceleration rate of vehicle n, a_n is calculated as:

$$a_n = \alpha^{\pm} \cdot \frac{v_n^{\beta^{\pm}}}{(x_{n-1} - l_{n-1} - x_n)^{\gamma^{\pm}}} \cdot (v_{n-1} - v_n) \tag{5.20}$$

where $\alpha^{\pm}$, $\beta^{\pm}$ and $\gamma^{\pm}$ are model parameters. α^+, β^+ and γ^+ are used if $v_n \le v_{n-1}$ and α^-, β^- and γ^- if $v_n > v_{n-1}$.

5.4.4.3 Emergency:

In this domain, the vehicle uses a deceleration rate that prevents collision and extends the headway. This deceleration rate is given by

$$a_n = \begin{cases} \min\left\{a_n^-, \dfrac{a_{n-1} - 0.5 \cdot (v_n - v_{n-1})^2}{(x_{n-1} - L_{n-1} - x_n)}\right\} & v_n > v_{n-1} \\ \min\left\{a_n^-, a_{n-1} + 0.25 \cdot a_n^-\right\} & v_n \le v_{n-1} \end{cases} \tag{5.21}$$

Typical values for MITSIM model are showed in table 5.3

Parameter	Description	Speed (m/s)				
a_n^-	Normal deceleration rate of vehicle n $(\frac{m}{s^2})$	< 6.1	6.1 – 12.2	12.2 – 18.3	18.3 – 24.4	> 24.2
		8.7	5.2	4.4	2.9	2

Parameter	Description	Speed (m/s)		
a_n^+	Maximum acceleration rate of vehicle n $(\frac{m}{s^2})$	< 6.096	6.096 – 12.192	> 12.192
		7.8	6.7	4.8

Parameter	Description	Value
α^+	Car-following parameter, acceleration	2.15
β^+	Car-following parameter, acceleration	-1.67
γ^+	Car-following parameter, acceleration	-0.89
α^-	Car-following parameter, deceleration	1.55
β^-	Car-following parameter, deceleration	1.08
γ^-	Car-following parameter, deceleration	1.65
h_{upper}	Max following time headway	1.36 s
h_{lower}	Min following time headway	0.5 s

Table 5.3: Typical values used in MITSIM model.

The Traffic Simulation Model MITSIM bases its conduct in three domains with different follower behaviour according to free driving, following and emergency deceleration. In MITSIM, the main drawback is that it bases its acceleration or deceleration rate to many speed intervals.

5.4.5 The Microscopic Traffic Flow Model VISSIM

VISSIM implements a psycho-spacing car-following model. The basic idea is that the driver can be in one of the four driving modes (Figure 5.1) **[95]**:

- Following
- Free driving
- Closing in
- Emergency Regime

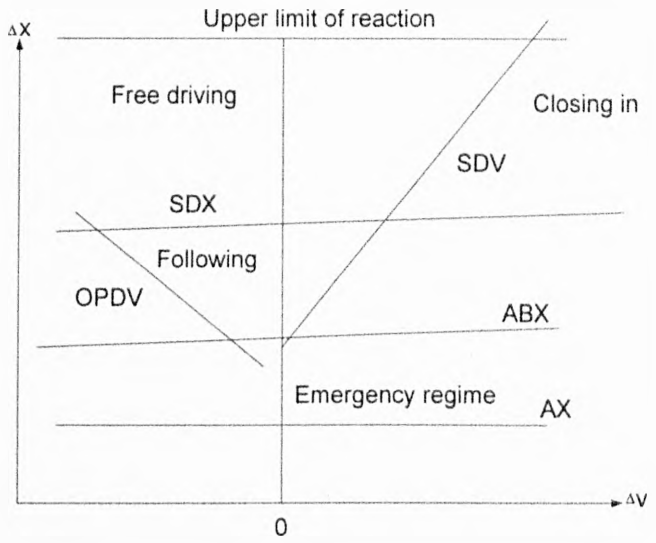


Figure 5.1: The different thresholds and regimes in the VISSIM model.

The critical situation occurs when a faster vehicle approaches a slower vehicle on a single lane and it has to decelerate. The action point of conscious reaction depends on the speed difference, distance and driver dependent behaviour.

Several conditions govern the VISSIM model:

1. ***The desired distance between stationary vehicles, AX.*** This threshold consists of the length of the front vehicle and the desired front-to-rear distance and is defined as:

$$AX = L_{n-1} + AXadd + RND1_n \cdot AXmult \qquad (5.22)$$

where $AXadd$ and $AXmult$ are calibration parameters. $RND1_n$, is normally a driver-dependent parameter.

2. ***The desired minimum following distance at low speed differences, ABX.*** This threshold is calculated as:

$$ABX = AX + BX \qquad (5.23)$$

with $$BX = (BXadd + BXmult \cdot RND1_n) \cdot \sqrt{v} \qquad (5.24)$$

where $BXadd$ and $BXmult$ are calibration parameters. The speed v is defined as:

$$v = \begin{cases} v_{n-1} \; for & v_n > v_{n-1} \\ v_n \; for & v_n \le v_{n-1} \end{cases} \tag{5.25}$$

3. ***The maximum following distance, SDX***. This distance varies between 1.5 and 2.5 times the minimum following distance, *ABX*. *SDX* is defined as:

$$SDX = AX + EX \cdot BX \tag{5.26}$$

with $EX = EXadd + EXmult \cdot (NRND - RND2_n)$ (5.27)

where *EXadd* and *EXmult* are calibration parameters. *NRND* is a distributed random number and $RND2_n$ is a driver-dependent parameter.

4. ***Approaching point, SDV***. This threshold is used to describe the point where the driver notices that he or she approaches a slower vehicle. *SDV* is defined as:

$$SDV = \left(\frac{\Delta x - L_{n-1} - AX}{CX} \right)^2$$

with $CX = CXconst \cdot (CXadd + CXmult \cdot (RND1_n + RND2_n))$ (5.28)

where *CXconst* , *CXadd* and *CXmult* are calibration parameters.

5. ***Increasing speed difference, OPDV***. This threshold describes the point where the driver observes that he or she is travelling at a lower speed than the leader. The threshold is defined as

$$OPDV = SDV \cdot (-OPDVadd - OPDVmult \cdot NRND) \tag{5.29}$$

where *OPDVadd* and *OPDVmult* are calibration parameters. NRDN is a distributed random number.

The threshold above gives rise to the following car-following regimes:

5.4.5.1 Following:

The thresholds *SDV*, *SDX*, *OPDV* and *ABX* constitute the following regime. The acceleration of vehicles is assumed to be different to zero at all times. When the vehicle passes *SDV* and *ABX* thresholds the acceleration rate is $-b_{null}$ and when the vehicle passes the threshold of *SDX* and *OPDV* it is assigned the acceleration of b_{null}. The acceleration or deceleration rate, b_{null} is defined as:

$$b_{null} = BNULLmult \cdot (RND4_n + NRND) \tag{5.30}$$

where *BNULLmult* is a calibration parameter. $RND4_n$ is driver parameter and *NRND* is a random number.

5.4.5.2 Free driving

In this domain, the vehicle uses its maximum acceleration to reach its desired speed. The maximum acceleration, b_{max} for passenger cars is defined as

$$b_{max} = BMAXmult \cdot (v_{max} - v \cdot FaktorV) \tag{5.31}$$

$$FaktorV = \frac{v_{max}}{v_{des} + FAKTORVmult \cdot (v_{max} - v_{des})} \tag{5.32}$$

where v_{max} is the maximum speed for the vehicle. *FAKTORVmult* is a calibration parameter.

5.4.5.3 Closing in

When the vehicle passes the *SDV* threshold, the driver notices that he or she is approaching a slower vehicle, then the driver decelerates in order to avoid collisions, the following deceleration rate is used:

$$b_n = \frac{1}{2} \cdot \frac{(\Delta v)^2}{ABX - (\Delta x - L_{n-1})} + b_{n-1} \tag{5.33}$$

where b_{n-1} is the deceleration of the leader.

5.4.5.4 Emergency regime:

When the front to rear distance is smaller than *ABX*, the follower decelerates to avoid colliding with the vehicle in front, according to the following:

$$b_n = \frac{1}{2} \cdot \frac{(\Delta v)^2}{AX - (\Delta x - L_{n-1})} + b_{n-1} + b_{min} \cdot \frac{ABX - (\Delta x - L_{n-1})}{BX} \tag{5.34}$$

The vehicle's maximum deceleration rate, b_{min}, is calculated as

$$b_{min} = -BMINadd - BMINmult \cdot RND3_n + BMINmult \cdot v_n \tag{5.35}$$

where *BMINadd* and *BMINmult* are calibration parameters. $RND3_n$ is a driver parameter. Typical values for the VISSIM model are showed in table 5.4

Parameter	Description	Value
AXadd	Additive calibration parameter	1.25
AXmult	Multiplicative calibration parameter	2.5
BXadd	Additive calibration parameter	2.0
BXmult	Multiplicative calibration parameter	1.0
EXadd	Additive calibration parameter	1.5
EXmult	Multiplicative calibration parameter	0.55
OPDVadd	Additive calibration parameter	1.5
OPDVmult	Multiplicative calibration parameter	1.5
CX	Calibration parameter	40
BNullmult	Multiplicative calibration parameter	0.1
NRND	Random number	N(0.5, 0.15)
*RND*1	Driver parameter	N(0.5, 0.15)
*RND*2	Driver parameter	N(0.5, 0.15)
*RND*4	Driver parameter	N(0.5, 0.15)
b_{max}	Max acceleration	$3.5-\frac{3.5}{40}\cdot v$
b_{min}	Max deceleration	$-20+\frac{1.5}{60}\cdot v$

Table 5.4: Typical values used in VISSIM model.

The Traffic Simulation Model VISSIM employs four regimes with several thresholds in the car-following model. The main drawback of VISSIM is that it requires many calibration parameters.

5.5.6 The Traffic Simulation Model Simone 2000

Simone 2000 can be described as a highly-detailed microscopic traffic simulation model for motorways with a variety of intelligent support systems: AICC (Autonomous Intelligent Cruise Control), SSL (Static Speed Limit Device), ISA (Intelligent Speed Limit Device), CDC (Centralized Distance Control), Platoon-Driving, and Multi-lane Platoon Driving.

- ***AICC (Autonomous Intelligent Cruise Control)***

Every vehicle equipped with AICC keeps a pre-defined time headway (gap) from the car preceding it, using an on-board distance and speed sensor, a computer unit, and algorithms.

- ***SSL (Static Speed Limit Device)***

A static speed limit device limits the maximum speed that can be achieved in the specific vehicle.

- ***ISA (Intelligent Speed Limit Device)***

An intelligent speed adapter limits the vehicle maximum speed, as the SSL does, but this limit is flexible and depends on the actual speed limit of the road section or the actual control regime. In order to adapt the speed of vehicles automatically, communication between roadside and the vehicle is needed, when the car enters or leaves a road section with a certain speed limit,

- ***CDC (Centralized Distance Control)***

An advanced version of the AICC is the Centralized Distance Control (CDC). The AICC functionality is present and discretionary. The CDC functionality is activated on request by a centralized control regime and is mandatory. The CDC system also maintains a safe distance to the car immediately in front and responds to speed differences, but uses distance parameters supplied by the roadside traffic centre, and thus requires roadside communication.

- ***Platoon Driving: ISA and CPC [+LG]***

The combination ISA and CPC is needed to drive a vehicle more or less automatically over a dedicated lane. The maximum speed is set by the roadside traffic centre and communicated to the equipped vehicle (by means of ISA). The distance control is maintained by the Centralized Platoon Control (CPC) system. This is a system similar to CDC but extended with *inter-vehicle communication* to allow for fast responses times, and to receive and send platoon commands. Here cars are guided along the dedicated lanes by a Lane Guidance system (LG), for example, by following magnetic markers in the road surface.

- ***Multi-Lane Platoon Driving: ISA and CPC and ILC [+LG]***
  The most sophisticated system considered can be described as automatic driving over a multi-lane motorway. *Inter-vehicle communication* is necessary to allow for fast response times while receiving and transmitting vehicle positions and speeds, as well as platoon-commands.

5.5.6.1 Distance Controller

The longitudinal distance controller is one of the main elements of a microscopic simulation model for traffic flows. It describes how a vehicle progresses along a lane focusing on its leader. In Simone, a detailed search procedure of the leader is performed to get the direct leader. The longitudinal controller determines the acceleration (positive or negative) needed to obtain a desired minimum distance to the leader. The desired gap function is defined:

$$s_i(t) = l_i + \eta_i(t) \cdot (z0_i + z1_i \cdot v_i(t) + z2_i \cdot v_i(t)^2) \qquad (5.36)$$

where:

$s(t)$	: desired gap distance (from rear follower i to rear leader) (m);
i	: index vehicle;
l	: length of vehicle i;
η	: congestion factor;
z0	: margin parameter (m);
z1	: linear headway parameter (s);
z2	: quadratic headway parameter (s^2);
V (t)	: speed at time t (m/s)

The congestion factor has been introduced to take into account changing driver behaviour in congested conditions. The value of the proportional congestion factor η is a function of the experienced traffic state (congested/non-congested) and driving speed in case of congested conditions. In non-congested driving state, the factor η is equal to one. For congested conditions, the congestion factor is equal to 1.3. The values for z0, z1 and z2 are defined in **[96]**. The approach described here for the desired gap distance is also followed in other microscopic models (FOSIM, and MIXIC).

5.5.6.2 Longitudinal Controller

The longitudinal controller tries to minimize the distance error and speed differences of subject vehicle i with respect to the leading vehicle i-1. The model can be described as follows:

$$a_i(t+\tau) = \alpha_i \cdot \left(x_{i-1}(t) - x_i(t) - s_i(t)\right) + \beta_i^{\pm} \cdot \left(v_{i-1}(t) - v_i(t)\right) \qquad (5.37)$$

with:

$a(t+\tau)$: Acceleration applied after delay time (m/s^2)

$x(t)$: x-coordinate vehicle rear bumper at time t (m)

$v(t)$: speed at time t (m/s)

i : index subject vehicle (follower)

$i-1$: index subjects' leader

α : distance error sensitivity (1/s)

β^+ : speed difference sensitivity (for positive difference) (1/s^2)

β^- : speed difference sensitivity (for negative difference) (1/s^2)

The Table 5.5 shows some constant for Simone 2000.

Margin Parameter (m)	2.5
Linear headway parameter (s)	0.55
Quadratic headway parameter (s^2)	0.005
Distance error sensitivity (1/s)	0.3
Positive speed difference	2.0
Negative speed difference	1.7

Table 5.5: Constants used in Simone 2000.

The Traffic Simulation Model Simone 2000 bases it performance on the distance and longitudinal controller. Simone 2000 has been designed for use on a motorway with a variety of intelligent support systems. However, its basic mechanism can be employed in urban environments.

5.5 CONCLUSIONS

Since the appearance of the 1985 Highway Capacity Manual, there has been an increasing amount of research on traffic flow models, which has led to a different understanding of how traffic operates, especially on a motorway [97]. Efforts to implement ITS with regard to both traffic management and traffic information provision will provide challenges for applying this improvement on new microscopic traffic models.

One car following model experiment was studied segment by segment using a model where the stimulus included terms proportional to deviations from the mean inter-vehicle spacing, deviation from the mean speed of the lead vehicle and deviations from the mean speed of the following car [98]. An interesting result of the analysis of this model is that it implies an asymmetry in the response, depending on whether the relative stimulus is positive or negative.

In this chapter, we have analyzed six microscopic traffic models. These microscopic traffic models are based on three types of car-following models: safe-distance models, stimulus-response and psycho-physiological models. Of the studied models VISSIM contains the largest number of parameter and AIMSUM is the model with the smallest number of parameters. The main drawback of MIMIC, INTEGRATION and VISSIM models are the calibration parameters. These parameters have to be calibrated during the simulation period. AIMSUM model does not define clearly how to estimate the maximum deceleration of the vehicle n-1 and MITSIM bases its acceleration or deceleration rate to many speed intervals.

The traffic simulation model Simone 2000, seems to be more suitable for our simulation scenarios. It based its behaviour in two different controllers: *Distance Controller and Longitudinal Controller*. In addition, it incorporates the effect when the relative speed stimulus is positive or negative. We will use this microscopic traffic model to simulate the mobility of the vehicles on a motorway and in an urban scenario, further in this thesis.

CHAPTER 6: VALIDATION OF THE SIMULATION ALGORITHM IN A SMALL AND LARGE SCALE AD-HOC NETWORK

6.1 INTRODUCTION

There are three general forms of experimental techniques: measurements, analysis and simulation. Here, our goal is to develop of an algorithm for inter-vehicle communication. Measurements, although possible, would be impractical for many nodes. In addition, we would also need to considerate the economic cost associated with deployment. Analysis, although useful, would not be very suitable for a very complex system **[107]**. Simulation models are increasingly being used in complex systems, where several parameters need to be considered **[108]**. Here, we will use the results of a test-bed to validate our algorithm in a small scale ad-hoc network. Then, with the use of extensive simulation, we will validate our protocol in a large scale ad-hoc network with two prominent ad-hoc routing algorithms, DSR and AODV, in both urban and motorway environments. Finally, we will compare our positional algorithm with another positional algorithm, Greedy Parameter Stateless Routing (GPSR).

6.2 DYNAMIC SOURCE ROUTING PROTOCOL (DSR)

DSR uses source routing rather than hop-by-hop routing. Each packet carries in its header the complete, ordered list of nodes through which the packet must pass. The key advantage of source routing is that intermediate nodes do not need to maintain up-to-date routing information in order to route the packets they forward, since the packets themselves already contain all the routing decisions. This fact, coupled with the on-demand nature of the protocol, eliminates the need for the periodic route advertisement and neighbour detection packets present in other protocols.

6.2.1 Basic Mechanisms of DSR

The DSR protocol consists of two mechanisms: *Route Discovery* and Route *Maintenance*. Route Discovery is the mechanism by which a node S wishing to send a

packet to a destination D obtains a source route to D. To Perform a Route Discovery, the source node S broadcasts a Route Request packet that is flooded through the network in a controlled manner and is answered by a Route Reply packet from either the destination node or another node that knows a route to the destination. To reduce the cost of Route Discovery, each node maintains a cache of source routes it has learned or overheard, which it aggressively uses to limit the frequency and propagation of Route Requests.

Route Maintenance is the mechanism by which a packet's sender S detects if the network topology has changed such that it can no longer use its route to the destination D because two nodes listed in the route have moved out of range of each other. When Route Maintenance indicates a source route is broken, S is notified with a Route Error packet. The sender S can then attempt to use any other route to D already in its cache or can invoke Route Discovery again to find a new route.

6.2.2 Implementation Decision

Using the model developed by the National Institute of Standard and Technology (NIST) **[109]**, we have enhanced the unicast mechanism for data packets. The enhanced mechanism requires that a destination node return an acknowledgement packet for each data packet received. The acknowledgement packet is returned by reversing the path over which the data packet came.

6.3 AD-HOC ON-DEMAND DISTANCE VECTOR (AODV)

AODV is essentially a combination of both DSR and DSDV **[110]**. It borrows the basic on-demand mechanism of Route Discovery and Route Maintenance from DSR, plus the use of hop-by-hop routing, sequence numbers, and periodic beacons from DSDV.

6.3.1 Basic Mechanism

When a node S needs a route to some destination D, it broadcasts a Route Request message to its neighbours, including the last known sequence number for that

destination. The Route Request is flooded in a controlled manner through the network until it reaches a node that has a route to destination. Each node that forwards the Route Request creates a reverse route for itself back to node S.

When the Route Request reaches a node with a route to D, that node generates a Route Reply that contains the number of hops necessary to reach D and the sequence number for D most recently seen by the node generating the Reply. Each node that participating in forwarding this Reply back toward the originator of the Route Request (node S), creates a forward route to D. The state created in each node remembers only the next hop and not the entire route, as would be done in source routing.

In order to maintain routes, AODV normally requires that each node periodically transmit a Hello message with a once per second default rate. Failure to receive three consecutive Hello messages from a neighbour is taken as an indication that the link to the neighbour in question is down.

When a link goes down, any upstream node that has recently forwarded packets to a destination using that link is notified via an Unsolicited Route Reply containing an infinite metric for that destination. Upon receipt of such a Route Reply, a node must acquire a new route to the destination using Route Discovery as described above.

6.3.2 Implementation Decision

Similar to what we did in DSR, we used the model developed by the National Institute of Standard and Technology (NIST) **[109]** for AODV. Furthermore, we also enhanced the unicast mechanism for data packets. The enhanced mechanism requires that a destination node return an acknowledgement packet for each data packet received. The acknowledgement packet is returned by reversing the path over which the data packet came.

6.4 GREEDY PERIMETER STATELESS ROUTING (GPSR)

GPSR is based on two forwarding methods: Greedy forwarding and perimeter forwarding **[35]**. In Greedy forwarding, a forwarding node can make a locally greedy choice. This means that at each hop packets are sent to the neighbour that is closest to

the indicated destination's position. On the other hand, in perimeter forwarding, the node chooses the right-hand rule. The right-hand rule traverses the interior of a closed polygonal region in clock-wise edge order.

6.4.1 Implementation Decision

In GPSR, a node knows the position of its neighbours by means of their beacons and the position of a packet's destination, with the help of the location service. The location service that GPSR uses is the Reactive Location Service (RLS). Figure 6.1 shows the flow diagram that we have for the beaconing and registration of neighbour nodes in GPSR. In addition, we have implemented a location discovery procedure containing Location Request packet (LREQ) and a Location Reply packet (LREP). Whenever the position of a node is required, the node looking for location information floods a request containing the ID of the node it is looking for. When a node receives a request with its own ID, it replies to the node looking for its position. Routing is done in a greedy way by forwarding the packet to neighbour closer to the physical location of the destination. This local optimal choice repeats at each intermediate node until the destination is reached. Thus, in order to forward packets to the destination the correspondent node only requires the knowledge of its neighbours' location.

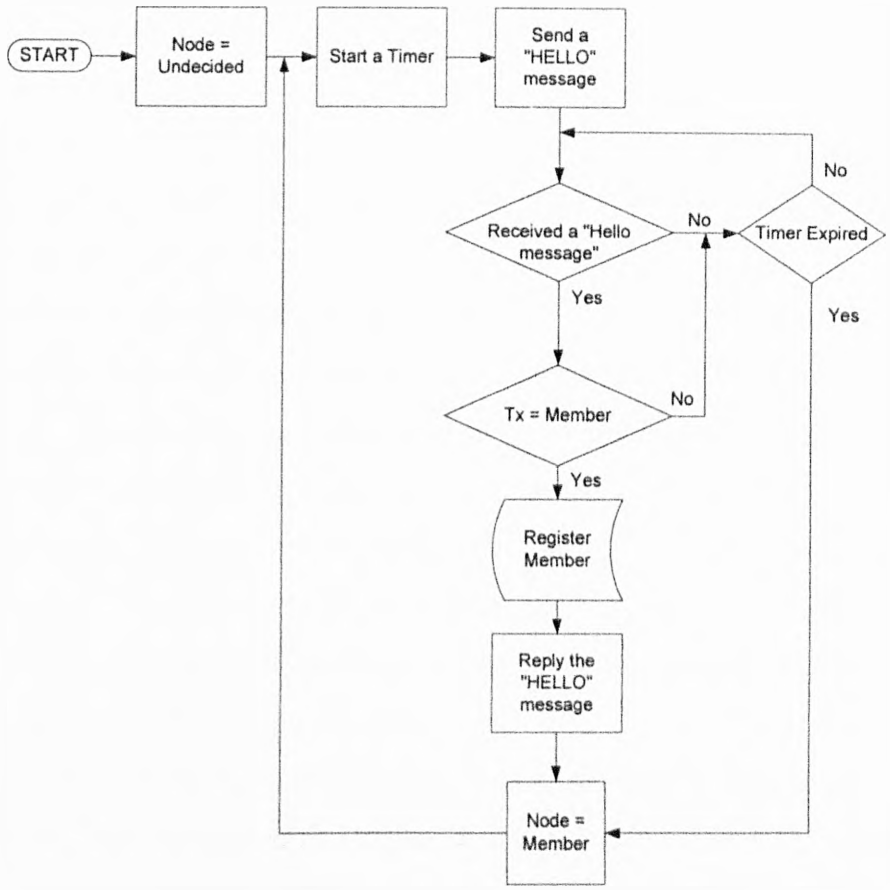


Figure 6.1: Beaconing and registration of neighbours' nodes in GPSR.

6.5 METHODOLOGY

First, we have modified the DSR algorithm (see appendices) with an acknowledgement packet. This allows the algorithm to determine when the link between the source and the destination is broken.

Then, we also modified the AODV algorithm (see appendices). Similarly to DSR, an acknowledgement packet has been included.

The next part of the work was the validation of our protocol in small scale ad-hoc network with the results of a test-bed. Here, AODV, DSR and LORA-CBF were compared in terms of end-to-end delay and throughput against the Associativity-Based Routing Protocol (ABR) **[25]**. Based on the results obtained in a small-scale ad-hoc network, the next part of the work was the validation of the LORA-CBF algorithm in a large-scale ad-hoc network. We selected two prominent routing algorithms: DSR and AODV. We propose to validate our LORA-CBF algorithm in both and urban and

motorway environment. These scenarios represent, to a great degree, the typical application of vehicular networks.

The goal of our simulations was to validate our model and compare the reaction of non-positional and positional routing algorithms to network topology changes.

For urban environments, we have used a square figure. This topology is a typical representation of the urban traffic flow of a city, where vehicles change their speed and direction around the corners. Here, we have considered an average speed of 8 m/s (18 miles/hour). On the other hand, for motorway environments, we have used a circular figure. Also, this topology can represent the mobility of the vehicles on a motorway, and since the rate of the curvature is small compared with the length, vehicle mobility is more constant. We have considered an average speed of 42 m/s (95 miles/hour). In addition, we have considered three lanes per direction. These three lanes per direction are reasonable for UK motorway traffic.

Our protocol evaluations were based on the simulation of 250 vehicles forming an ad-hoc network, moving around a circular and a square road of 6283 m length (Figure 6.2).We have considered only the car-following model because it can be applied in both traffic scenarios. The car-following model describes the interactions with preceding vehicles in the same lane and it only considers the gap and the speeds between vehicles to promote their next position.

We have selected the microscopic traffic model Simone 2000 to be implemented in OPNET **[111]**, because it is the most popular traffic simulation model for motorway environments and it employs the car-following model as its longitudinal distance controller. In addition, it also incorporates the effect when the relative speed stimulus is positive or negative.

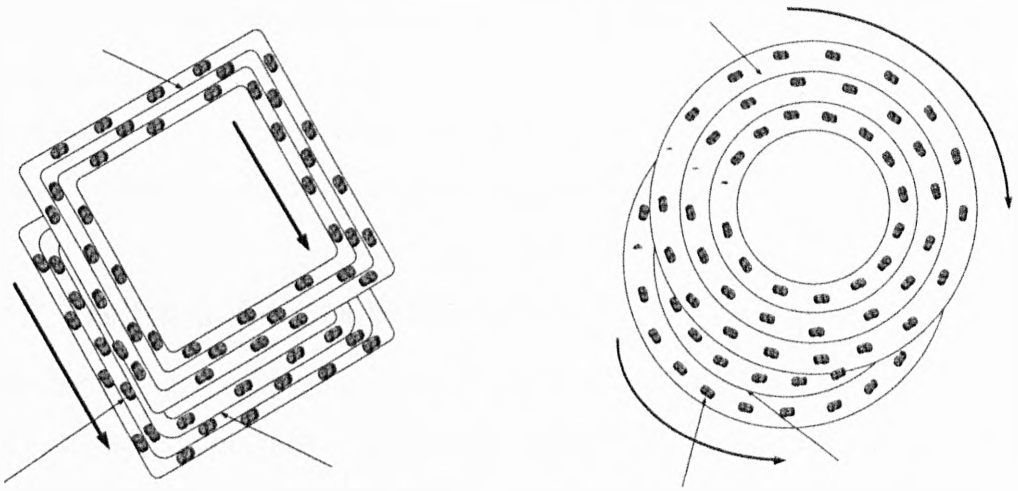


Figure 6.2: Scenarios evaluated. Vehicles are considered to be arranged in an endless loop.

6.5.1 Communication Model

The model for evaluating the routing protocols is implemented in OPNET. Because one of the goals of our simulation is to compare the performance of each routing protocol, we have chosen our traffic sources to be constant bit rate (CBR) sources. When defining the parameters of the communication model, we experimented with data rates of 1 Mbps and 11 Mbps and packet sizes of 1448 bytes. We have selected the same size of packets used in the test-bed. The communication between the source and destination is peer-to-peer, and communication is initiated at times uniformly distributed between 1 – 5 seconds. Since both DSR and AODV lack a predicting mechanism, we started their communication at 1 s. Details about the prediction mechanism can be found on page 70, in Chapter 3.

It should be noted that to make a fair comparison between algorithms that do and do not have such a mechanism, it was necessary to arrange both DSR and AODV by effectively managing the disconnection between transmitter and receiver. For LORA-CBF, we have started at five seconds because of its cluster formation. The cluster formation mechanism requires less than two seconds, but we have started at five seconds to guarantee a steady-state network. Also, we have implemented several metrics used commonly in the evaluation of routing algorithms: route discovery time,

average end-to-end delay of data packets, routing load, routing overhead, overhead and packet delivery ratio.

Every value represented in each graph depicts the results of 30 simulations and the error is less than 10 % in most of the parameters analysed.

6.5.2 Medium Access Mechanism

The IEEE 802.11b Distributed Coordination Function (DCF) is used as the medium access control protocol. DCF is designed to use both physical carrier sense and virtual carrier sense mechanisms to reduce the probability of collisions due to hidden terminals.

The transmission of each unicast packet is preceded by a Request-to-Send/Clear-to-Send (RTS/CTS) exchange that reserves the wireless channel for transmission of a data packet. Each correctly received unicast packet is followed by an acknowledgement (ACK) to the sender, which retransmits the packet a limited number of times until the ACK is received. Broadcast packets are sent only when virtual and physical carrier sense indicates that the medium is clear, but they are not preceded by RTS/CTS and are not acknowledged by their recipients.

The physical radio characteristics of each mobile node's network interface, such as the antenna gain, transmit power, and receiver sensitivity, were chosen to approximate the Enterasys's IEEE 802.11b WLAN **[75]** direct sequence spread spectrum radio. A transmission range of 300 m. was chosen which is consistent with current 802.11b Wireless LAN and 5 dBi gain car-mounted antennas. An experiment was realized between two vehicles driving in opposite direction in order to validate the transmission range. The results of this experiment are included on page 85, in Chapter 4.

6.6 VALIDATING THE ALGORITHM IN A SMALL SCALE-NETWORK

We have validated our simulation models with the results of an experimental test-bed **[112]**. Here, the Associativity-based routing protocol (ABR) **[25]** has been evaluated in a multi-hop network. ABR is a source-initiated on-demand ad-hoc routing protocol that discovers routes when the source desires it. ABR uses periodic beaconing to inform neighbouring nodes about their presence. Nodes running the ABR protocol maintain a

neighbouring table which indicates how many times the beacon from a neighbour has been heard over a specific period of time.

In this experiment **[112]**, four Laptop computers were configured with the enhanced TCP/IP/ABR protocol. The nodes were positioned so that they had only one upstream and one downstream neighbour. As shown in Figure 6.3, only one hop physical links were possible. The control was realized by C.-K Toh, Minar Delwar and Donald Allen.

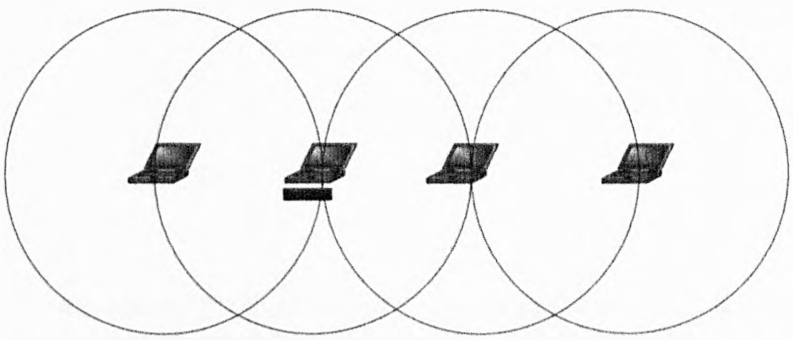


Figure 6.3 Test-bed considered and replicated in OPNET.

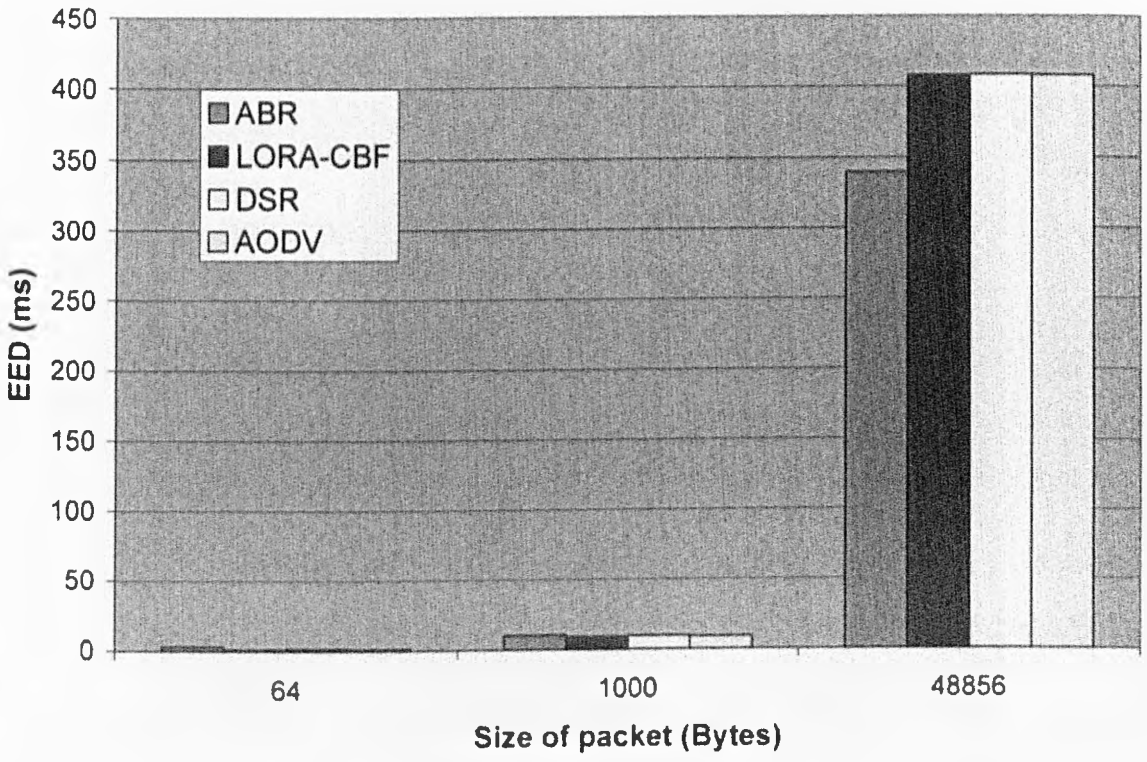
To partially validate our model, we compared the control of **[112]** with the algorithms of AODV, DSR and LORA-CBF. We found that results from small packets (64 Bytes) were slightly different, but the results obtained from medium and large packets (1000, 1448 and 48856 Bytes), are very similar (Table 6.1 and 6.2 and Graphs 6.1, 6.2, 6.3, 6.4, 6.5 and 6.6). The difference at 64 Bytes may be due to the increased packet exchanging between two nodes. In IEEE 802.11b, every data packet is verified by an acknowledgement packet. In small packets, this increase is considerably.

EED (ms) 1 hop	64 Bytes	1000 Bytes	48856 Bytes
ABR (C. –K. Toh)	3.25	10.40	340
LORA-CBF	0.937	9.102	407.53
DSR	1.463	9.628	408.044
AODV	0.929	9.094	407.51

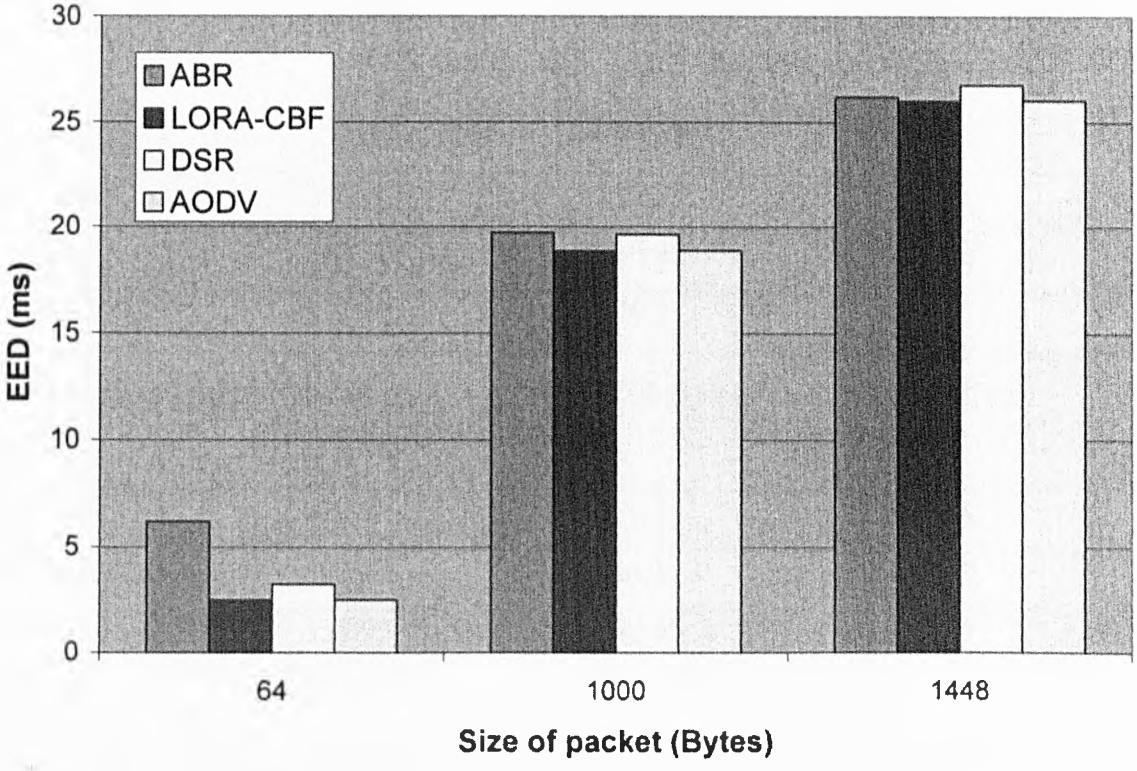
EED (ms) 2 hops	64 Bytes	1000 Bytes	1448 Bytes
ABR (C. –K. Toh)	6.20	19.7	26.20
LORA-CBF	2.524	18.854	26.022
DSR	3.283	19.614	26.782
AODV	2.548	18.879	26.047

EED (ms) 3 hops	64 Bytes	1000 Bytes	1448 Bytes
ABR (C. –K. Toh)	8.80	29.2	38.30
LORA-CBF	4.095	28.591	39.343
DSR	5.199	29.695	40.447
AODV	4.157	28.653	39.405

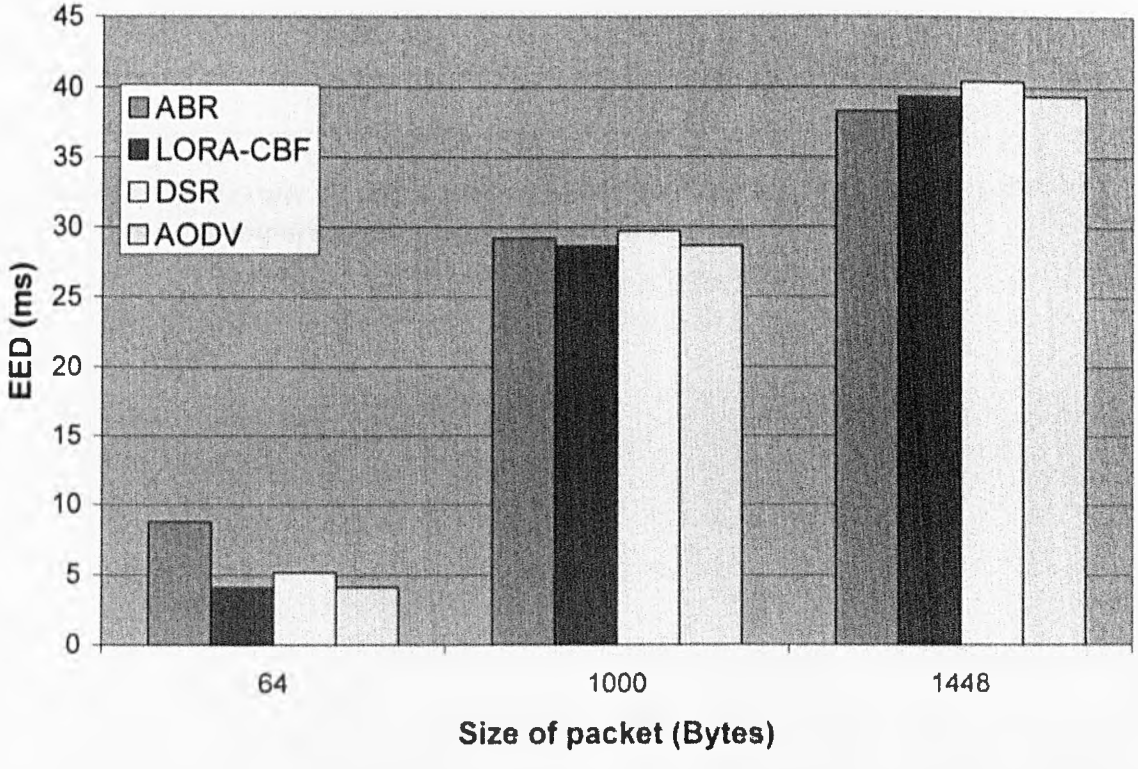
Table 6.1 ***Results of EED obtained from simulation using OPNET and compared with the results of the test-bed.***



Graph 6.1: ***Results of EED obtained from simulation using OPNET and compared with the results of the test-bed in one hop.***



Graph 6.2: Results of EED obtained from simulation using OPNET and compared with the results of the test-bed in two hops.



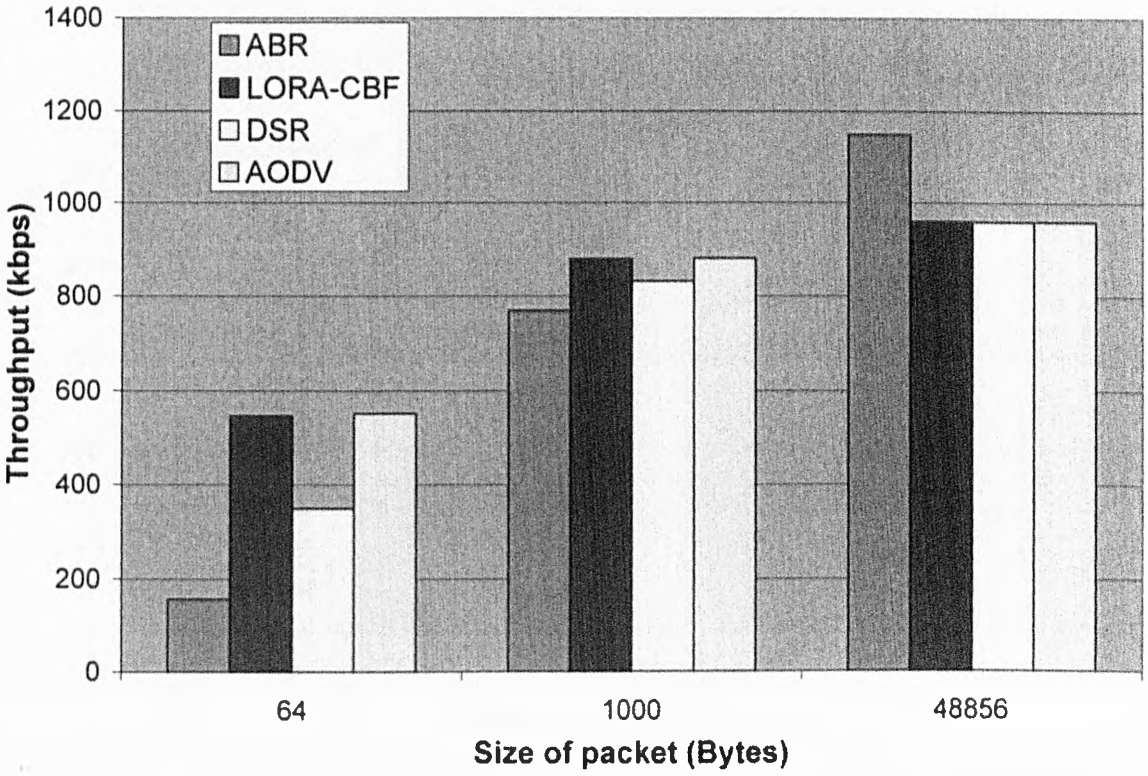
Graph 6.3: Results of EED obtained from simulation using OPNET and compared with the results of the test-bed in three hops.

Throughput (kbps) 1 hop	64 Bytes	1000 Bytes	48856 Bytes
ABR (C. –K. Toh)	157.54	769.23	1149.55
LORA-CBF	546.4	878.93	959.1
DSR	349.96	830.91	957.86
AODV	551.13	879.7	959.1

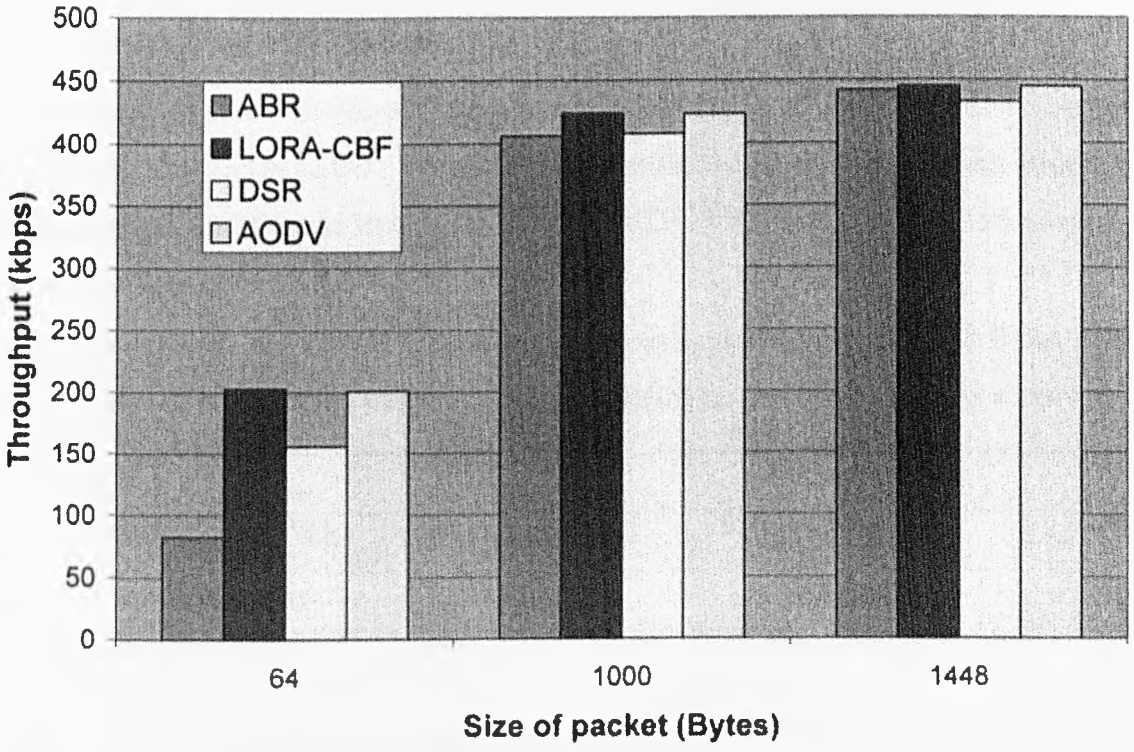
Throughput (kbps) 2 hops	64 Bytes	1000 Bytes	1448 Bytes
ABR (C. –K. Toh)	82.58	406.1	442.14
LORA-CBF	202.85	424.3	445.16
DSR	155.95	407.87	432.53
AODV	200.94	423.75	444.73

Throughput (kbps) 3 hops	64 Bytes	1000 Bytes	1448 Bytes
ABR (C. –K. Toh)	58.18	273.97	302.45
LORA-CBF	125	279.8	294.44
DSR	98.48	269.41	286.4
AODV	123.17	279.2	293.97

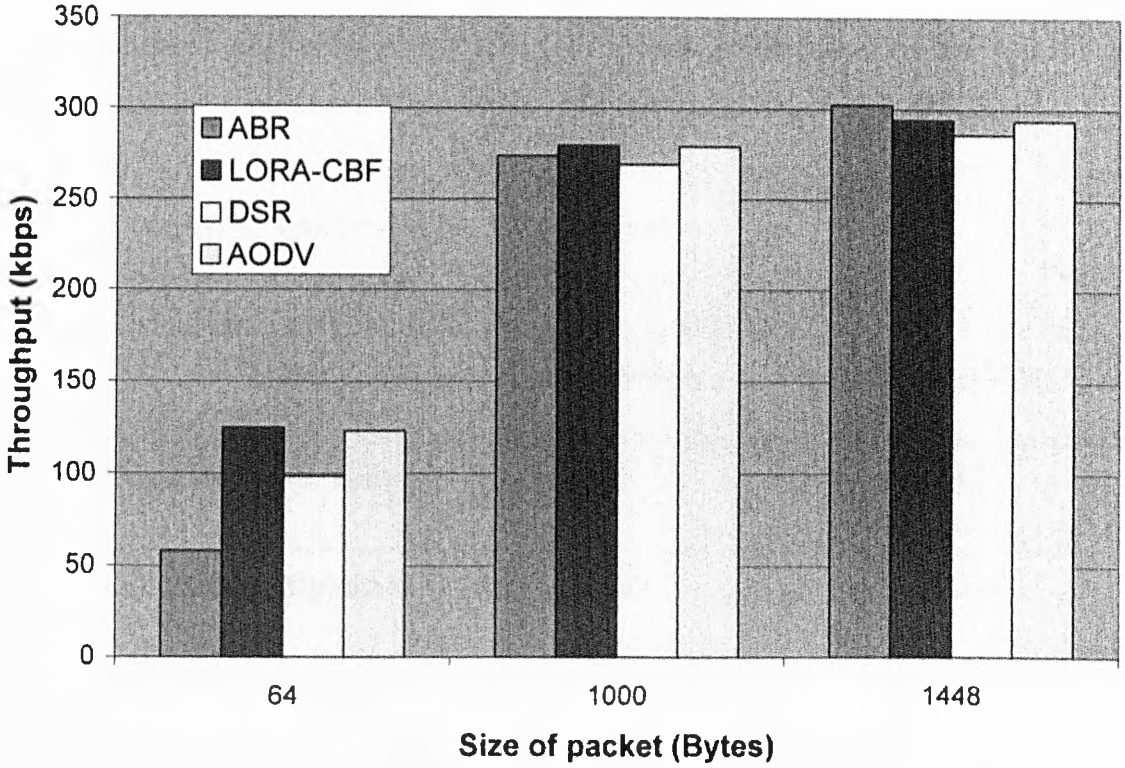
Table 6.2 ***Results of Throughput obtained from simulation using OPNET and compared with the results of the test-bed.***



Graph 6.4: Results of Throughput obtained from simulation using OPNET and compared with the results of the test-bed in one hop.



Graph 6.5: Results of Throughput obtained from simulation using OPNET and compared with the results of the test-bed in two hops.



Graph 6.6: Results of Throughput obtained from simulation using OPNET and compared with the results of the test-bed in three hops.

6.6.1 One-hop validation

Two metrics were considered in the one hop validation of our models: Throughput and End to End Delay (EED). For further confirmation, we also applied equation C.10 for throughput and time to transmit a packet (EED) in IEEE 802.11 wireless networks, (see appendices).

Table 6.3 shows the results of one hop with data packet size of 1000 Bytes and a range of 300 m between the transmitter and the receiver. The three values are similar.

	EED	Throughput
Test Bed (C-K Toh)	10.4 ms	769.23 Kbps
Numerical Analysis	8.4 ms	956 Kbps
OPNET model	9.1 ms	878.93 Kbps

Table 6.3: Results validating LORA-CBF in one hop

The numerical analysis does not take into account retransmissions due to collisions. This reflects the minimum difference between the results showed.

6.6.2 Two and three-hops validation

EED (ms)	**Two Hops**	**Three Hops**
Test Bed (C-K Toh)	19.7	29.2
LORA-CBF Algorithm	18.854	28.591
Throughput (Kbps)	**Two Hops**	**Three Hops**
Test Bed (C-K Toh)	406.091	273.972
LORA-CBF Algorithm	424.313	279.808

Table 6.4: Results validating LORA-CBF for two and three hops

Table 6.4 shows the results of the comparison between the results of the test bed and the simulation results in OPNET validating LORA-CBF. According to our results, we believe that our algorithm has been validated in one, two and three hops. This small-scale network is economically feasible, but a large-scale network is impractical due to deployment costs of deployment. We have used extensive simulation for the evaluation of LORA-CBF in a large-scale network.

6.7 Validating the Location Routing Algorithm with Cluster-Based Flooding (LORA-CBF) in a large-scale network

We have compared our algorithm with the models of two prominent algorithms, AODV and DSR. The comparison is reasonable because we have improved the data reception mechanism with an acknowledgement packet in AODV and DSR protocols. When the timer for an acknowledgement data packet expires, AODV and DSR starts a new Route Request (RREQ) packet.

6.7.1 Metrics of Simulation

In comparing the performance of the algorithms in large scale ad-hoc networks, we have chosen to evaluate them according to the most common metrics:

Route discovery time (Latency): Is the amount of time the source has to wait for sending the first data packet.

Average end-to-end delay of data packets: This includes all possible delays caused by buffering during route discovery, queuing at the interface queue, retransmissions delays at the MAC, and propagation and transfer times.

Routing load: *Is* measured in terms of routing packets transmitted per data packets transmitted. The latter includes only the data packets finally delivered at the destination and not the ones that are dropped. The transmission of each hop is counted once for both routing and data packets. This provides an idea of network bandwidth consumed by routing packets with respect to "useful" data packets.

Routing overhead: Is the total number of routing packets transmitted during the simulation. For packets sent over multiple hops, each transmission of the packet (each hop) counts as one transmission.

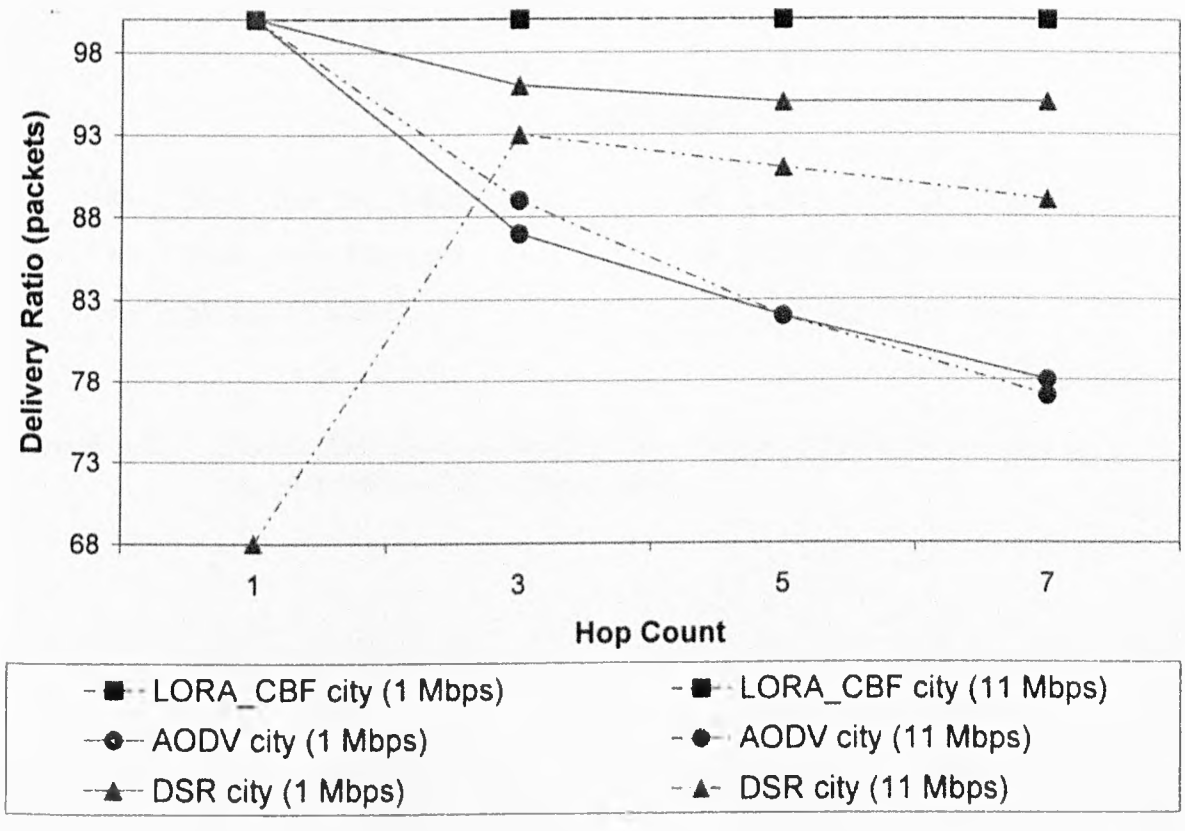
Overhead (packets): *I*s the total number of routing packets generated divided by the total number of data packets transmitted, plus the number total routing packets.

Packet delivery ratio: Is measured as a ratio of the number of data packets delivered to the destination and the number of data packets sent by the sender. Data packets may be dropped en route for one reason: the next hop link is broken when the data packet is ready to be transmitted.

6.7.1.1 A Comparison of the algorithms LORA-CBF, AODV and DSR on a multi-lane rectangular dual carriageway representative of city driving

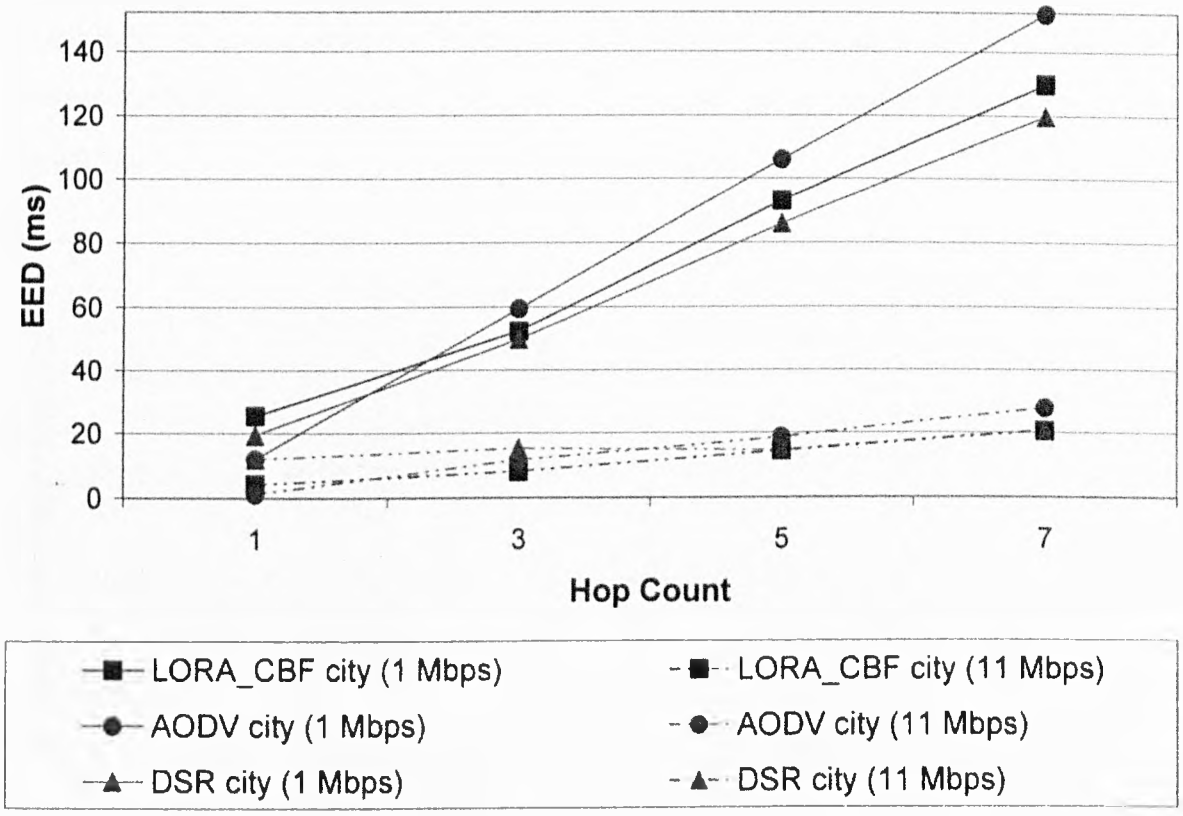
The results for the next series of important comparisons were acquired using the models described in sections 6.2 and 6.3 and Chapter 3, using the parameters shown in section 6.4.1. Care was taken to make sure that results are not biased.

Graph 6.7 shows delivery ratios of the three algorithms considered for 1 Mbps and 11 Mbps and shows that congestion due to lack of spatial diversity is a problem for DSR at the higher speed. However, at a low speed (1Mbps), it has a better delivery ratio compared to AODV. LORA-CBF has the best performance at both data rates, in contrast with AODV that has the lowest delivery ratio.



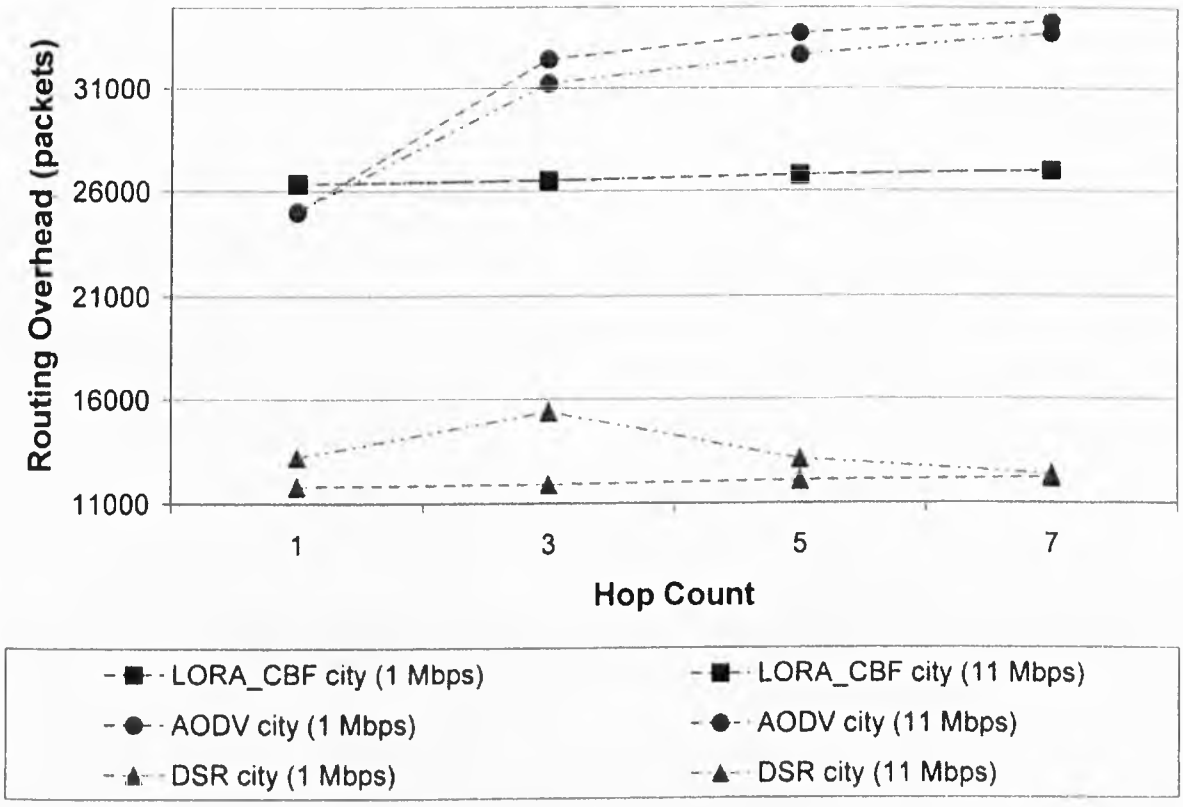
*Graph 6.7: **Delivery Ratio on a multi-lane rectangular dual carriageway representative of city driving (packets).***

The results in Graph 6.8 show the End-to-End Delay (EED). All the algorithms evaluated have shown higher EED with a data rate of 1 Mbps and a better EED with a data rate of 11 Mbps. This is because more bits are transmitted at higher data rates.



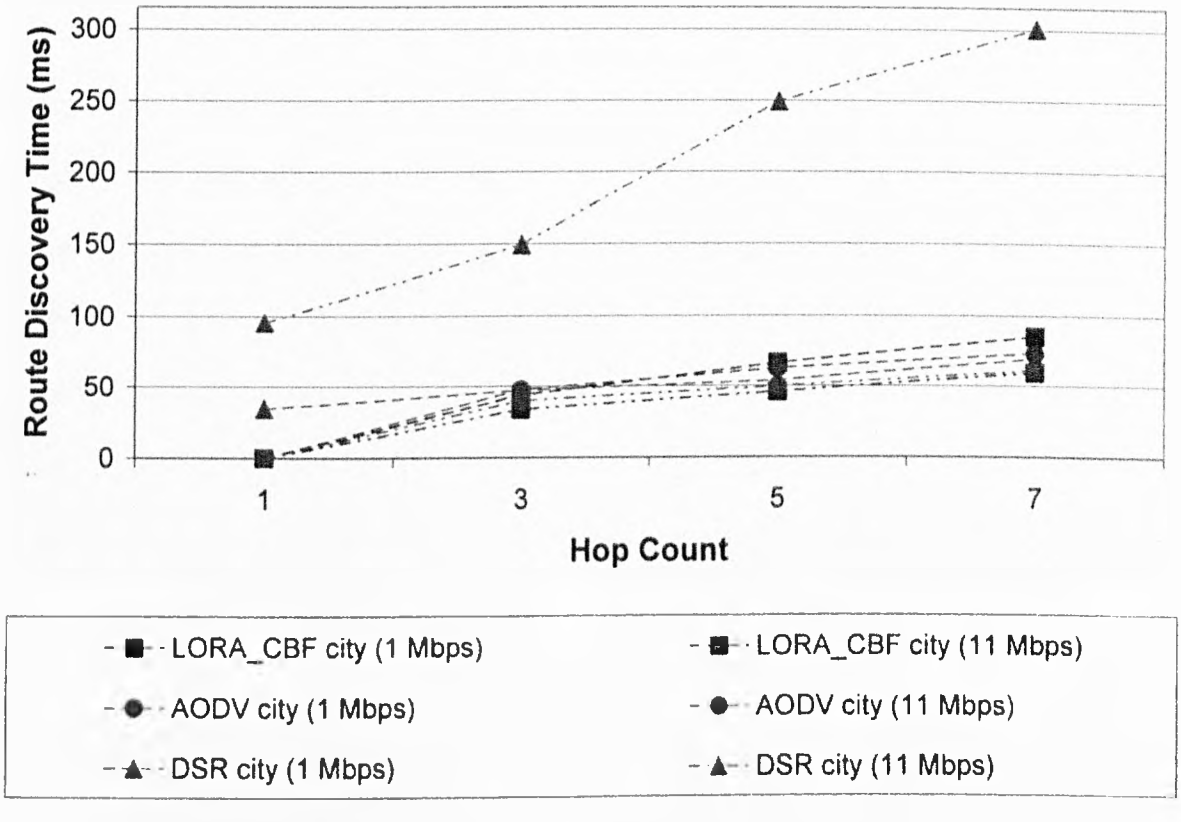
Graph 6.8: ***End-to-End Delay on a multi-lane rectangular dual carriageway representative of city driving (ms).***

Routing Overhead is shown in Graph 6.9. The results show that DSR, of all the algorithms analyzed, has the best performance for overhead. This is because DSR lacks transmission control. This means that with a data rate of 11 Mbps, DSR has slightly higher routing overhead when compared to a data rate of 1 Mbps. This is caused by congestion due to lack of spatial diversity. We can recollect that spatial diversity is the mechanism of transmitting the signal via several independent diversity branches to get independent signals replicas. This can be realized using multiple transmit/receive antennas.



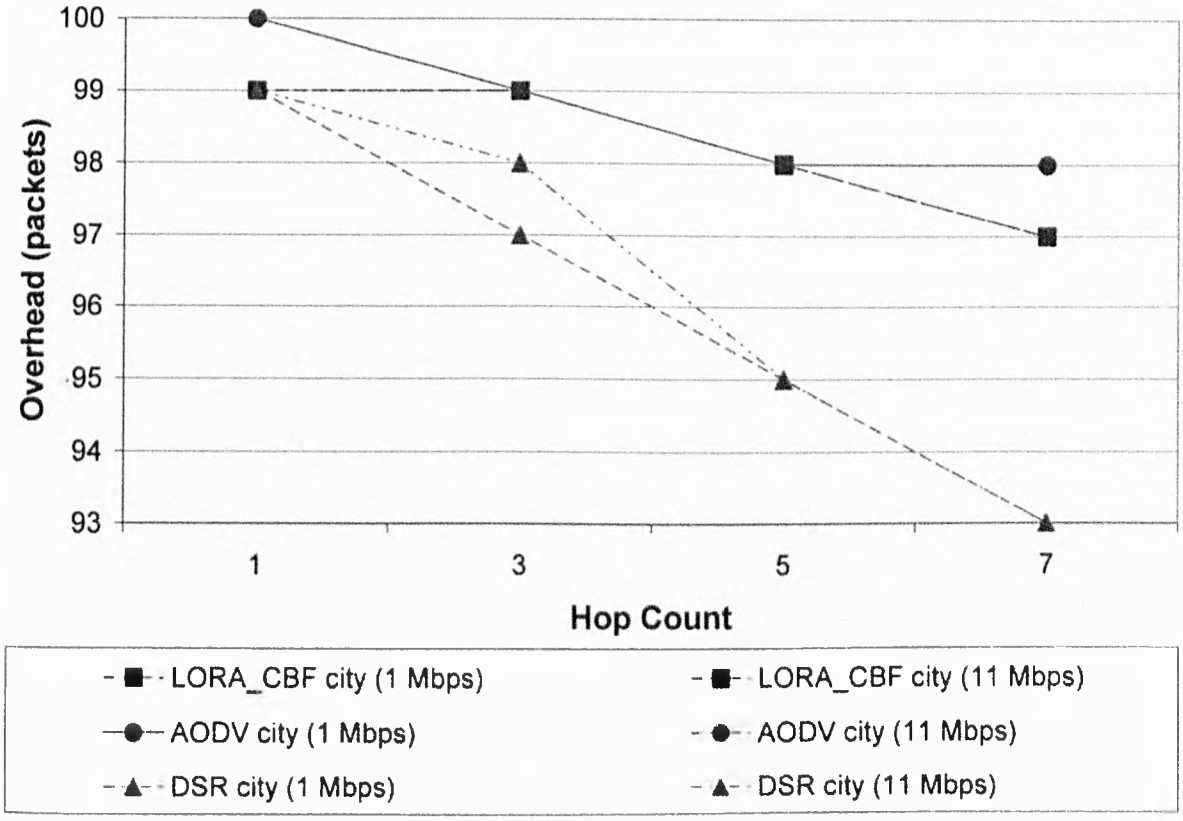
Graph 6.9: Routing Overhead on a multi-lane rectangular dual carriageway representative of city driving (packets).

Graph 6.10 shows the route discovery times for the three algorithms considered. DSR has a poor performance with a data rate of 11 Mbps. In contrast with a data rate of 1 Mbps, it has similar behaviour compared to AODV and LORA-CBF.



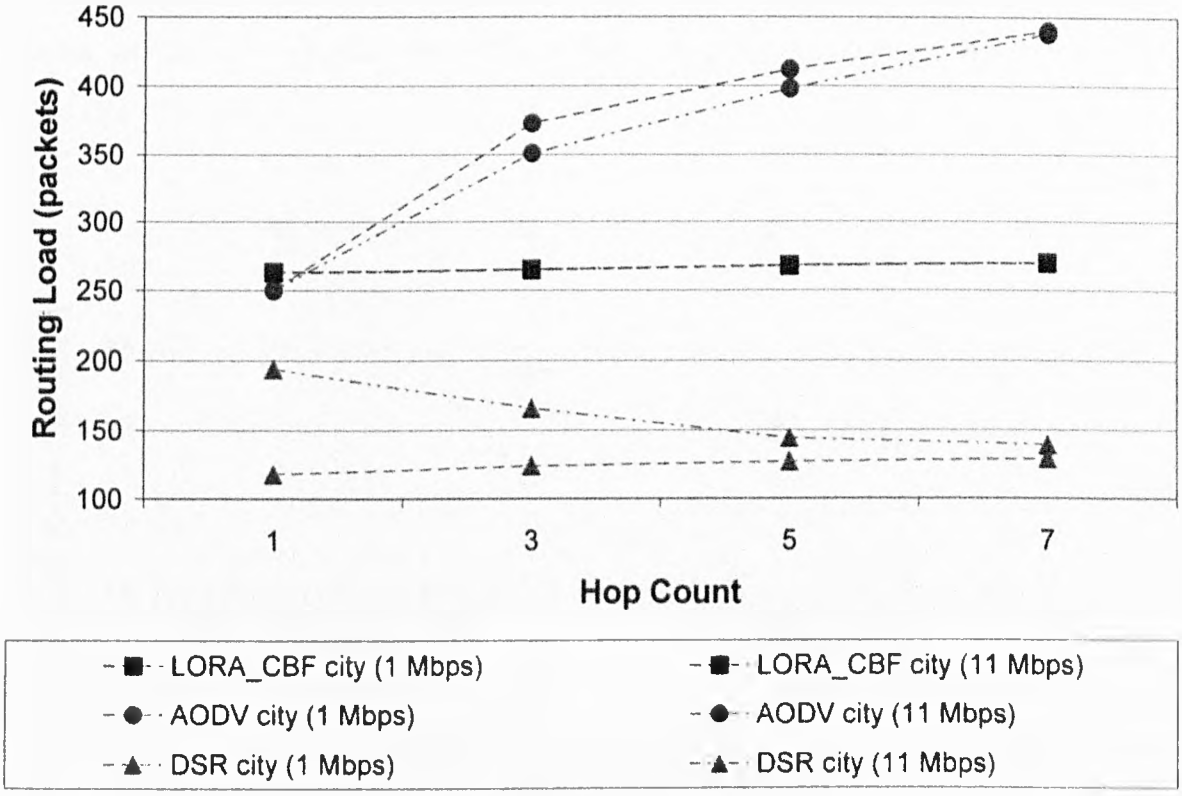
Graph 6.10: Route Discovery Time on a multi-lane rectangular dual carriageway representative of city driving (ms).

Overhead is shown in Graph 6.11. Again, AODV has the lowest performance. On the other hand, DSR has the highest performance due to its lack of transmission control. LORA-CBF has relatively good performance in spite of its complexity and robustness. We define complexity in terms of its structure and functions, and robustness in terms of neighbour mobility and link breaks.



Graph 6.11: Overhead on a multi-lane rectangular dual carriageway representative of city driving (packets).

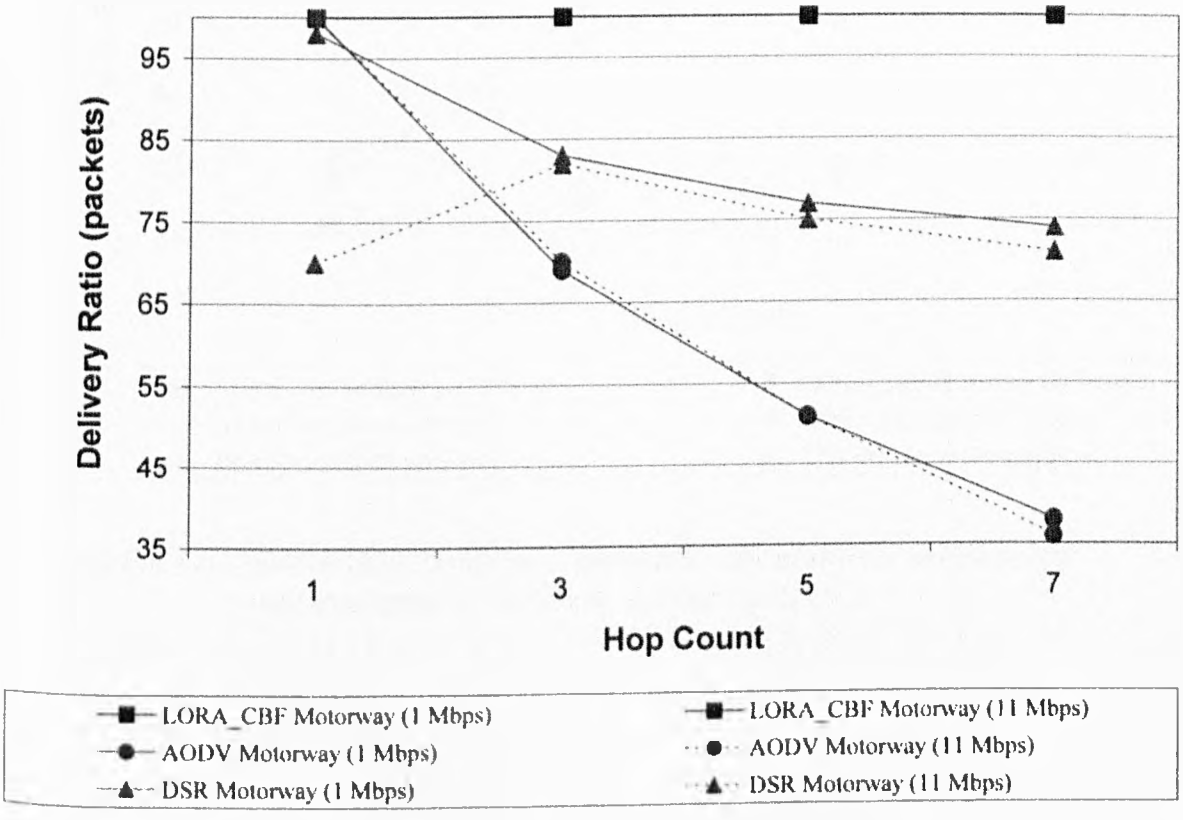
In Graph 6.12 we see the routing load, showing that AODV has more routing load than LORA-CBF and DSR. Again, DSR has the best performance in terms of routing load due to its lack of a neighbour sensing mechanism, as discussed on Chapter 3. LORA-CBF maintains its routing load at a constant level with both data rates. Routing load is indicative of scalability because it shows the number of packets used to maintain the routing tables updated.



Graph 6.12: Routing Load on a multi-lane rectangular dual carriageway representative of city driving (packets).

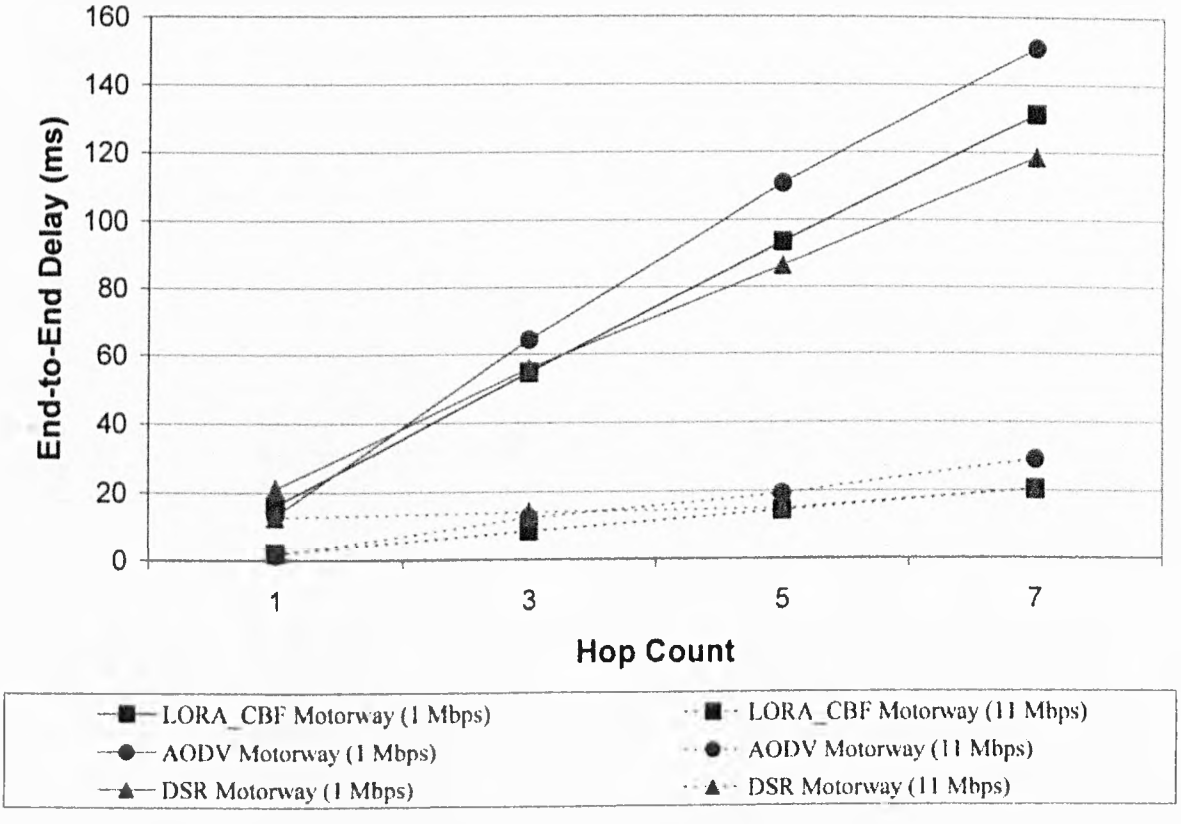
6.7.1.2 A Comparison of the algorithms LORA-CBF, AODV and DSR on a multi-lane circular dual carriageway representative of motorway driving

Once again we selected a packet size of 1448 Bytes, and considered two data transmission rates for our simulations: a theoretical maximum of 11 Mbps and the worst case with 1 Mbps. Graph 6.13 compares the packet delivery ratio of the three algorithms considered. LORA-CBF shows good results with both data rates. AODV has a slightly worse packet delivery ratio than DSR. DSR shows the poorest delivery ratios with a data rate of 11 Mbps



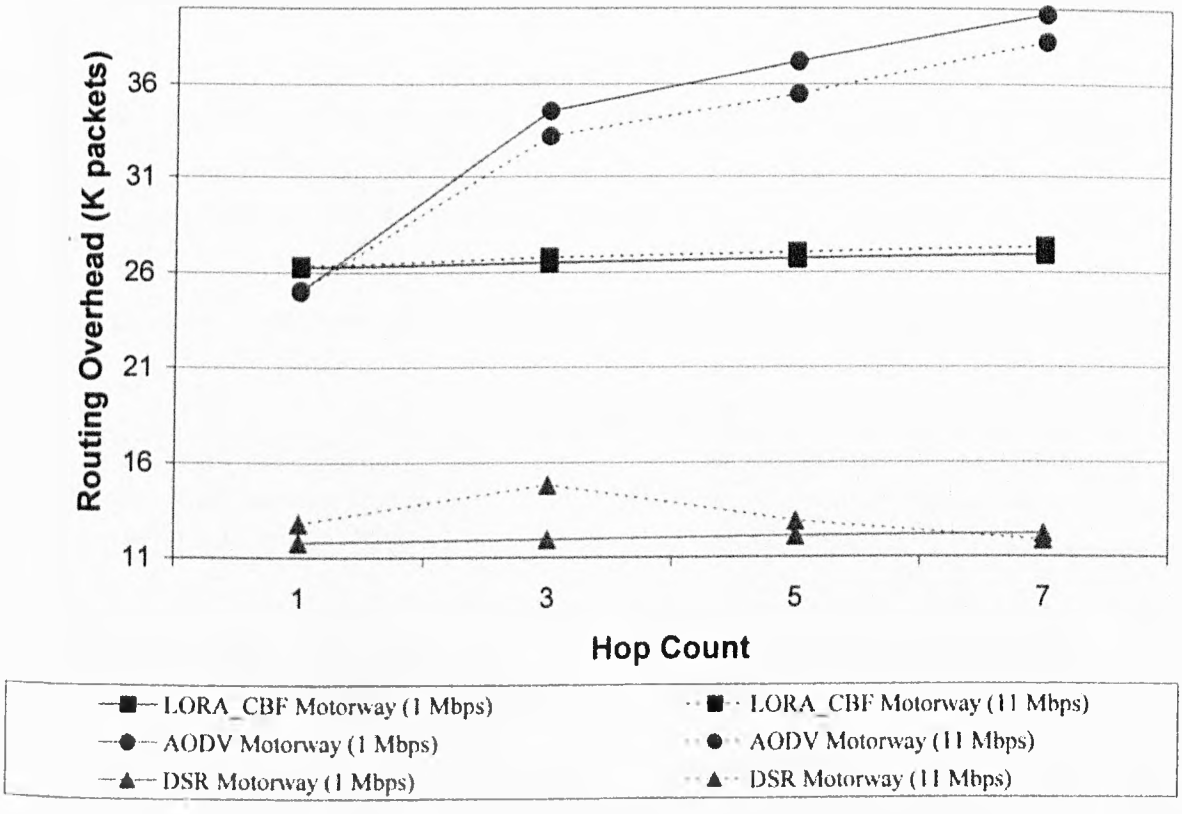
Graph 6.13: Delivery Ratio on a multi-lane circular dual carriageway representative of motorway driving (packets).

End to End delay (EED) is shown in Graph 6.14. All of the algorithms have the lower performance with a data rate of 1 Mbps. In general, AODV has the worse delay due its frequent retransmissions. DSR has been shown to have higher performance due its economy of control packets. LORA-CBF has slightly larger EED compared with DSR.



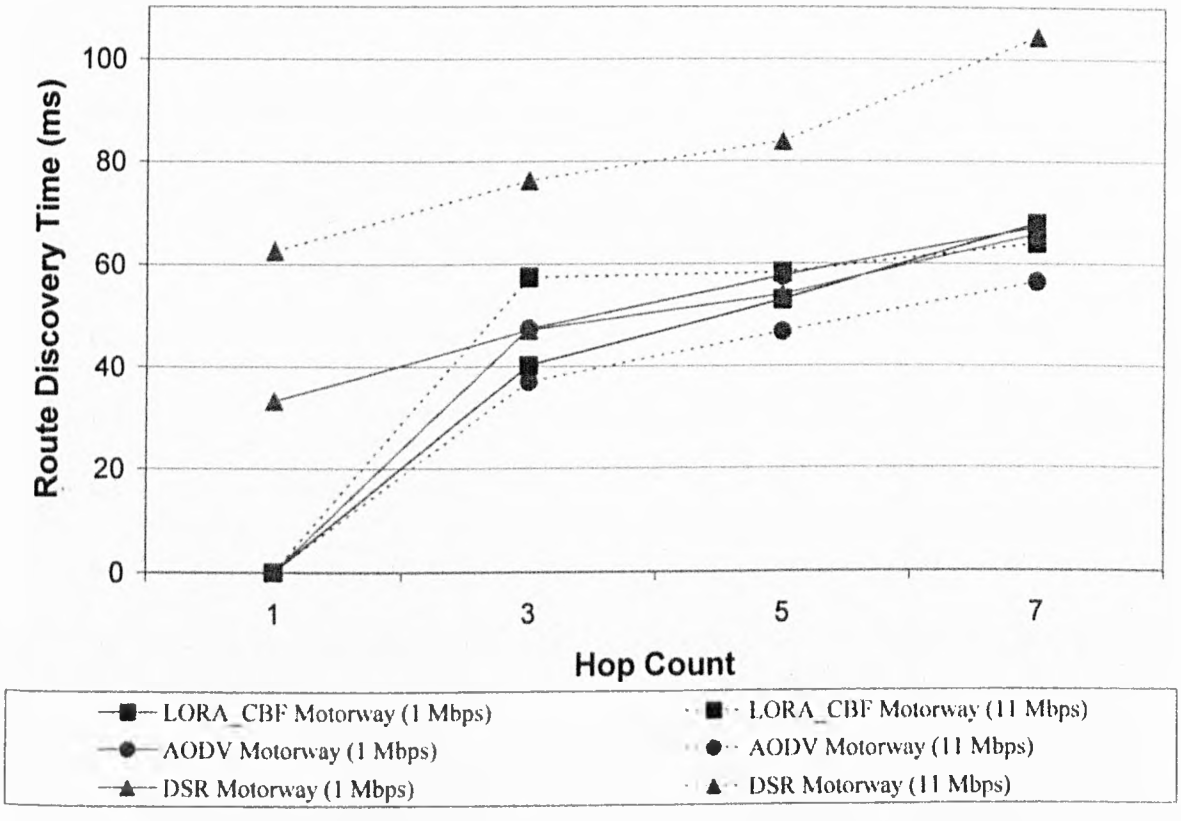
Graph 6.14: End-to-End Delay on a multi-lane circular dual carriageway representative of motorway driving (ms).

Graph 6.15 shows the routing overhead. Here, DSR has the better performance in terms of routing overhead due to its lack of neighbour sensing mechanism and AODV increases its routing overhead according to the distance between nodes. LORA-CBF maintains its routing overhead at an almost constant level, because the level depends on the frequency of a Hello messages. This is constant and independent of the maximum distance between communication partners. AODV requires about 3 times the routing overhead of DSR (Also reported in **[48]**).



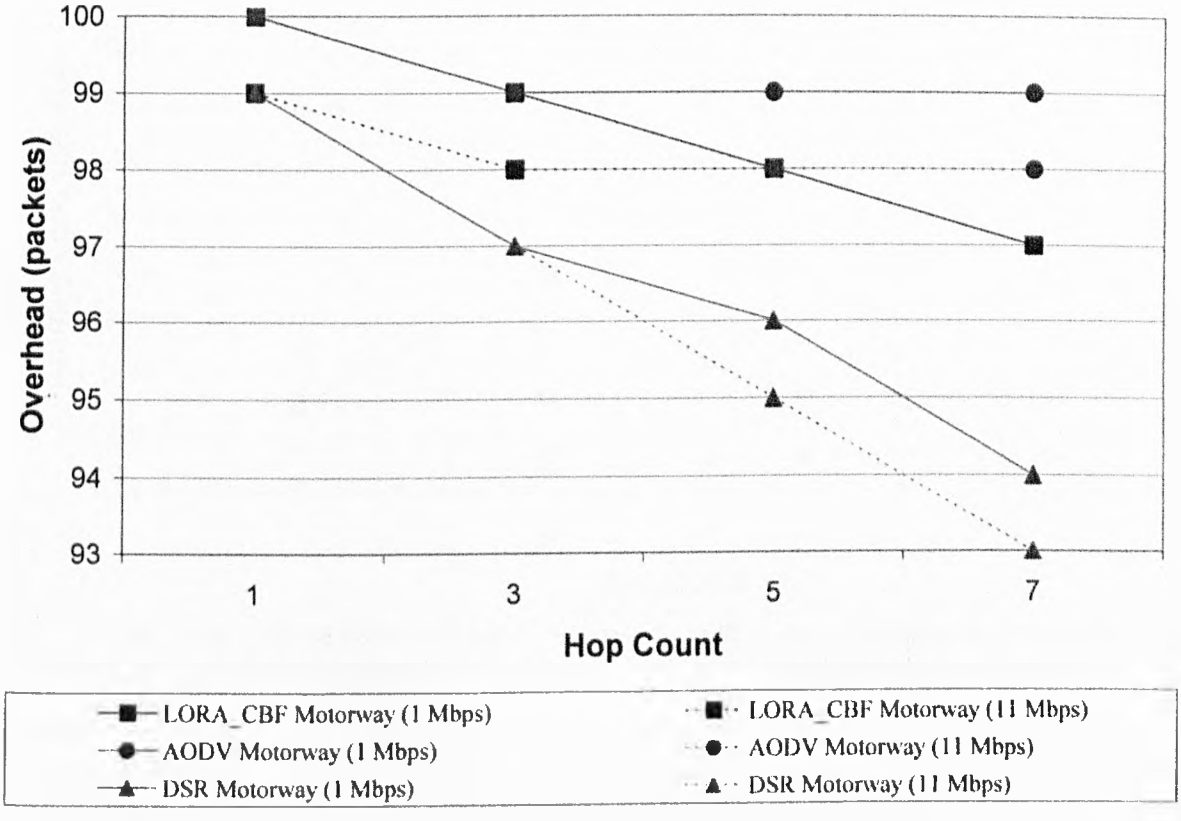
Graph 6.15: Routing Overhead on a multi-lane circular dual carriageway representative of motorway driving (packets).

Graph 6.16 shows the route discovery time for all the algorithms evaluated. DSR has been shown to have worst behaviour at 11 Mbps and its performance with a data rate of 1 Mbps is similar to AODV and LORA-CBF.



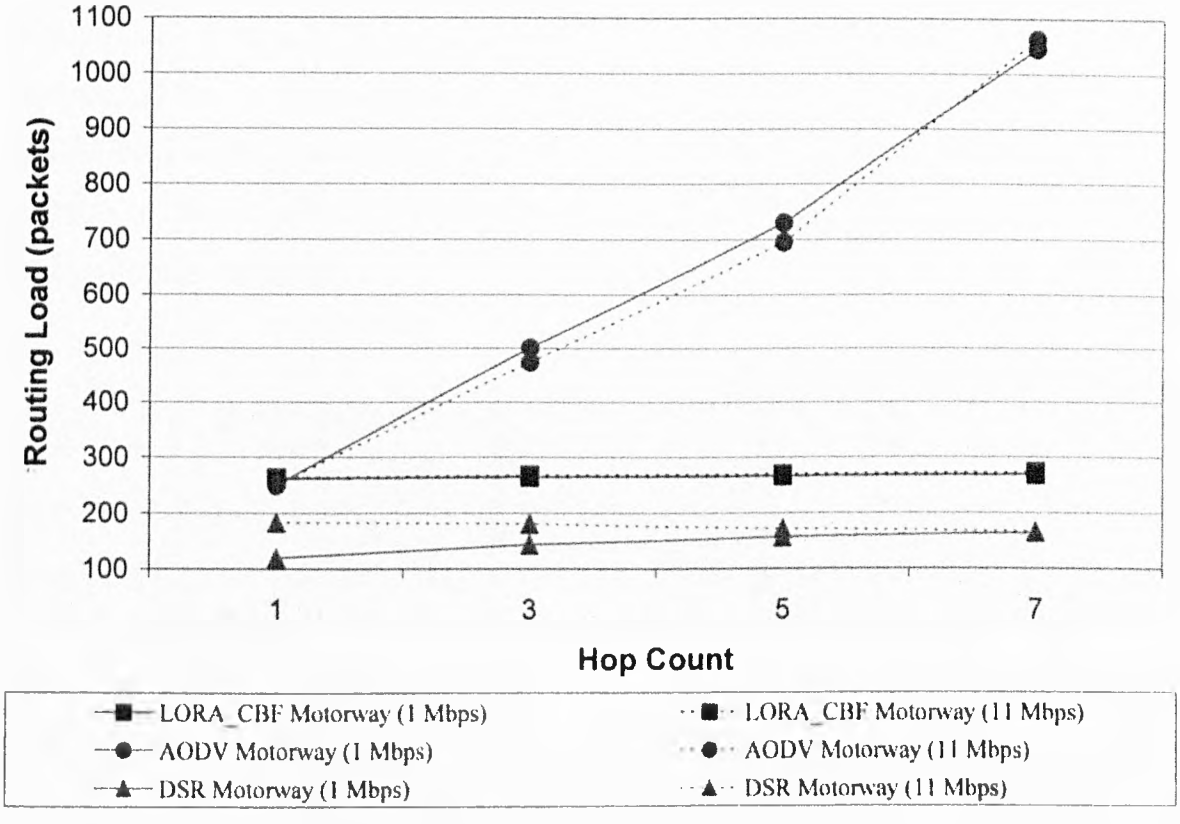
Graph 6.16: Route Discovery Time on a multi-lane circular dual carriageway representative of motorway driving (ms).

Overhead is shown in the Graph 6.17. Again, AODV has the lowest performance. Highly mobile environments have high link breaks rates and each breaks cause a transmission of RERR messages. In the case of AODV, the overhead is increased by the Hello messages.



Graph 6.17: Overhead on a multi-lane circular dual carriageway representative of motorway driving (packets).

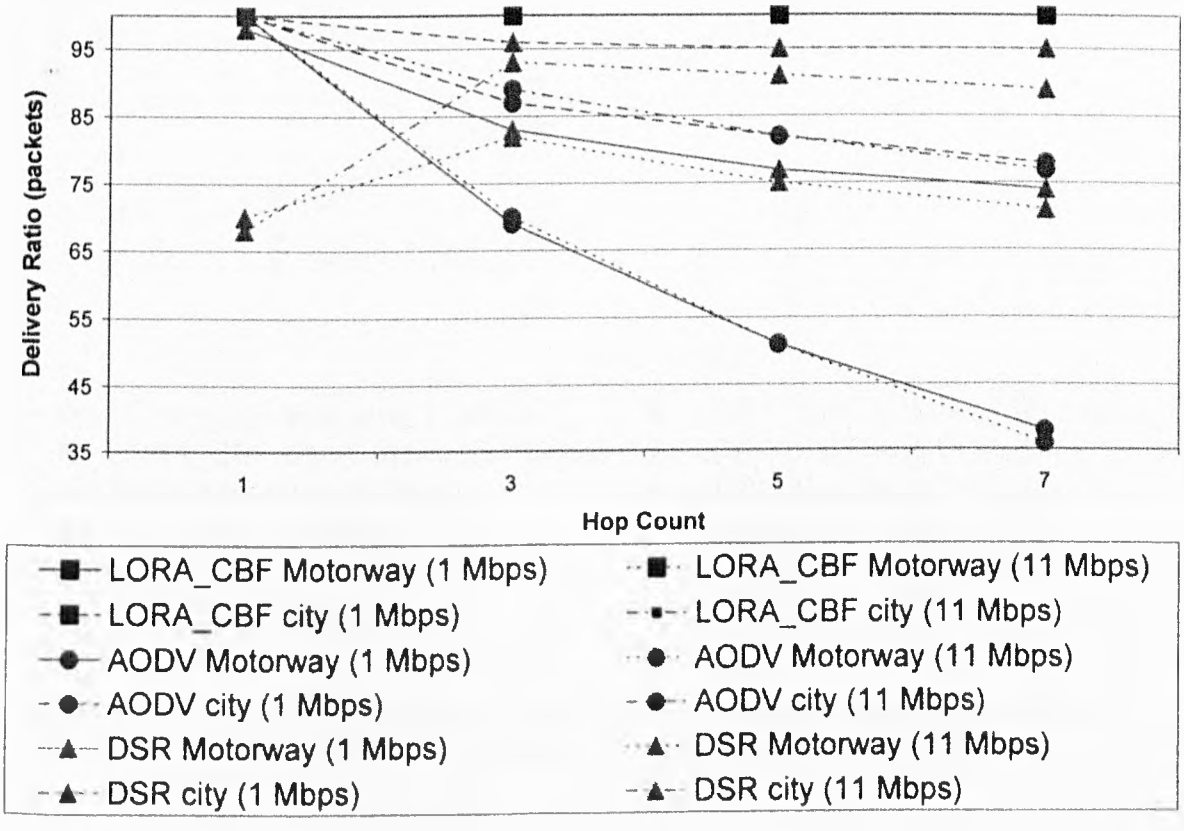
Graph 6.18 represents the routing load. AODV shows more routing load than both LORA-CBF and DSR. This also increases with distance between source and destination and depends on the amount of data delivered.



Graph 6.18: Routing Load on a multi-lane circular dual carriageway representative of motorway driving (packets).

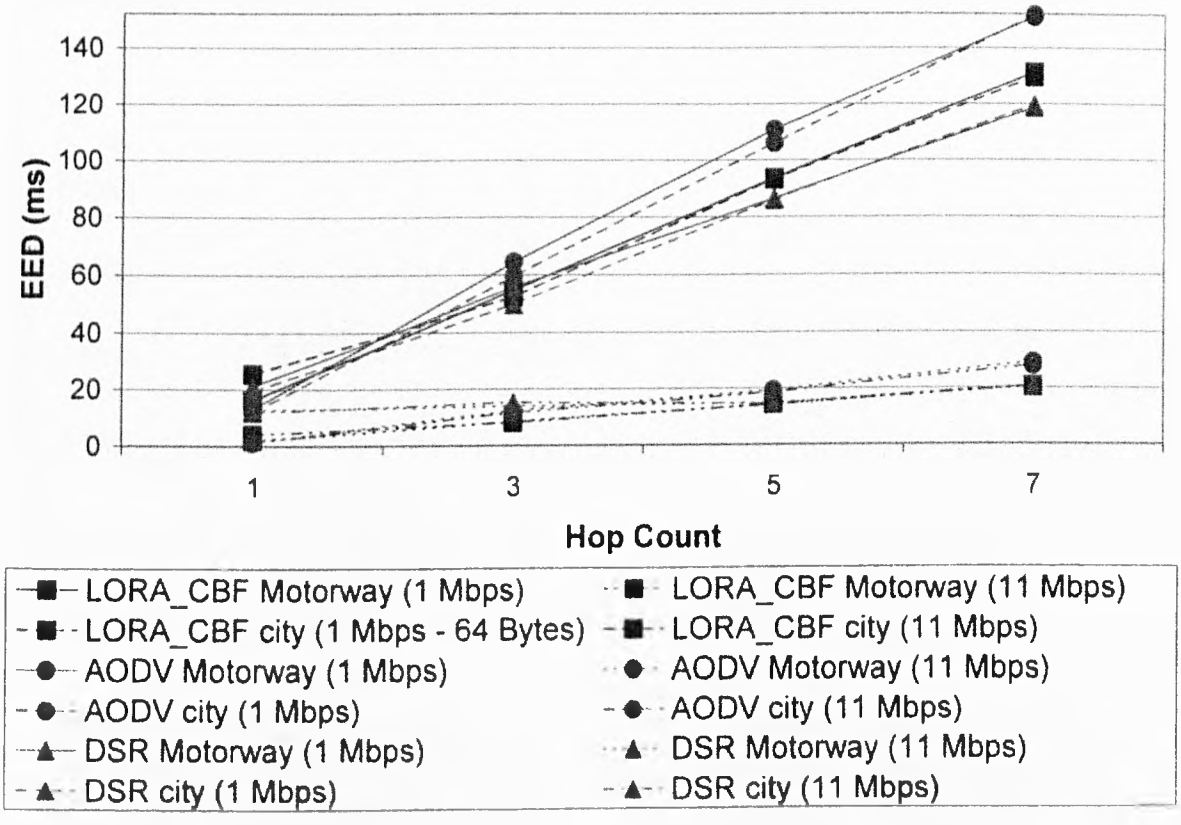
6.7.1.3 A Comparison of the algorithms LORA-CBF, AODV and DSR on a multi-lane rectangular and circular dual carriageway representative of city and motorway driving

Graph 6.19 compares the packet delivery ratio of all the algorithms considered. LORA-CBF has shown good results with both data rates and both scenarios considered. AODV has a slightly worse packet delivery ratio than DSR and it is influenced by the mobility of the vehicles. Both AODV and DSR have displayed the poorer delivery ratios with high mobility.



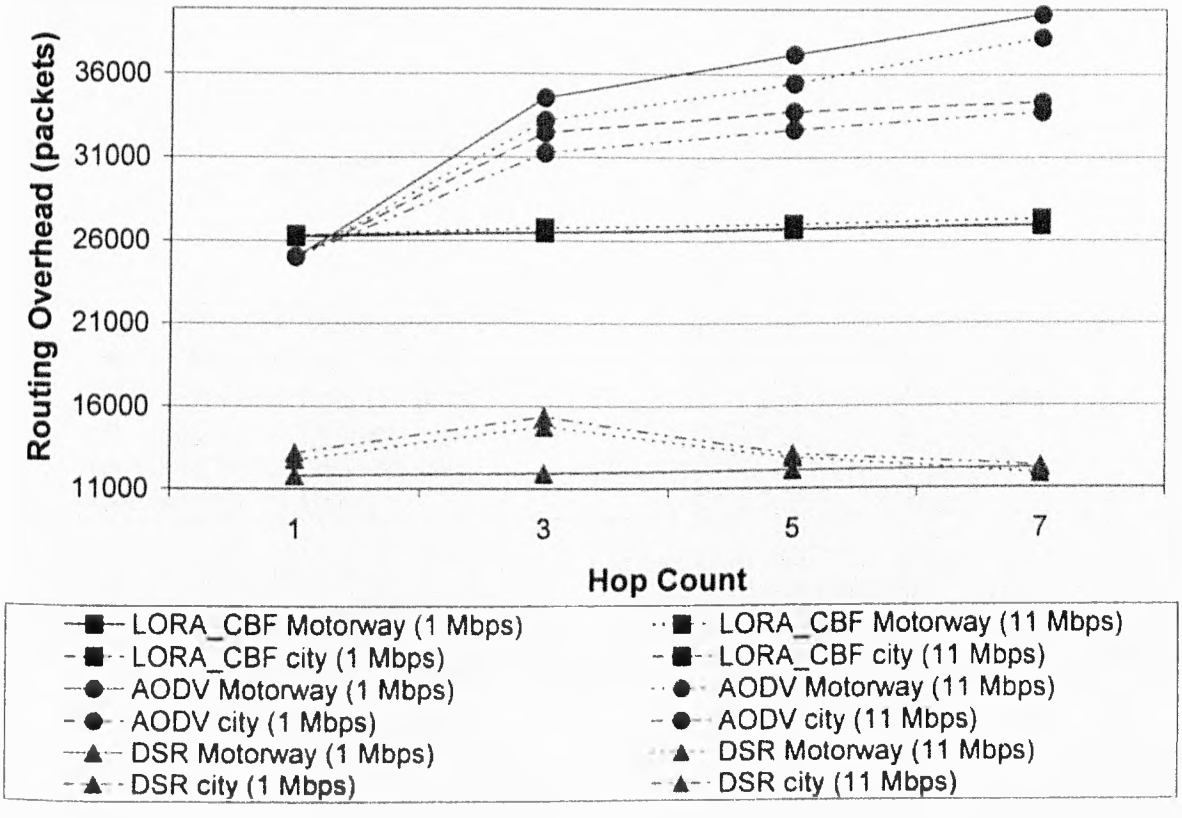
Graph 6.19: Delivery Ratio on a multi-lane rectangular and circular dual carriageway representative of city and motorway driving (packets)

End to End delay (EED) is shown in Graph 6.20. All of the algorithms have lower performance with a data rate of 1 Mbps. In general, AODV has the worse delay due its frequent retransmissions. The impact of high mobility has only a small influence on EED. DSR has been shown to have a better EED performance because of its economization of control packets while, LORA-CBF has slightly bigger EED compared with DSR.



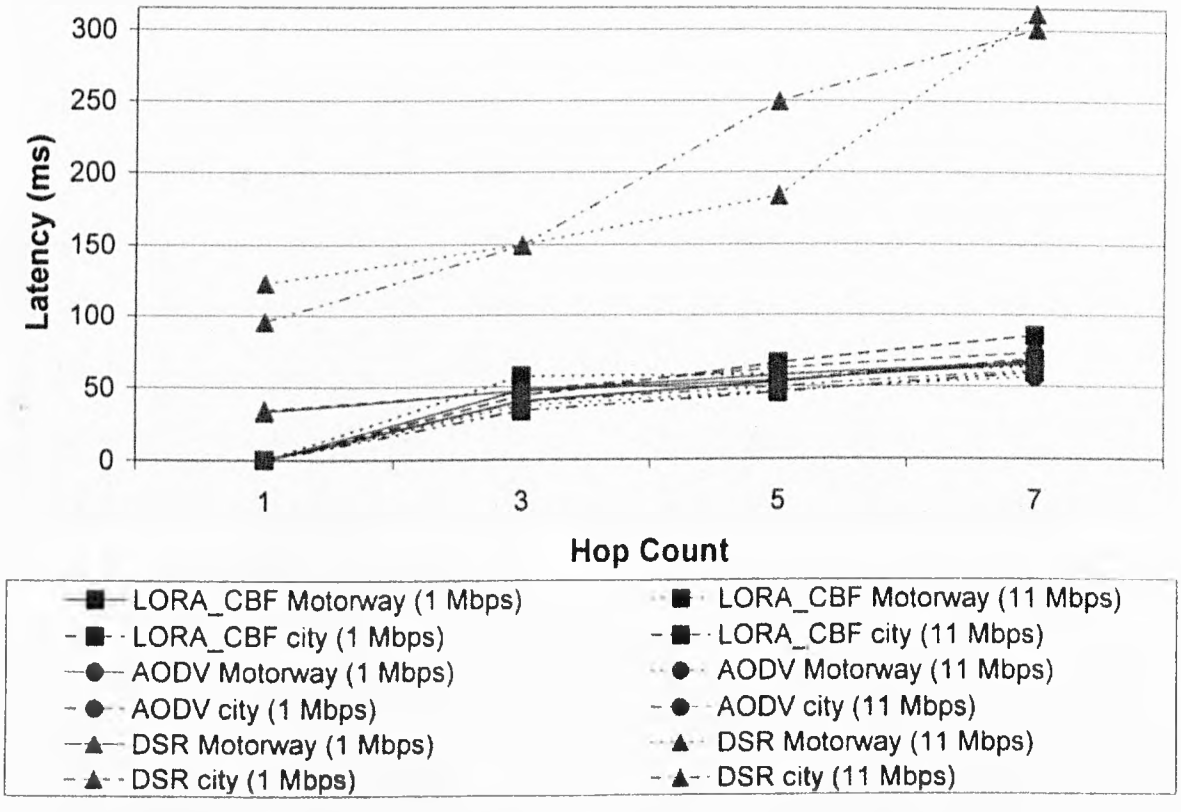
Graph 6.20: ***End-to-End Delay on a multi-lane rectangular and circular dual carriageway representative of city and motorway driving (ms).***

Graph 6.21 shows the routing overhead. Here, DSR has the better performance due to its lack of a neighbour sensing mechanism. Also, with the higher data rate, it shows a higher routing overhead in both of the scenarios that were compared. On the other hand, AODV increases its routing overhead according to the variables of distance and mobility. It has performed better performance in terms of routing overhead with the low data rate. LORA-CBF maintains its routing overhead at an almost constant level, because the level depends on the frequency of Hello messages. This is constant and independent of the maximum distance between communication partners. AODV requires about 3 times the routing overhead of DSR.



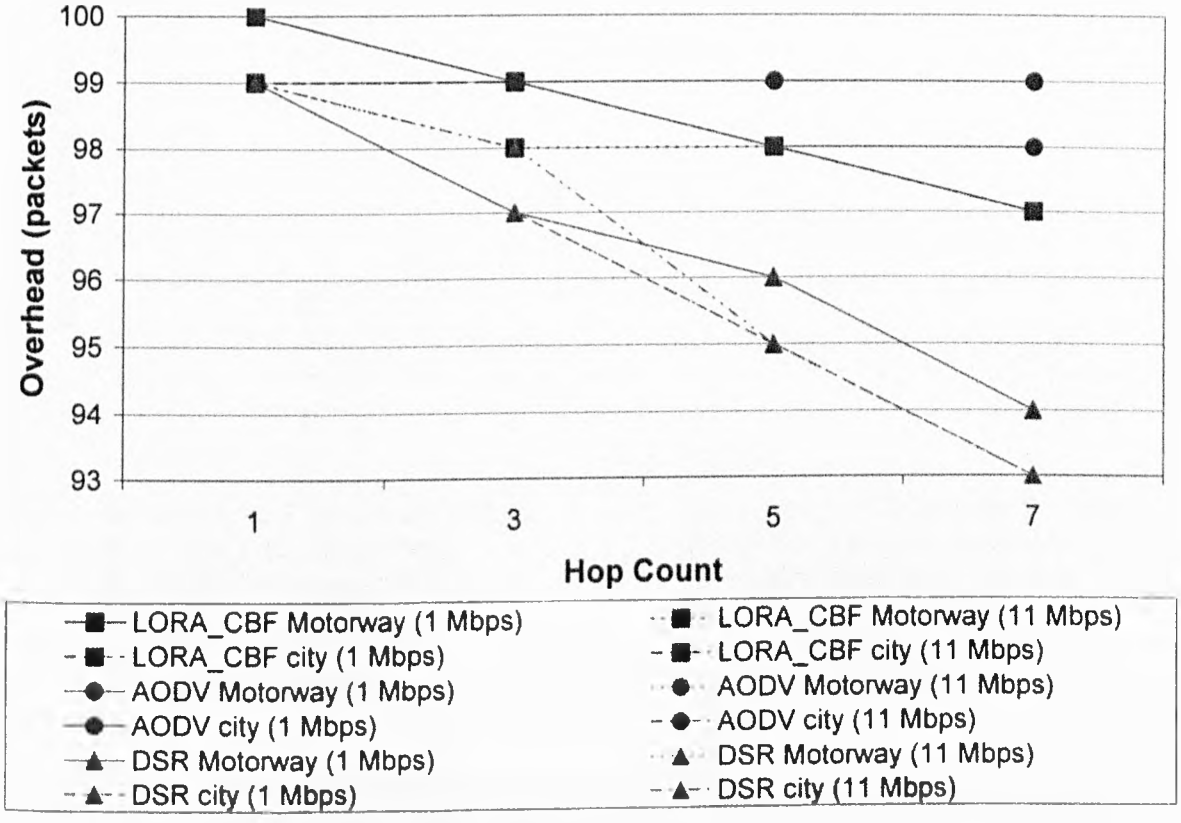
Graph 6.21: Routing Overhead on a multi-lane rectangular and circular dual carriageway representative of city and motorway driving (packets)

Graph 6.22 shows the route discovery time for all of the algorithms evaluated. DSR shows poor performance with a data rate of 11 Mbps in both scenarios evaluated. LORA-CBF and AODV have show similar performance in terms of route discovery time in both situations with both data rates.



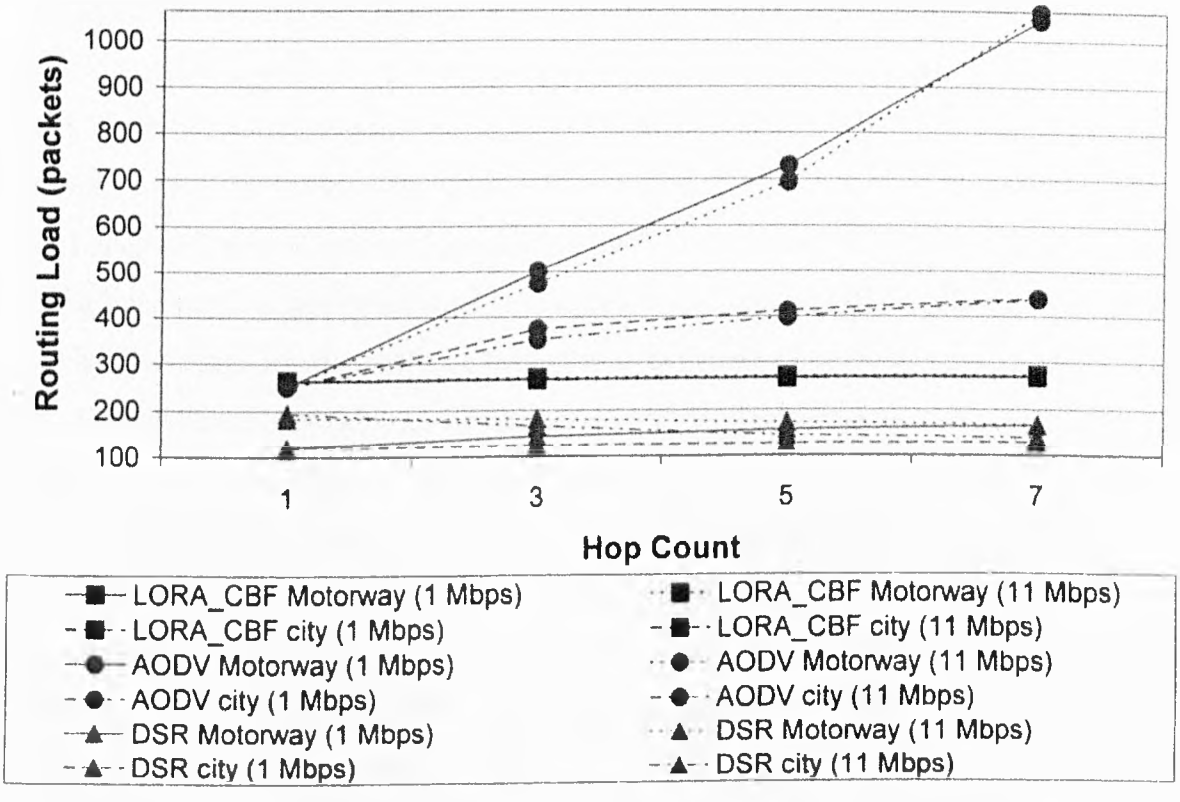
Graph 6.22: ***Route Discovery Time Overhead on a multi-lane rectangular and circular dual carriageway representative of city and motorway driving (ms)***

Overhead is shown in the Graph 6.23. Again, AODV has the poorest performance. Highly mobile environments have high link break rates and each break causes a transmission of RERR messages. In the case of AODV, the overhead is increased by the Hello messages. On the other hand, DSR has the highest performance due to its lack of transmission control. LORA-CBF has relatively good performance in spite of its complexity and robustness.



Graph 6.23: Overhead on a multi-lane rectangular and circular dual carriageway representative of city and motorway driving (packets).

Graph 6.24 represents the routing load. AODV shows more routing load than LORA-CBF and DSR. This also increases according to distance and mobility. Similarly, DSR shows better behaviour with low mobility. LORA-CBF shows similar results on urban and motorway scenarios at both data rates.



Graph 6.24: Routing Load on a multi-lane rectangular and circular dual carriageway representative of city and motorway driving (packets).

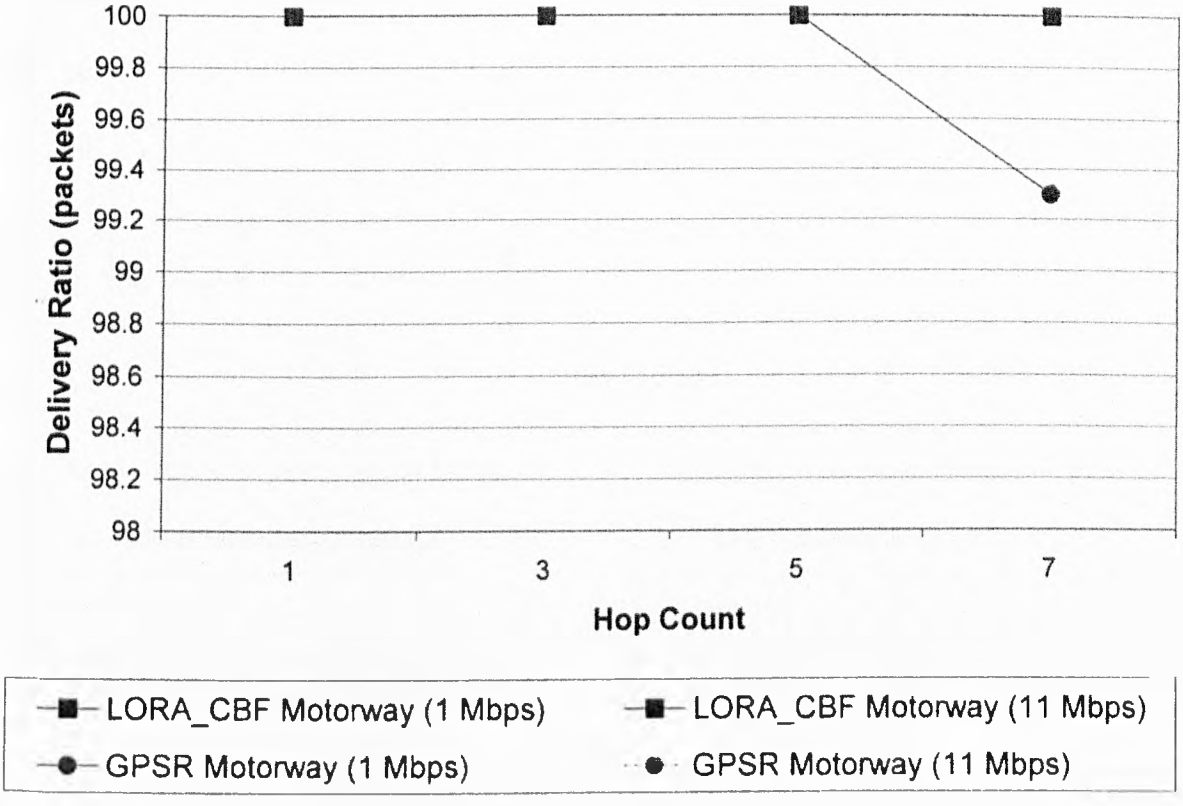
6.8 A Comparison of the LORA-CBF and GPSR algorithms on a multi-lane circular dual carriageway representative of a six lane motorway driving

The development of routing algorithms for vehicular ad-hoc networks is a challenging task because such networks have a high level of mobility and therefore, are very dynamic with respect to their links. In such networks, classical algorithms such as AODV and DSR suffer from sub-optimal routes and low delivery ratios. On the other hand, algorithms that employ Geographical Position System (GPS) do not satisfy the requirements of multi-hop vehicular ad-hoc networks. In this section, we will compare our Location-Based Routing Algorithm with Cluster-Based Flooding (LORA-CBF), with a very popular position-based routing algorithm called GPSR (Greedy Perimeter Stateless Routing) and demonstrate that at an average speed of 42 m/s (~150 km/h), the lack of a predictive algorithm in GPSR deteriorates its performance.

We have implemented the GPSR algorithm on the same circular dual carriageway scenario and with the same number of vehicles, considering a relative speed of 84 m/s (~300 km/h). Results have shown that without a predictive algorithm, it is not possible to communicate between a source-destination pair located further than two hops. Our intention here is to show that employing a predictive algorithm will improve communication on a motorway.

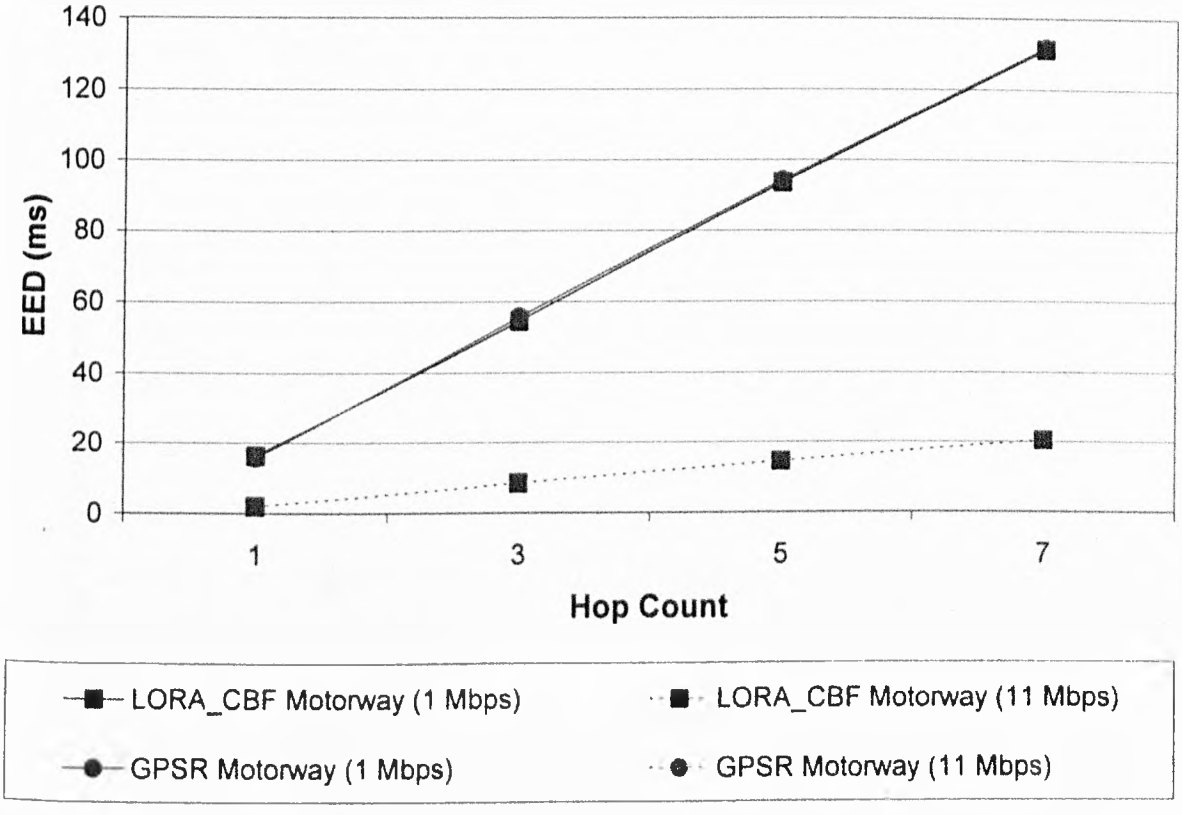
We have implemented in GPSR, the same short-term predictive algorithm used in LORA-CBF and the same physical and MAC layer. In addition, we have used the same metrics that were employed to analyze the behaviour of LORA-CBF, AODV and DSR to analyze the performance between LORA-CBF and GPSR.

Graph 6.25 represents the delivery ratio of GPSR and LORA-CBF. The short-term predictive algorithm has improved the communication of GPSR significantly (90 %). Both algorithms have similar results with data rates of 11 Mbps and GPSR has slightly lower delivery ratio with data rates of 1 Mbps. In general, both algorithms have shown very similar results because both of them use the same forwarding mechanism (greedy forwarding).



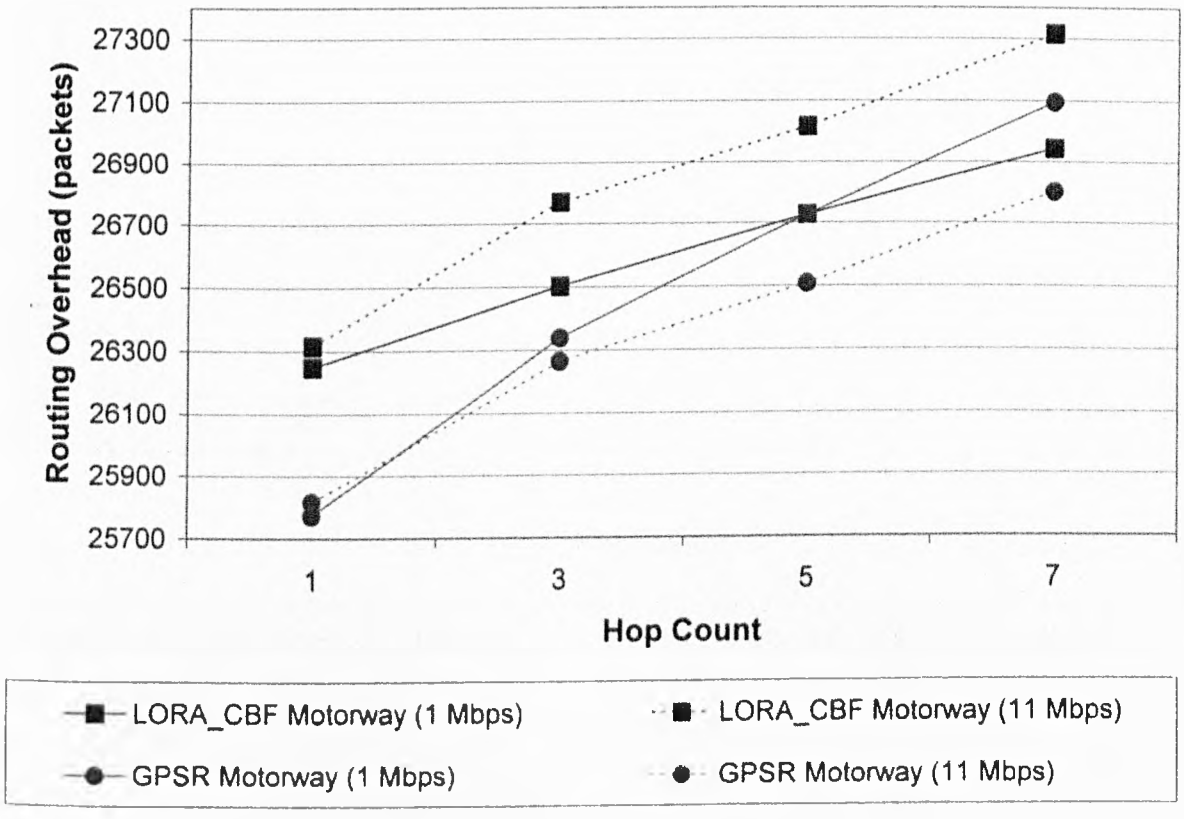
Graph 6.25: Delivery Ratio on a multi-lane circular dual carriageway representative of a six lane motorway driving (packets).

End-to-End Delay is shown in Graph 6.26. Because they employ the same forwarding mechanism, GPSR and LORA-CBF have similar behaviour in terms of EED.



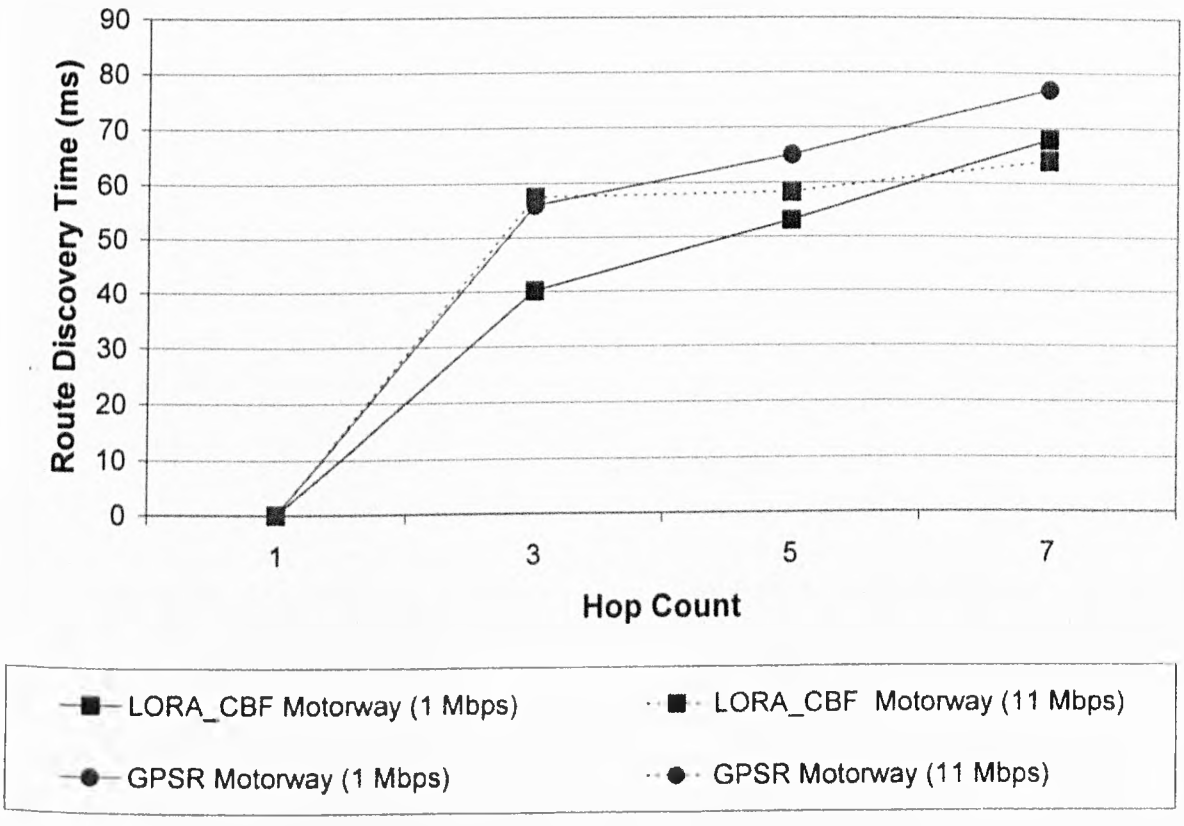
Graph 6.26: ***End-to-End Delay on a multi-lane circular dual carriageway representative of a six lane motorway driving (ms).***

Graph 6.27 describes the routing overhead. Here, LORA-CBF has a slightly higher routing overhead at a data rate of 11 Mbps compared with GPSR at the same data rate. On the other hand, at a data rate of 1 Mbps, LORA-CBF begins with a slightly higher routing overhead and a distance of 3 hops (900 m). Both algorithms show exactly the same routing overhead. At greater distances, however, LORA-CBF shows a lower routing overhead than GPSR.



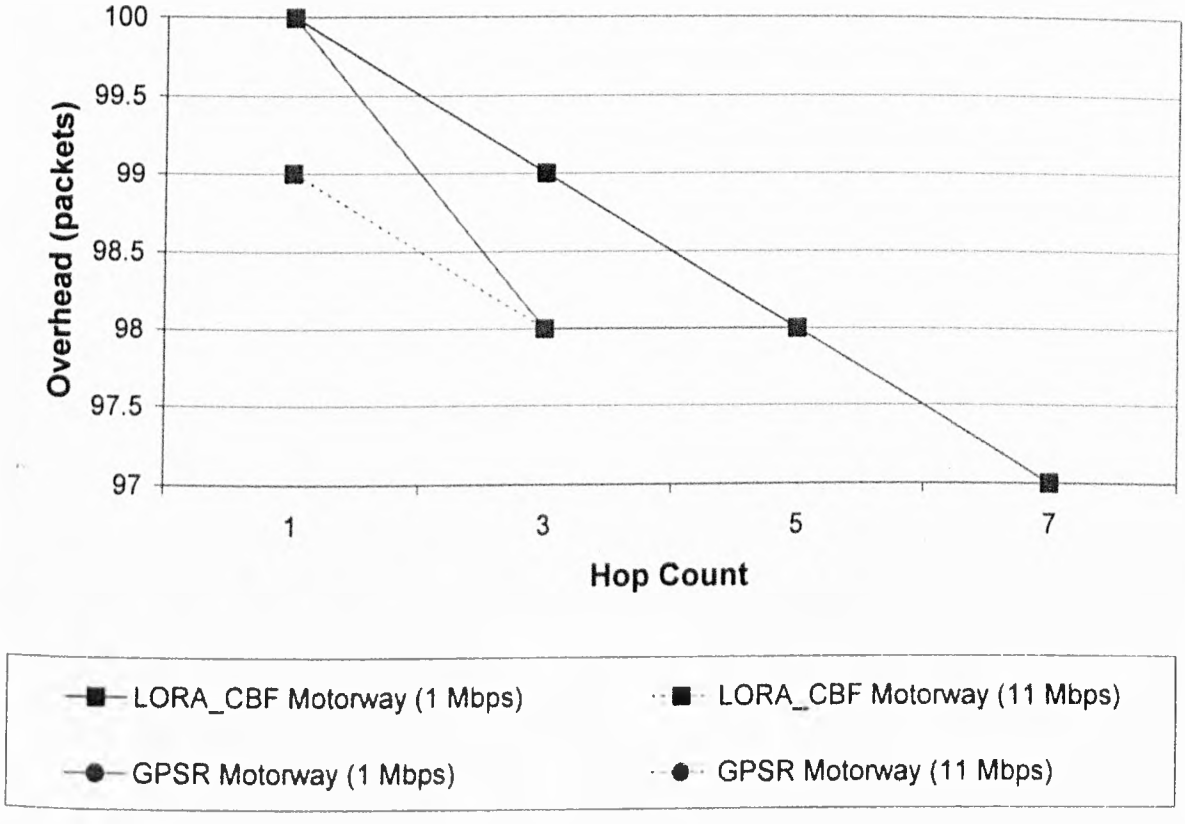
Graph 6.27: Routing Overhead on a multi-lane circular dual carriageway representative of a six lane motorway driving (packets)

Graph 6.28 represents the route discovery time. Both algorithms have shown similar behaviour at data rate of 11 Mbps, but a data rate of 1 Mbps GPSR has a greater route discovery time. Due to the limited spatial diversity, GPSR has more packet collisions. On the other hand, LORA-CBF reduces the route discovery time due to its cluster-based flooding mechanism.



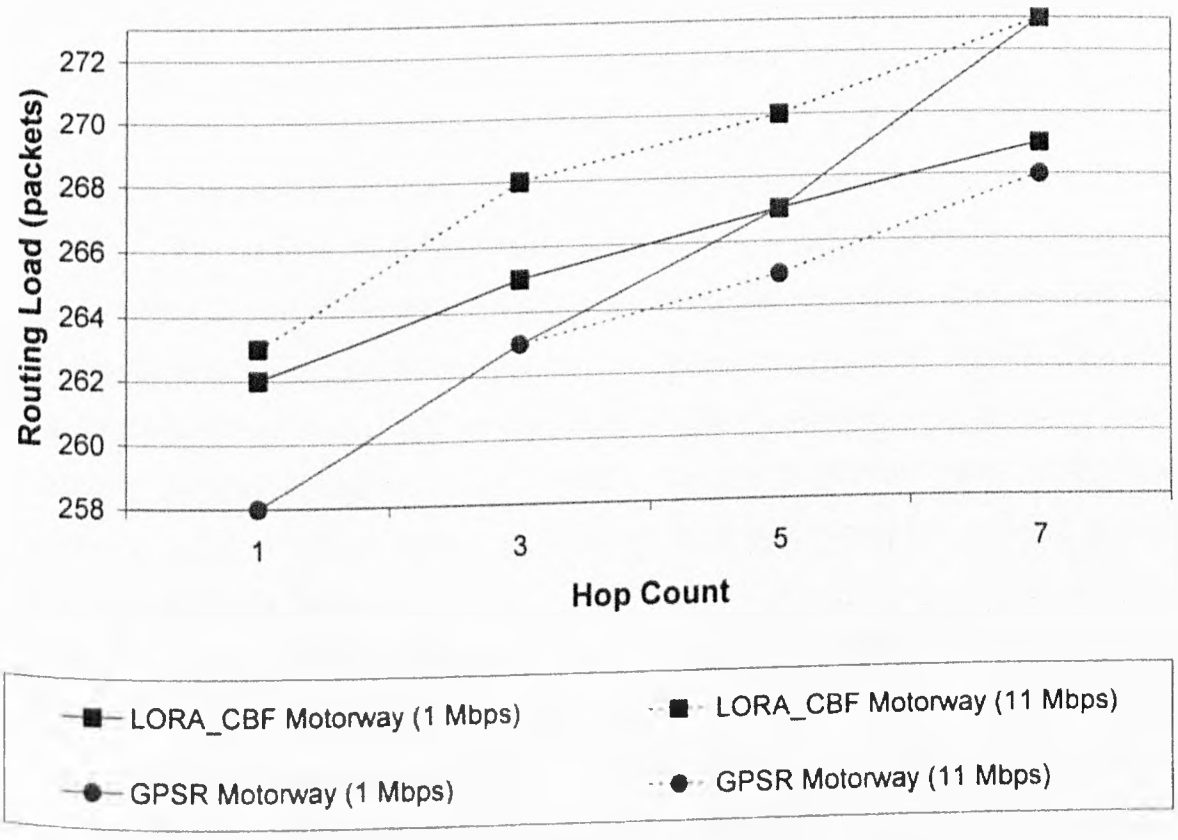
Graph 6.28: Route Discovery Time on a multi-lane circular dual carriageway representative of a six lane motorway driving (ms)

Graph 6.29 shows the Overhead. Here, GPSR has a slightly lower overhead compared to LORA-CBF at data rates of 1 Mbps. With data rates of 11 Mbps, however, both algorithms performance similarly.



Graph 6.29: ***Overhead on a multi-lane circular dual carriageway representative of a six lane motorway driving (packets)***

The routing load is shown in Graph 6.30. Again, with a data rate of 11 Mbps, LORA-CBF has a slightly higher routing load than GPSR, but with a data rate of 1 Mbps, LORA-CBF begins with a slightly higher routing load than GPSR. However, at five hops (1500 m), both algorithms have the same routing load. After this distance, LORA-CBF shows a lower routing load than GPSR. Again, due to spatial diversity, GPSR suffers more collisions that cause the increase of the routing load.



Graph 6.30: Routing Load on a multi-lane circular dual carriageway representative of a six lane motorway driving (packets)

6.9 CONCLUSIONS

In this chapter we have taken account of the mobility involved in typical urban and motorway traffic scenarios and have simulated a very large network of two hundred and fifty nodes. Our simulation model has been validated where possible, using both measurement and analysis against a control.

We have considered two non-positional-based algorithms (AODV and DSR) and one positional-based algorithm (LORA-CBF). Results show that the mobility and size of the network affects the performance of AODV and DSR more significantly in comparison with LORA-CBF. In the presence of high mobility, link failures are more common AODV and DSR. Link failures trigger new routes discoveries in all of the algorithms, but in AODV and DSR, this happens more frequently due to their routing mechanism. Thus, the frequency of route discoveries is directly proportional to the number of route breaks. We observe that Positional-based routing protocol provides an excellent performance in terms of end-to-end delay and packet delivery ratio, at the cost of using additional information (location information). Non-positional-based routing algorithms suffer from sub-optimal routes as well as worse packet delivery ratio because of more dropped packets. In addition, our Location Routing Algorithm with Cluster-Based Flooding (LORA-CBF) is robust in terms of Routing Overhead, Overhead, Routing Load and Delivery Ratio.

Also, we have compared GPSR and LORA-CBF on a motorway. We have improved GPSR with a short-term predictive algorithm to improve its performance over high speed. With the short-term predictive algorithm GPSR and LORA-CBF have shown similar behaviour. Both algorithms employ the same greedy forwarding mechanism and the neighbour sensing mechanism. Therefore, the main difference between them is the hierarchical architecture in LORA-CBF. This hierarchical architecture requires less route discovery time and a lower delivery ratio, but increases the routing overhead, and general overhead as well as routing load.

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

The challenge of wireless ad-hoc routing algorithms is the limited bandwidth and the high rate of geographical changes in highly mobile environments. In these types of environments, frequent broadcasting of routing information is necessary to maintain the routing tables updated. Limiting this distribution across the network is essential in the scalability of the wireless ad-hoc routing algorithms. On one hand, proactive algorithms keep the information of all the nodes in the wireless ad-hoc network, requiring additional control traffic to continually update route entries. Consequently, proactive protocols suffer the disadvantage of requiring significantly greater of bandwidth resources. On the other hand, reactive algorithms reduce control traffic overhead at the cost of increased latency in finding the route to a destination. Existing pure proactive and reactive algorithms are not suitable in highly mobile environments, as they result in poor route convergence and very low communication throughput. To overcome this limitation, positional algorithms have emerged that provide additional information in their routing information.

Nevertheless, positional algorithms are insufficient to cope with dense and large networks with high mobility if they do not consider any hierarchical or clustering combination. To date, the mechanism for sending information in cluster-based networks is inefficient because it constrains all traffic that must traverse cluster-heads, thereby augmenting the possibility of congestion due to traffic concentration. Each cluster-head may become a single point of failure for communication across its cluster, causing the reduction of both throughput and robustness of the network.

We have addressed several challenges in vehicular ad-hoc networks (VANET) with the Reactive Location-Based Routing Algorithm with Cluster-Based Flooding (LORA-CBF). First, using location information, we have improved packet forwarding decisions with regards to two location services: Simple and Reactive. For neighbouring nodes, a Simple Location Service has been implemented, and for faraway nodes, a Reactive Location Service is employed. Second, in geographic forwarding, the source node includes the location of its destination in each packet. Here, the packet moves hop by hop through the network, forwarded along via cooperating intermediates nodes. At each

node, a purely local decision is made to forward the packet to the neighbour that is geographically closest to the destination. However, location information by itself does not guarantee the transmission between neighbouring nodes in vehicular ad-hoc networks. Mobility and contention of wireless media may cause loss of packets, and this is a very important aspect to consider in the development of wireless routing algorithms. Here, we have addressed this problem by including a predictive algorithm in LORA-CBF. Finally, due to the increasing number of vehicles, VANET networks are foreseen to include hundreds or thousands of vehicles in their architecture. This tendency motivates us to design an algorithm that can cope with a very dense and highly mobile network. What we have developed here divides the size of VANET networks into several clusters, thus improving their scalability.

In this thesis, we have shown that IEEE 802.11b wireless networks are suitable for inter-vehicle communication. We have supported our hypothesis with the results of two propagation models. On one hand, according to large scale models, the maximum distance between the transmitter and the receiver is 446 m. In addition, the System Operating Margin (SOM) feasible at 446 m is over 13 dB. The SOM is over the minimum margin recommended. On the other hand, we have found that the Doppler Effect does not affect the communication between communication partners at high speed in small scale models. Finally, we have realized an experiment that has allowed us to validate the former results in the worst case scenario, when the transmitter and receiver are travelling in opposite directions. Results have shown that at least 8 packets are possible when the transmitter and receiver are within communication range.

Since the appearance of the 1985 Highway Capacity Manual, there has been an increasing amount of research on traffic flow models, which has led to a different understanding of how traffic operates, especially on a motorway. Efforts to implement ITS with regard to both traffic management and traffic information provision will provide challenges for applying this improvement to new microscopic traffic models.

In this thesis, we have analyzed six microscopic traffic models. These microscopic traffic models are based on three types of car-following models: safe-distance models, stimulus-response and psycho-physiological models. Of the studied models VISSIM contains the largest number of parameter and AIMSUM is the model with the smallest number of parameters. The main drawback of MIMIC, INTEGRATION and VISSIM

models are the calibration parameters. These parameters have to be calibrated during the simulation period. The AIMSUM model does not define clearly how to estimate the maximum deceleration of the vehicle n-1 and MITSIM bases its acceleration or deceleration rate to many speed intervals.

The Simone 2000 traffic simulation model seems to be more suitable for our simulation scenarios. It bases its behaviour on two different controllers: *Distance Controller and Longitudinal Controller*. In addition, it incorporates the acceleration/deceleration effect when the relative speed stimulus is positive or negative. We have used this microscopic traffic model to simulate the mobility of the vehicles on a multi-lane rectangular and circular dual carriageway representative of city and motorway driving.

In the last part of this thesis, we have taken into account the mobility involved in typical urban and motorway traffic scenarios and have simulated a very large network of two hundred and fifty nodes. Our simulation model has been validated, where possible, using both measurement and analysis against a control.

We have considered two non-positional-based algorithms (AODV and DSR) and one positional-based algorithm (LORA-CBF). Results show that the mobility and size of the network affects the performance of AODV and DSR more significantly in comparison with LORA-CBF. In the presence of high mobility, link failures are more common. Link failures trigger new route discoveries in all of the algorithms, but in AODV and DSR, this happens more frequently due to their routing mechanism. Thus, the frequency of route discoveries is directly proportional to the number of route breaks. We observe that Positional-based routing protocol provides an excellent performance in terms of end-to-end delay and packet delivery ratio, at the cost of using additional information (location information). Non-positional-based routing algorithms suffer from sub-optimal routes as well as worse packet delivery ratio because of more dropped packets. In addition, our Location Routing Algorithm with Cluster-Based Flooding (LORA-CBF) is robust in terms of Routing Overhead, Overhead, Routing Load and Delivery Ratio.

Also, we have compared GPSR and LORA-CBF on a motorway. We have improved GPSR with a short-term predictive algorithm to improve its performance over high speeds. With the short-term predictive algorithm, GPSR and LORA-CBF have shown

similar behaviour. Both algorithms employ the same greedy forwarding mechanism and neighbour sensing mechanism. Therefore, the main difference between them is the hierarchical architecture of LORA-CBF. This hierarchical architecture requires less route discovery time and a lower delivery ratio, but increases the routing overhead and general overhead as well as routing load.

One of our future research lines is in the direction of multi-cast algorithms, where we pretend to extend the capabilities of LORA-CBF towards group communication, which might have application in military and educational settings. Here, multicast algorithms maintain paths between a single source and multiple destinations, which appears more difficult than unicast algorithms, but not excessively, so. Given the growing importance of multicast as a means to reduce the bandwidth utilization for mass distribution of data, and the pressing need to conserve scarce bandwidth over wireless media, it is natural that multicast routing should receive some attention for ad-hoc networks.

Another future area is in the direction of sensor networks. Here, recent attention has been focused on ideas involving the possibility of coordinating the activities and reports of a large collection of small sensor devices. Such devices, are inexpensive to manufacture and can to be spread in large numbers of identical units to offer detailed information about terrain or dangerous environmental conditions. We hope to apply our cluster-based algorithm to minimize the transmission in a determined zone. This might be applicable, for example, in danger zones such as volcanic hazard areas.

Lastly, we also are investigating the possibility of employing our algorithm as a possible option that might contribute to the convergence of wireless ad-hoc and cellular networks. Fourth generation wireless networks are all about integrating a global network based on an open-system approach that will seamlessly integrate different types of wireless networks with wireline backbone networks to promote the convergence of voice, multimedia, and data traffic over a single IP-based core network. With the availability of ultrahigh bandwidth of up to 100 Mbps, multimedia services can be supported efficiently. Ubiquitous computing will be facilitated with enhanced system mobility and portability support, with location-based services and support of ad-hoc networking.

LIST OF REFERENCES

[1] Jyrki Oraskari: Bluetooth versus WLAN IEEE 802.11x. Product Modelling and Realization Group (PM&RG), Department of Computer Science and Engineering, Helsinki University of Engineering.

[2] Brent A. Miller and Chatschik Bisdian. Bluetooth Revealed. Prentice-Hall, 2001.

[3] David Kammer, Gordon McNutt, Brian Senese and Jennifer Bray. Bluethooth, Application Developer's Guide: The Short Range Interconnect Solution.

[4] Bob O'Hara and Al Petrick. *The IEEE 802.11 Handbook: A Designer's Companion.* Standards Information Network. IEEE Press, 1999.

[5] Broadcom. IEEE 802.11g. White Paper, February, 2003.

[6] Andrew S. Tanenbaum. *Computer Networks*, third edition. Prentice-Hall International, Inc. 1996.

[7] C-K Toh. *Ad Hoc Mobile Wireless Networks: Protocols and Systems*. Prentice-Hall International, Inc. 2002.

[8] Carl-Herbert Rokitansky. SIMCO2: Simulator for Performance evaluation of Vehicle-Beacon and Inter-vehicle Communication Protocols (Media Access/ Knowledge-Based Routing). In IEEE Vehicular Technology Conference, pp. 893-899, 1991.

[9] Carl-Herbert Rokitansky. Performance Analysis of Adaptive Multi-hop Routing Protocol Using a Markov Model. Proceedings IEEE Vehicular Technology Conference, 1992.

[10] Carl-Herbert Rokitansky and Christian Wietfeld. Comparison of Adaptive Medium Access Control Schemes for Beacon-Vehicle Communications. IEEE-IEE Vehicle Navigation & Information Systems Conference, 1993.

[11] G. Brasche, C. –H. Rokitansky, and C. Wietfeld. Communication Architecture and Performance Analysis of Protocols for RTT Infrastructure Networks and Vehicle-Roadside Communications. IEEE 44th Vehicular Technology Conference, 1994.

[12] Christian Wietfeld and Carl-Herbert Rokitansky. Performance of Vehicle-Roadside Communication Systems supporting Multiple RTI-Applications. Proceedings in Vehicle Navigation & Information Systems Conference, 1994.

[13] Carl-Herbert Rokitansky and Christian Wietfeld. Methods and Tools for Performance Evaluation and Validation of Vehicle-Roadside Communication Proposed for Standarization. In Vehicular Technology Conference, vol 2, pp. 964-970, 1995.

[14] Christian Wietfeld and Carl-Herbert Rokitansky. Markov Chain Analysis of Alternative Medium Access Control Protocol for Vehicle-Roadside Communications. In Vehicular Technology Conference, vol. 2, pp 958-963, 1995.

[15] Ioan Chisalita and Nahid Shahmehri. A Novel Architecture for Supporting Vehicular Communication. IEEE 56th Vehicular Technology Conference, 2002.

[16] Robert Morris, John Jannotti, Frans Kaashock, Jinyang Li and Douglass Decouto. CarNet: A ScalableAd Hoc Wireless Network System. In Proceedings of the 9th ACM SIGOPS European workshop: Beyond the PC: New Challenges for the Operating System, Kolding, Denmark, September 2000.

[17] Zong Da Chen, HT Kung and Dario Vlah. Ad Hoc Relay Wireless Networks over Moving Vehicles on Highways. International Conference on Mobile Computing and Networking, MobiCom 2001.

[18] Holger Füßler, Martin Mauve, Hannes Hartenstein, Michael Käsemann and Dieter Vollmer. MobiCom Poster: Location-Based Routing for Vehicular Ad-Hoc Networks. ACM SIGMOBILE Mobile Computing and Communication Review, volume 7 issue 1, 2003.

[19] http://www.fleetnet.de.

[20] Christian Lochert, Holger Füßler, Hannes Hartenstein, Dagmar Hermann, Jiang Tian and Martin Mauve. A Routing Strategy for Vehicular Ad Hoc Networks in City Environments. IEEE Intelligent Vehicles Simposium, 2003.

[21] Timo Kosh, Christian Schwingenschlögl, and Li Ai. Information Dissemination in Multihop Inter-vehicle Networks – Adapting the Ad-hoc On-demand Distance Vector Routing Protocol (AODV). 5th International Conference on Intelligent Transportation System, 2002.

[22] http://www.comnets.rwth-aachen.de/~ftp-wg9/

[23] http://www.williamson-labs.com/ivhs.htm

[24] http://www.cartalk2000.net

[25] Charles E. Perkins. Ad hoc networking. Addison Wesley. 2000.

[26] Thomas Clausen, Philippe Jacket, Anis Laouiti, Pascale Minet, Paul Muhlethaler, Amir Qayyum,

Laurent Viennot. Optimized Link State Routing Protocol (OLSR). http://www.ietf.org/rfc/rfc3626.txt, Ocotber 2003. Request for Comments (Work in Progress).

[27] Richard G. Ogier, Mark G. Lewis, Fred L. Templin. Topology Dissemination based on Reverse-Path Forwarding (TBRPF). http://www.ietf.org/rfc/rfc3684.txt, February 2004. Request for Comments (Work in Progress).

[28] Xukai Zou, Byrav Ramamurthy and Spyros Magliveras. Routing Techniques in Wireless Ad Hoc Networks –Classification and Comparison. Proceedings of the Sixth World Multiconference on Systemics, Cybernetics and Informatics, SCI, July 2002.

[29] Charles E. Perkins, Elizabeth M. Belding-Royer, Samir R. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. http://www.ietf.org/rfc/rfc3561.txt, July 2003. Request for Comments (Work in Progress).

[30] David B. Johnson, David A. Maltz, Yih-Chun Hu. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt, July 2004. IETF Internet Draft (Work in Progress).

[31] Jan Schaumann. Analysis of the Zone Routing Protocol. http://www.netmeister.org/misc/zrp/zrp.pdf, December 2002.

[32] Martin Mauve, Jörg Widmer and Hannes Hartenstein. A Survey on Position-Based Routing in Mobile Ad-Hoc Networks. IEEE Network Magazine, 15(6): 30-39, November 2001.

[33] Young-Bae Ko, Nitin H. Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. Proceedings of the 4th annual ACM/IEEE International Conference on Mobile Computing and Networking, pp 66-75, 1998.

[34] Stefano Basagni, Imrich Chalamtac, Violet R. Syrotiuk. A Distance Routing Effect Algorithm for Mobility (DREAM). MOBICOM 98. Dallas Texas USA.

[35] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, Robert Morris. A Scalable Location Service for Geographic Ad Hoc Routing. ACM Mobicom 2000, Boston, MA.

[36] Brad Karp, H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000).

[37] Alina Bejan and Ramon Lawrence. Peer-to-Peer Cooperative Driving. Seventeenth International Symposium On Computer and Information Sciences (ISCIS XVII), 2002.

[38] http://www.ietf.org/html.charters/manet-charter.html

[39] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A Qayyum, and L. Viennot. Optimized Link State Routing Protocol for Ad Hoc Networks. IEEE INMIC Pakistan 2001.

[40] Anis Laouti, Paul Mühlethaler, Abdellah Najid, Epiphane Plakoo. Simulation Results of the OLSR Routing Protocol for Wireless Network. 1st Mediterranean Ad-Hoc Networks Workshop (Med-Hoc-Net), Sardegna, Italy 2002.

[41] Philippe Jacquet, Anis Laouiti, Pascale Minet and Laurent Viennot. Performance of multipoint relaying in ad hoc mobile routing protocols. Networking 2002, Pise (Italy) 2002.

[42] Amir Qayyum, Laurent Viennot and Anis Laouiti. Multipoint Relaying for Flooding Broadcast Messages in Mobile Wireless Networks. 35th Annual Hawaii International Conference on System Sciences (HICSS'2002).

[43] Sally Floyd and Van Jacobson. The Synchronization of Periodic Routing Messages. IEEE/ACM Transaction on Networking, vol. 2 issue 2, pp 122-136, 1994.

[44] Thomas Heide Clausen, Gitte Hansen, Lars Christensen and Gerd Behrmann. The Optimized Link State Routing Protocol Evaluation through Experiments and Simulation. IEEE Symposium on "Wireless Personal Mobile Communications", September 2001.

[45] Mounir Benzaid, Pascale Minet and Khaldoun Al Agha. Integrating fast mobility in the OLSR routing protocol. Fourth IEEE Conference on Mobile and Wireless Communication Networks (MWCN), Stockholm Sweden, September 2002.

[46] Mounir Benzaid, Pascale Minet and Khaldoun Al Agha. Analysis and simulation of Fast-OLSR. Proceedings of the 57th IEEE Semi-annual Vehicular Technology Conference (VTC), Jeku Korea, April 2003.

[47] http://www.erg.sri.com/projects/tbrpf/

[48] Elizabeth M. Royer and Chai-Keong Toh. A review of current routing protocols for ad hoc wireless networks. IEEE Personal Communications, pages 46-55, April 1999.

[49] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc networks routing protocols. In Proceedings of the 4th ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), pages 85-97, Dallas, TX, USA,

1998.
[50] http://moment.cs.ucsb.edu/AODV/aodv.html
[51] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc On-Demand Distance Vector Routing. Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, February 1999, pp. 90-100.
[52] http://www.cs.rice.edu/Database/dbj.shtml
[53] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. Computer Science Department, Carnegie Mellon University.
[54] David B. Johnson, David A. Maltz and Josh Broch. DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks.
[55] Avinash Kashyap, Hitendra Nishar and Parag Agarwal. Survey on Unicast Routing in Mobile Ad Hoc Networks. Computer Science Department. University of Texas at Dallas, TX.
[56] Samir R. Das, Robert Castañeda and Jiangtao Yan. Simulation Based Performance Evaluation of Mobile, Ad hoc Network Routing Protocols. *ACM/Baltzer Mobile Networks and Applications (MONET) Journal,* July 2000, pages 179-189.
[57] Charles E. Perkins, Elizabeth M. Royer, Samir R. Das and Mahesh K. Marina. Performance Comparison of two On-Demand Routing Protocols for Ad Hoc Networks. IEEE Personal Communication Magazine special issue on Ad hoc Networking, February 2001, p. 16-28.
[58] Young-Bae Ko and Nitin H. Vaidya. Location-Aided Routing (LAR) in mobile ad hoc networks. Wireless Networks, pp 307-321, 2000.
[59] Silvia Giordano, Ivan Stojmenovic and Ljubica Blazevic. Position Based Routing Algorithms for Ad Hoc Networks: A Taxonomy. Ad Hoc Wireless Networking, 2003.
[60] Elizabeth M. Belding-Royer. Multi-Level Hierarchies for scalable Ad-hoc Routing. Wireless Networks, pp 461-478, 2003.
[61] Dennis J. Baker, Anthony Ephremides and Julia A. Flynn. The Design and Simulation of a Mobile Radio Network with Distribute Control. IEEE Journal on Selected Areas in Communication, 1984.
[62] Mario Gerla and Jack Tzu-Chieh Tsai. Multicluster, mobile, multimedia radio network. Baltzer Journals, 1995.
[63] Mingliang Jiang, Jinyang Li, Y.C. Tay. Cluster Based Routing Protocol (CBRP). http://www.ietf.org/internet-drafts/draft-ietf-manet-cbrp-espec-01.txt, July 1999. IETF Internet Draft (Work in Progress).
[64] Michael J. Chu and Wayne E. Stark. Effect of Mobile Velocity on Communications in Fading Channels. IEEE Transactions on Vehicular Technology, vol. 49, no. 1, 2000.
[65] Michael Käsemann, Hannes Hartenstein, Holger Füßler, and Martin Mauve. Analysis of a Location Service for Position-Based Routing in Mobile Ad Hoc Networks. Proceedings of the 1st German Workshop on Mobile Ad Hoc Networks (WMAN 2002).
[66] Tracy Camp, Jeff Boleng and Lucas Wilcox. Location Information Services in Mobile Ad Hoc Networks. International Communication Conference (ICC), 2002.
[67] [19] http://msl1.mit.edu/Mar2Lecture/Use_Safety.htm
[68] Boris Mitelman, Arkady Zaslavsky. Link State Routing Protocol with Cluster Based Flooding for Mobile Ad-hoc Computer Networks. Proceedings of the Workshop on Computer Science and Information Technologies CSIT'99.
[69] P. Krishna, N. H. Vaidya, M. Chatterjee and D. K. Pradhan. A Cluster-Based Approach for routing in Dynamic Networks. ACM SIGCOMM, Computer Communication Review, pages 49-65, 1997.
[70] Bevas Das, Raghupathy Sivakumar, and Vaduvur Bharghavan. Routing in Ad Hoc Networks Using a Spine. Proceedings in International Conference in Computer and Communication Networks, 1997.
[71] Raghupathy Sivakumar, Bevas Das and Vaduvur Bharghavan. Spine Routing in Ad Hoc Networks. ACM/Baltzer Cluster Computing Journal, 1998.
[72] Ching-Chuan Chiang, Hsiao-Kuang Wu, Winston Liu and Mario Gerla. Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel. The IEEE Singapore International Conference on Networks, pp 197-211, 1997.
[73] Vilalta R. ET AL. Predictive algorithms in the Management of Computer Systems, IBM Systems Journal, Vol 41, No 3, 2002.
[74] Theodore S. Rappaport. Wireless Communications: Principles and Practice, Prentice Hall Communications Engineering and Emerging Technologies Series, 2002.
[75] http://www.enterasys.com/products/wireless/CSIxD-AA/
[76] http://www.ispplanet.com/fixed_wireless/equipment/2001/dMystifying_dB.html
[77] Jatinder Pal Singh, Nicholas Bambos, Bashkar Srinivasan and Detlef Clawin. Wireless LAN

Performance under Varied Stress Condition in Vehicular Traffic Scenarios. IEEE Vehicular Technology Conference, fall 2002, Vol 2, pp 743-747, Vancouver.
[78] David A. Maltz, Josh Broch and David B. Johnson. Lessons from a Full-Scale Multihop Wireless Ad Hoc Network Testbed. IEEE Personal Communication, 2001.
[79] L. B. Michel, et. al. DS/SS Inter-Vehicle Communication Experiments in 2.4 GHz ISM Band. IEEE International Conference on Intelligent Vehicles, 1998.
[80] http://www.roke.co.uk/download/datasheets/dtt-dab_receiver.pdf
[81] Serge P. Hoogendoorn, Piet H. L. Bovy. State-of-the-art of Vehicular Traffic Flow Modelling. Special Issue on Road Traffic Modelling and Control of the Journal of System and Control Engineering.
[82] Demetrio Carmine Festa, Giovanni Longo, Gabriella Mazzulla and Giuseppe Musolino. Experimental analysis of different simulation models for motorway traffic flow. Proceedings of the IEEE Intelligent Transportation Systems Conference, 2001.
[83] Bobi Cvetkovski and Liljana Gavrilovska. A simulation of a mobile Highway traffic. IEEE VTC, 1998.
[84] Bobi Cvetkovski and Liljana Gavrilovska. A simulation of a mobile traffic on a highway. Series Electronics and Energetics Vol. 11, No 1, pp 57-70, 1998.
[85] Angela Di Febbraro and Antonella Ferrara. A new two-level model for multiclass Freeway traffic. IEEE Transactions on vehicular technology, Vol 45, No 1, 1996.
[86] Fu-Sheng Ho and Petros Ioannou. Traffic Flow Modeling and Control using Artificial Neural Networks. IEEE Control System Magazine, Vol 16, No 5, pp 16-27, 1996.
[87] Gabriel Montenegro, Masakazu Sengoku, Yoshio Yamaguchi and Takeo Abe. Time-Dependent Análisis of Mobile Communication Traffic in a Ring-Shaped Service Area with Nonuniform Vehicle Distribution. IEEE Transactions on Vehicular Technology, Vol. 41, No 3, 1992.
[88] Kin K. Leung, William A. Massey and Ward Whitt. Traffic Models for Wireless Communication Networks. IEEE Journal on selected areas in communications, Vol. 12, No. 8, 1994.
[89] Johan Janson Olstam and Andreas Tapani. Comparison of car-following models. Swedish National Road and Transport Research Institute (VTI), 2004.
[90] Logghe S. Dynamic Modelling of Heterogeneous Vehicular Traffic. Ph. D. Thesis, K. U. Leuven, Belgium, 2003.
[91] Axel Klar, Reinhart D. Kühne and Raimund Wegener. Mathematical Models for Vehicular Traffic. Surveys on mathematics for industry, 6:215-239, 1996.
[92] Stig O. Simonsson. Car-Following as a tool in road traffic simulation. IEEE-IEE Vehicle Navigation & Information System Conference, 1993.
[93] Jaime Barceló et al. Smartest Simulation Report, http://www.its.leeds.ac.uk/smartest
[94] M. Van Aerde, B. Hellinga, M. Baker and H. Rakha. INTEGRATION: An Overview of Traffic Simulation Features, Transportation Research Board Annual Meeting, 1996.
[95] R. Hoyer, M. Fellendorf. Parametrization of Microscopic Traffic Flow Models through image processing. 8th IFAC Symposium on Transport, Chania, Crete, June 1997.
[96]M. M. Minderhoud. *Simone 2000, simulation model of motorways with next generation vehicles, technical specification.* March 2002.
[97] Fred L. Hall. Traffic Stream Characteristics. http://www.tfhrc.gov/its/tft/tft.htm, June, 1992.
[98] Richard W. Rothery. Car Following Models. http://www.tfhrc.gov/its/tft/tft.htm, June, 1992.
[99] Mohammad Llyas. *The Handbook of Ad Hoc Wireless Networks, CRC Press, 2002.*
[100] Erol A. Peköz and Nitindra Joglekar. Poisson Traffic Flow in a General Feedback Queue. Journal of Applied Probability, September 2002.
[101] M. Rudack, M. Meincke and M. Lott. On the Dynamic of Ad-hoc Networks for Inter Vehicle Communications (IVC). ICWN'02, Las Vegas USA, 2002.
[102] http://stat.tamu.edu/stat30x/notes/node70.html.
[103] Padma Panchapakesan and D. Manjunath. On the Transmission Range in Dense Ad Hoc Radio Networks, Conference on Signal Processing and Communications, SPCOMM, 2001.
[104] Mattias Grossglauser and David Tse. Mobility Increases the Capacity of Ad-Hoc Wireless Networks, in proc. of the IEEE infocom, 2001.
[105] Piyush Gupta and P. R. Kumar. The Capacity of Wireless Networks, IEEE Trans. on information theory, vol 46, no 2, pp 388-404, 2002.
[106] Suhas N. Diggavi, Matthias Grossglauser and David N. C. Tse. Even one-Dimensional Mobility Increases Ad Hoc Capacity, ISIT 2002.
[107] J. W. Schmidt. Fundamental of Digital Simulation Modelling. Proceedings of the Winter

Simulation Conference. pp 13- 21, 1981.
[108] Robert G. Sargent. Validation and Verification of Simulation Models. Proceedings of the Winter Simulation Conference, 1992.
[109] http://www.antd.nist.gov/wahn_home.shtml
[110] Charles E. Perkins and Pravin Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for mobile computers. Proceedings of the conference on Communication Architectures, Protocols and Applications, pp 234-244, SIGCOM 94.
[111] http://enterprise16.opnet.com/4dcgi/Models_SearchSubmit
[112] C. –K. Toh, Minar Delwar and Donald Allen. Evaluating the communication Performance of an Ad-Hoc Wireless Network. IEEE Transaction on Wireless Communication, vol. 1, no, 3, 2002.

APPENDICES

A. STRUCTURES USED IN LORA-CBF ALGORITHM

This Appendix describes the structures used in LORA-CBF algorithm. The specification provided here has been written in proto-C, a combination of standard C language notation and OPNET functions that can be employed as an implementation guideline.

A.1 Data structures used in LORA-CBF

A.1.1 Cluster Table

Each cluster-head maintains a routing table of its members with the following attributes:

- Node Address.
- Type of node (Member or Gateway).
- Location (Latitude and Longitude).

A.1.2 Cluster Neighbour Table

Each cluster-head maintains a routing table with the information of its Cluster-Head Neighbours and the gateways through which the Cluster-Head can be reached.

- Adjacent Cluster-head Address.
- Adjacent Cluster-head Location.
- Gateway or Gateways (Address and Location).

A.1.3 Hello Message

The Hello message is broadcasted by the cluster-head according to a specific period of time (Figure A.1). The attributes of the Hello message are:

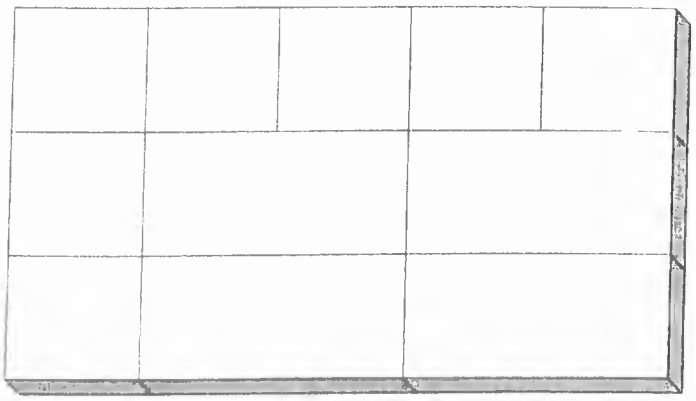


Figure A.1: Hello Message.

Type: This field determines the type of the packet that has been sent (Hello, Location Request, Location Reply, Data, Acknowledgement).

Type of node: This field determines the type of the node that has sent the message, i.e. Undecided, Member, Gateway or Cluster-head.

Hop Count: This field indicates the number of hops from the source node.

Num_Members: This field indicates the number of members that belongs to the cluster.

DEST: This field determines the destination node.

Source: This field determines the address of source.

Source node latitude: This field indicates the latitude of the source node.

Source node longitude: This field determines the longitude of the source node.

Cluster-head neighbour address: This field indicates the address of the Cluster-head neighbour.

Cluster-head neighbour latitude: This field indicates the latitude of the Cluster-head neighbour.

Cluster-head neighbour longitude: This field determines the longitude of the Cluster head neighbour.

Location Request Packet (LREQ)

Location request packets allow the source node to request a location of the destination node (Figure A.2). The attributes of the location request packet include:

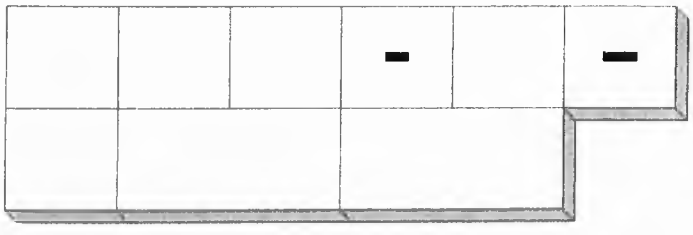


Figure A.2: Location Request Packet (LREQ).

Type: This field determines the type of the packet that has been sent (Hello, Location Request, Location Reply, Data, and Acknowledgment).
Type of node: This field determines the type of the node that has sent the message, Undecided, Member, Gateway or Cluster-head.

Hop Count: This field indicates the number of hops from the source node.

TTL: This field determines the time to live for the packet.

Broadcast ID: This field indicates the ID of the packet being transmitted.

DEST: This field determines the destination node.

Source: This field determines the address of source.

Source node latitude: This field indicates the latitude of the source node.

Source node longitude: This field determines the longitude of the source node.

A.1.5 Data Packet

The transmission of the data packet is then based on the location of the source, the neighbours and the destination (Figure A.4). Data Packets have the following attributes:

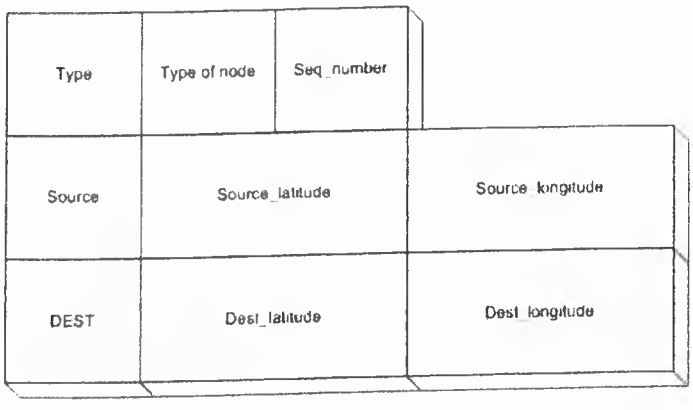


Figure A.4: Data Packet.

Type: This field determines the type of the packet that has been sent (Hello, Location Request, Location Reply, Data, Acknowledgment).

Type of node: This field determines the type of the node that has sent the message, Undecided, Member, Gateway or Cluster-head.

Seq_number: This field determines the sequence of the number of the packet being transmitted.

Source: This field determines the address of source.

Source node latitude: This field indicates the latitude of the source node.

Source node longitude: This field determines the longitude of the source node.

DEST: This field determines the destination node.

Dest_latitude: This field indicates the latitude of the destination node.

Dest_longitude: This field indicates the longitude of the destination node.

A.1.6 Acknowledgment of Data Packet

The maintenance of the location information of the source and destination is based on data and acknowledgment of data packets. Acknowledgment of a Data Packet has the following attributes (Figure A.5):

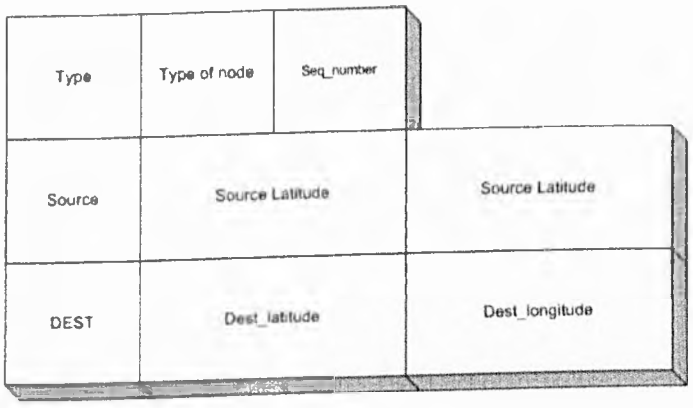


Figure A.5: Acknowledgment of Data Packet (ACK).

Type: This field determines the type of the packet that has been sent (Hello, Location Request, Location Reply, Data, Acknowledgment).

Type of node: This field determines the type of the node that has sent the message, Undecided, Member, Gateway or Cluster-head.

Seq_number: This field determines the sequence of the number of the packet being transmitted.

Source: This field determines the address of source.

Source node latitude: This field indicates the latitude of the source node.

Source node longitude: This field determines the longitude of the source node.

DEST: This field determines the destination node.

Dest_latitude: This field indicates the latitude of the destination node.

Dest_longitude: This field indicates the longitude of the destination node.

A.1.4 Location Reply Packet (LREP)

The destination node has to send a response back to the source giving its location and confirming the receipt of LREQ packet (Figure A.3). This is done by means of sending a LREP packet. The attributes of the LREP are:

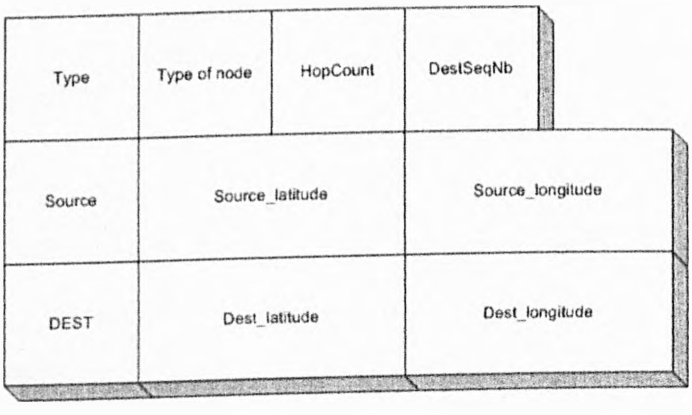


Figure A.3: Location Reply Packet (LREP).

Type: This field determines the type of the packet that has been sent (Hello, Location Request, Location Reply, Data, Acknowledgment).

Type of node: This field determines the type of the node that has sent the message, Undecided, Member, Gateway or Cluster-head.

Hop Count: This field indicates the number of hops from the source node.

DestSeqNb: This field determines the sequence number of the packet being transmitted.

Source: This field determines the address of source.

Source node latitude: This field indicates the latitude of the source node.

Source node longitude: This field determines the longitude of the source node.

DEST: This field determines the destination node.

Dest_latitude: This field indicates the latitude of the destination node.

Dest_longitude: This field indicates the longitude of the destination node.

B. Program Code for LORA-CBF

B.1 /* Process model C form file: LORA-CBF.pr.c */
/* Portions of this file copyright 1992-2002 by OPNET Technologies, Inc. */

```
/* This variable carries the header into the object file */
static const char LORA-CBF_clusters_and_members_only_pr_c [] =
"MIL_3_Tfile_Hdr_ 90A 30A modeler 7 3E75F921 3E75F921 1 elp00ra aquinor 0 0
none none 0 0 none 0 0 0 0 0 0
";
#include <string.h>

/* OPNET system definitions */
#include <opnet.h>

#if defined (__cplusplus)
extern "C" {
#endif
FSM_EXT_DECS
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif

/* Header Block */
#include <math.h>
/* Packet types */

#define REQUEST_PACKET_TYPE          7
#define REPLY_PACKET_TYPE            11
#define HELLO_PACKET_TYPE            19
#define DATA_PACKET_TYPE             21
#define ACK_RREQ                     23
#define ACK_DATA                     27

/* Node Types */

#define Undecided                    1
#define Member                       2
#define Cluster_head                 3
#define Gateway                      4

/* Input and output streams */
#define SRC_STRM                     0 // from upper layer
#define RCV_STRM                     1 // from mac layer
#define SEND_STRM                    0  // toward mac layer
#define DISCARD_STRM                 1  // towards upper layer
#define BROADCAST                    -1
```

```
/* Remote intrpt or self intrpt codes */

#define REP_CODE                30000 //self intrp code in order to serve buffer
#define HELLO_CODE              60000
#define UNDECIDED               200000
#define CLUSTER_HEAD            300000
#define MEMBER                  400000
#define RREQ_REPLY_CODE         500000
#define DATA_REPLY_CODE         600000

/* Offset */
#defineOPTION                   10000

/* Maximum number of nodes in the net */
#define N 64

/* Transition macro */

#define UPPER_LAYER_ARVL      (op_intrpt_type() == OPC_INTRPT_STRM &&
op_intrpt_strm() == SRC_STRM)
#define LOWER_LAYER_ARVL      (op_intrpt_type() == OPC_INTRPT_STRM &&
op_intrpt_strm() == RCV_STRM)
#define Type_Undecided (op_intrpt_type() == OPC_INTRPT_SELF &&
(op_intrpt_code() == UNDECIDED))
#define Type_Cluster_head (op_intrpt_type() == OPC_INTRPT_SELF &&
(op_intrpt_code() == CLUSTER_HEAD))
#define Notice_To_Serve_Buffer (op_intrpt_type() == OPC_INTRPT_SELF &&
op_intrpt_code () == REP_CODE)
#define Retransmit_RREQ (op_intrpt_type () == OPC_INTRPT_REMOTE &&
((int)(op_intrpt_code()/OPTION))*OPTION == RREQ_REPLY_CODE)
#define Retransmit_DATA (op_intrpt_type () == OPC_INTRPT_REMOTE &&
((int)(op_intrpt_code()/OPTION))*OPTION == DATA_REPLY_CODE)

#define IN_SERVICE                  1
#define DOWN                        0
#define ACTIVE                      1
#define INACTIVE                    0

/* infinity */
#define INFINITY -10

int             pk_transmited, pk_received;
double          latency;
double          delay_start = 0.0;
double          delay_stop = 0.0;
```

```
/* Request sent status  */
typedef enum Request_Status_Type
        {
        OFF = 0,                                // Request waiting for reply
        WAITING_FOR_REPLY = 1                   // Request was not sent
        } Request_Status_Type;

/* Hello Module (used when HELLO MODE is activated) */
typedef struct
        {
        Evhandle evt;
        Evhandle evt_cluster;
        Evhandle evt_undecided;
        Evhandle evt_member;
        Packet* hello_msg_template;
        } Hello_Module;

typedef struct
{
int             dest;
int             hopCount;
int             lastHopCount;
int              nextHop;
} RoutingTableEntry; // a routing table entry;

typedef struct
        {
        Evhandle        evt_cluster;
        Evhandle        evt_undecided;
        Objid           id;
        double          previa_position_x;
        double          previa_position_y;
        int             cluster_head;
        int             Seq_number;
        double          timer, live_timer;
        double          active_timer, member_timer;
        double          gateway_timer;
        int             status_of_node;
        int             transmission;
        int             direction;
        int             previa_direction;
        double          registration_time;
        int             status;
        double          latitude;
        double          longitude;
        double          previa_latitude;
        double          previa_longitude;
        } Cluster_Table;
```

```
Cluster_Table nodeid[N];

typedef struct
        {
        int             Member_id[N];
        int             Gw[N];
        int             Ch[N];
        } Member_Table;

        Member_Table Clusterhead[N];

typedef struct
        {
        int             Gw[N];
        } Gateway_Table;

        Gateway_Table Gateway_Gateway[N];

typedef struct
        {
        int             Cluster[N];
        } Cluster_head_table;

        Cluster_head_table Cluster[N];

typedef struct
        {
        int             Member_Neighbor[N];
        } Register_Member;

        Register_Member Member_Neighbor[N];

typedef struct
        {
        int             direction;
        int             status;
        int             num_clusters;
        double          latitude;
        double          longitude;
        int             counter;
        Evhandle        evt;
        double          timer;
        Boolean         flag;
        double          previa_latitude;
        double          previa_longitude;
        double          previo_timer;
        } Cluster_Head_Neighbour;
```

```
typedef struct
        {
        int                 direction;
        int                 status;
        double              timer;
        double              latitude;
        double              longitude;
        double              previa_latitude;
        double              previa_longitude;
        double              previo_timer;
        } Member_Neighbour_Attributes;

typedef struct
        {
        int                 status;
        double              timer;
        Evhandle            evt;
        double              latitude;
        double              longitude;
        int                 direction;
        double              previa_latitude;
        double              previa_longitude;
        double              previo_timer;
        } Cluster_Head_Member;

typedef struct
        {
        int                         sequence_number;
        double                      expirationTime;
        Request_Status_Type status;
        int                         ttl;
        Evhandle                    evt;
        } RequestSentAttributes;

typedef struct
        {
        int                         broadcastID;
        double                      expirationTime;
        Evhandle                    evt;
        } RequestSeenAttributes;

typedef struct
        {
        int                         sequence_number;
        double                      expirationTime;
        Evhandle                    evt;
        } ReplySeenAttributes;
```

```
typedef struct
        {
        int                     sequence_number;
        double                  expirationTime;
        } DataSeenAttributes;

typedef struct
        {
        int                     sequence_number;
        double                  expirationTime;
        } AckSeenAttributes;

typedef struct
        {
        double                  latitude;
        double                  longitude;
        double                  timer;
        } LocationAttributes;

typedef struct
        {
        int                     counter;
        } RetrasmissionAttributes;

typedef struct
        {
        double                  timer;
        } RegisterAttributes;

 RegisterAttributes                     Register_Cluster[N][N][N];

Cluster_Head_Neighbour                  Cluster_Neighbour[N][N];
Member_Neighbour_Attributes             Member_Neighbour[N][N];
Cluster_Head_Member                     Cluster_Member[N][N];

void LORA-CBF_pk_send_to_mac_layer(Packet *pk, int nextHop);
void LORA-CBF_pk_print(Packet* pk_ptr);
Boolean LORA-CBF_pk_is_in_tr(Packet * pk_ptr);
void LORA-CBF_hello_msg_receive(Packet* hello_pk_ptr);
void LORA-CBF_data_pk_queue(Packet* pk);
void LORA-CBF_rrep_pk_generate_from_destination(Packet* rreq_pk_ptr);
void LORA-CBF_rreq_pk_forward(Packet* rreq_pk_ptr);
Boolean LORA-CBF_rreq_pk_is_fresh_enough(Packet* rreq_pk_ptr);
void bubbleSort(double pos[][2], int max);
Boolean LORA-CBF_rrep_pk_is_fresh_enough(Packet* rrep_pk_ptr);
Packet* LORA-CBF_data_pk_dequeue(int destination);
Boolean LORA-CBF_data_pk_is_fresh_enough(Packet* data_pk_ptr);
void bubbleSortfar(double pos[][2], int max);
```

void LORA-CBF_entry_update_or_create_from_rreq(Packet* rreq_pk_ptr);
void LORA-CBF_become_member(Packet* hello_pk_ptr, int source, double latitude_receiver, double longitude_receiver, double latitude, double longitude);
void LORA-CBF_register_cluster_head(Packet* hello_pk_ptr, int source, double latitude_receiver, double longitude_receiver, double latitude, double longitude);
void LORA-CBF_register_member_and_cluster_head(Packet* hello_pk_ptr, int source, double latitude_receiver, double longitude_receiver, double latitude, double longitude, double Latitude_cluster, double Longitude_cluster, int Cluster_Neighbor);
int LORA-CBF_buffer_size_get(int destination);
void LORA-CBF_data_pk(Packet* data_pk_ptr);
Boolean LORA-CBF_buffer_is_empty(int destination);
void LORA-CBF_data_pk_receive(Packet* data_pk_ptr);
void LORA-CBF_ack_data_pk_receive(Packet* ack_data_pk_ptr);
void LORA-CBF_rreq_pk_generate(int destination);
void LORA-CBF_buffer_serve(int destination);
void LORA-CBF_become_member_from_undecided(Packet* hello_pk_ptr, int source, double latitude_receiver, double longitude_receiver, double latitude, double longitude);
void LORA-CBF_member_from_cluster_head(Packet* hello_pk_ptr, int source, double latitude_receiver, double longitude_receiver, double latitude, double longitude);
void LORA-CBF_register_member(Packet* hello_pk_ptr, int source, double Latitude_cluster, double Longitude_cluster, double latitude, double longitude, int Cluster_Neighbor, double latitude_receiver, double longitude_receiver);
void LORA-CBF_data_pk_renewals(int destination);
Boolean LORA-CBF_ack_rreq_pk_is_fresh_enough(Packet* ack_rreq_pk_ptr);
int nextCluster_source_from_rrep(Packet* rrep_pk_ptr, double latitude, double longitude, double source_latitude, double source_longitude, int source);
Boolean predicting_position_next_cluster(int nextCluster, double latitude, double longitude);
int nextCluster_dest_from_data(Packet* data_pk_ptr, double latitude, double longitude, double dest_latitude, double dest_longitude, int destination);
void Next_Member_dest_from_cluster(Packet* data_pk_ptr, double latitude, double longitude, double dest_latitude, double dest_longitude, int nextCluster);
int nextCluster_source_from_ack_data(Packet* ack_data_pk_ptr, double latitude, double longitude, double source_latitude, double source_longitude, int source);
void Next_Member_source_from_cluster_rrep(Packet* rrep_pk_ptr, double latitude, double longitude, double source_latitude, double source_longitude, int nextCluster);
void Next_Member_source_from_cluster_ack_data(Packet* ack_data_pk_ptr, double latitude, double longitude, double source_latitude, double source_longitude, int nextCluster);
Boolean predicting_position_destination(int dest, double latitude, double longitude);
Boolean predicting_position_source(int source, double latitude, double longitude);
Boolean LORA-CBF_ack_data_pk_is_fresh_enough(Packet* ack_data_pk_ptr);
void Next_Member_dest_data(Packet* data_pk_ptr, double latitude, double longitude, double dest_latitude, double dest_longitude, int nextCluster, int dest);
void Next_Member_source_rrep(Packet* rrep_pk_ptr, double latitude, double longitude, double source_latitude, double source_longitude, int nextCluster, int source);
void Next_Member_source_ack_data(Packet* ack_data_pk_ptr, double latitude, double longitude, double source_latitude, double source_longitude, int nextCluster, int source);

```
void LORA-CBF_become_member_gateway(Packet* hello_pk_ptr, int source, double
latitude_receiver, double longitude_receiver, double latitude, double longitude);
void LORA-CBF_register_member_and_cluster_only(Packet* hello_pk_ptr, int source,
double Latitude_cluster, double Longitude_cluster, double latitude, double longitude,
int Cluster_Neighbor, double latitude_receiver, double longitude_receiver);
double factor_Member_Neighbour(int nextHop);
double factor_Cluster_Neighbour(int nextCluster);
double factor_Cluster_Member(int dest);
double factor_source(int source);

/* End of Header Block */

#if !defined (VOSD_NO_FIN)
#undef BIN
#undef BOUT
#defineBIN    FIN_LOCAL_FIELD(last_line_passed) = __LINE__ - _block_origin;
#defineBOUT BIN
#defineBINIT FIN_LOCAL_FIELD(last_line_passed) = 0; _block_origin = __LINE__;
#else
#defineBINIT
#endif /* #if !defined (VOSD_NO_FIN) */

/* State variable definitions */
typedef struct
        {
        /* Internal state tracking for FSM */
        FSM_SYS_STATE
        /* State Variables */
        Objid                           node_id;
        int                             node_addr;
        int                             DEBUG;
        int                             HELLO_MODE;
        int                             TTL_START;
        double                          TR;
        Hello_Module                    hello_module;
        Distribution *                  hello_dist;
        double                          HELLO_INTERVAL;
        double                          CLUSTER_INTERVAL;
        RequestSentAttributes       RequestSent[N];
        RequestSeenAttributes           RequestSeen[N][N];
        int                             myBroadcastID;
        int                             NET_DIAMETER;
        double                          NODE_TRAVERSAL_TIME;
        double                          NET_TRAVERSAL_TIME;
        double                          BROADCAST_RECORD_TIME;
        double                          ACTIVE_ROUTE_TIMEOUT;
        double                          MY_ROUTE_TIMEOUT;
        int                             mySeqNb;
```

```
int                          TIMER_RENEW;
int                          ackNb;
LocationAttributes           Location[N];
RetrasmissionAttributes      Retransmission[N][N];
int                          Data_seq_number;
ReplySeenAttributes          ReplySeen[N][N];
Distribution *               transmit_dist;
double                       Range_predicted;
Stathandle                   retransmissions_hndl;
int                          Packet_Retransmit;
Stathandle                   latency_hndl;
double                       latency_start;
double                       delay;
double                       tot_delay;
double                       average_delay;
Stathandle                   average_delay_hndl;
Stathandle                   buffer_packets_hndl;
double                       buffer_time;
double                       latency_stop;
Stathandle                   pk_received_hndl;
double                       EXPIRATION_TIME;
Stathandle                   eficiency_hndl;
Stathandle                   pk_transmited_hndl;
double                       MINIMUM_TIME;
double                       MINIMUM_DELAY;
DataSeenAttributes           Dataseen[N][N];
AckSeenAttributes            Ackseen[N][N];
double                       CONVERT_CLUSTER;
} LORA-CBF_clusters_and_members_only_state;

#define pr_state_ptr  ((LORA-CBF_clusters_and_members_only_state*)
Siml_Mod_State_Ptr)
#define node_id                  pr_state_ptr->node_id
#define node_addr                pr_state_ptr->node_addr
#define DEBUG                    pr_state_ptr->DEBUG
#define HELLO_MODE               pr_state_ptr->HELLO_MODE
#define TTL_START                pr_state_ptr->TTL_START
#define TR                       pr_state_ptr->TR
#define hello_module             pr_state_ptr->hello_module
#define hello_dist               pr_state_ptr->hello_dist
#define HELLO_INTERVAL           pr_state_ptr->HELLO_INTERVAL
#define CLUSTER_INTERVAL         pr_state_ptr->CLUSTER_INTERVAL
#define RequestSent              pr_state_ptr->RequestSent
#define RequestSeen            pr_state_ptr->RequestSeen
#define myBroadcastID            pr_state_ptr->myBroadcastID
#define NET_DIAMETER             pr_state_ptr->NET_DIAMETER
#define NODE_TRAVERSAL_TIME    pr_state_ptr->NODE_TRAVERSAL_TIME
#define NET_TRAVERSAL_TIME       pr_state_ptr->NET_TRAVERSAL_TIME
```

```
#define BROADCAST_RECORD_TIME   pr_state_ptr-
>BROADCAST_RECORD_TIME
#define ACTIVE_ROUTE_TIMEOUT  pr_state_ptr->ACTIVE_ROUTE_TIMEOUT
#define MY_ROUTE_TIMEOUT          pr_state_ptr->MY_ROUTE_TIMEOUT
#define mySeqNb                   pr_state_ptr->mySeqNb
#define TIMER_RENEW               pr_state_ptr->TIMER_RENEW
#define ackNb                     pr_state_ptr->ackNb
#define Location                  pr_state_ptr->Location
#define Retransmission          pr_state_ptr->Retransmission
#define Data_seq_number           pr_state_ptr->Data_seq_number
#define ReplySeen                 pr_state_ptr->ReplySeen
#define transmit_dist             pr_state_ptr->transmit_dist
#define Range_predicted           pr_state_ptr->Range_predicted
#define retransmissions_hndl      pr_state_ptr->retransmissions_hndl
#define Packet_Retransmit         pr_state_ptr->Packet_Retransmit
#define latency_hndl              pr_state_ptr->latency_hndl
#define latency_start             pr_state_ptr->latency_start
#define delay                     pr_state_ptr->delay
#define tot_delay                 pr_state_ptr->tot_delay
#define average_delay           pr_state_ptr->average_delay
#define average_delay_hndl        pr_state_ptr->average_delay_hndl
#define buffer_packets_hndl       pr_state_ptr->buffer_packets_hndl
#define buffer_time               pr_state_ptr->buffer_time
#define latency_stop              pr_state_ptr->latency_stop
#define pk_received_hndl          pr_state_ptr->pk_received_hndl
#define EXPIRATION_TIME           pr_state_ptr->EXPIRATION_TIME
#define eficiency_hndl            pr_state_ptr->eficiency_hndl
#define pk_transmited_hndl        pr_state_ptr->pk_transmited_hndl
#define MINIMUM_TIME              pr_state_ptr->MINIMUM_TIME
#define MINIMUM_DELAY             pr_state_ptr->MINIMUM_DELAY
#define Dataseen                  pr_state_ptr->Dataseen
#define Ackseen                   pr_state_ptr->Ackseen
#define CONVERT_CLUSTER           pr_state_ptr->CONVERT_CLUSTER
```

```
/* This macro definition will define a local variable called   */
/* "op_sv_ptr" in each function containing a FIN statement.*/
/* This variable points to the state variable data structure,    */
/* and can be used from a C debugger to display their values.          */
#undef FIN_PREAMBLE
#define FIN_PREAMBLE     LORA-CBF_clusters_and_members_only_state
*op_sv_ptr = pr_state_ptr;

/* Function Block */

enum { _block_origin = __LINE__ };

////////////////////////////////////////////////////////////
/***************** LORA-CBF_pk_send_to_mac_layer () *********/
////////////////////////////////////////////////////////////

void LORA-CBF_pk_send_to_mac_layer(Packet *pk, int nextHop)
{

Ici*   ici_ptr;

/*
/* Send packet (pk) to the MAC layer interface. The routine also
/* installs an ICI in order to indicate the next hop address to
/* which the packet should be sent.
*/

// set IP header
// PreviousHop field = source IP address
op_pk_nfd_set(pk,"PreviousHop",node_addr);
// NextHop field = destination IP address
op_pk_nfd_set(pk,"NextHop",nextHop);
// set TR_source field for Transmission Range purpose
op_pk_nfd_set (pk,"TR_source",node_id);
// print packet
LORA-CBF_pk_print(pk);
// create and install an ici to communicate the packet destination
// to the MAC layer interface.
ici_ptr = op_ici_create("LORA-CBF_notice_to_send");
op_ici_attr_set(ici_ptr,"Packet_Destination",nextHop);
op_ici_install(ici_ptr);
// send packet to mac layer
op_pk_send(pk,SEND_STRM);

printf("     > Packet succesfully sent to MAC layer\n");
}
```

```
/*************************************************************/
/********************* PRINT PACKET **********************/
/*************************************************************/
void LORA-CBF_pk_print(Packet* pk_ptr)
        {
int             ttl, pkType,nextHop,direction, previousHop,dest,num_members,
                destSeqNb, broadcastID,hopCount,Type_of_node,source;
int             next_cluster, Transmission, seq_number;
double          timer,latitude,longitude;
double          source_latitude, source_longitude;
double          dest_latitude, dest_longitude;

/*
/* Print the packet given in parameter
*/

// get the type of the packet
op_pk_nfd_get(pk_ptr,"Type",&pkType);
op_pk_nfd_get(pk_ptr,"NextHop",&nextHop);
op_pk_nfd_get(pk_ptr,"PreviousHop",&previousHop);
op_pk_nfd_get(pk_ptr,"SRC",&source);
op_pk_nfd_get(pk_ptr,"DEST",&dest);
op_pk_nfd_get(pk_ptr,"HopCount",&hopCount);
op_pk_nfd_get(pk_ptr,"Next_Cluster",&next_cluster);
op_pk_nfd_get(pk_ptr,"Dest_latitude",&dest_latitude);
op_pk_nfd_get(pk_ptr,"Dest_longitude",&dest_longitude);
op_pk_nfd_get(pk_ptr,"Seq_number",&seq_number);
op_pk_nfd_get(pk_ptr,"SRC_direction",&direction);

timer= op_sim_time();

switch(pkType) {
case DATA_PACKET_TYPE: // DATA
                {

op_pk_nfd_get(pk_ptr,"Source_latitude", &source_latitude);
op_pk_nfd_get(pk_ptr,"Source_longitude",&source_longitude);
printf("    /*****************************************************/\n");
printf("    /****************** DATA Packet ******************/\n");
printf("    /*****************************************************/\n");
printf("    /* From  = %d\n",previousHop);
printf("    /* To    = %d\n",nextHop);
printf("    /* Timer = %f\n",timer);
printf("    /-----------------------------------------------/\n");
printf("    /* Source                = %d\n",source);
printf("    /* Destination           = %d\n",dest);
printf("    /* Source_latitude       = %f\n",source_latitude);
```

```
printf("    /*  Source_longitude       = %f\n",source_longitude);
printf("    /*  Sequence_number        = %d\n",seq_number);
printf("    /*                                 /\n");
switch(nodeid[node_addr].status_of_node) {
case Undecided: { // Undecided
printf("    /*  Type of node           = Undecided          /\n");
break;
                }
case Member: { // Member
printf("    /*  Type of node           = Member             /\n");
break;
                                       }
case Cluster_head: { // Cluster_head
printf("    /*  Type of node           = Cluster_head       /\n");
switch(Transmission) {
case ACTIVE: {
printf("    /*  Status          = Active             /\n");
printf("    /*                                  /\n");
break;
                                       }
case INACTIVE: {
printf("    /*  Status          = Inactive           /\n");
break;
                                       }
                }
break;
                                       }
case Gateway: { // Bridge
printf("    /*  Type of node           = Bridge             /\n");
break;
                }
                }
printf("    /*****************************************/\n");
break;
                }

case REQUEST_PACKET_TYPE: // RREQ received
                {
op_pk_nfd_get(pk_ptr,"TTL",&ttl);
op_pk_nfd_get(pk_ptr,"BroadcastID",&broadcastID);
printf("    /*****************************************/\n");
printf("    /*************** REQUEST Packet **********/\n");
printf("    /*****************************************/\n");
printf("    /*  TTL   = %d\n",ttl);
printf("    /-----------------------------------------/\n");
printf("    /*  From  = %d\n",previousHop);
printf("    /*  Timer = %f\n",timer);
printf("    /-----------------------------------------/\n");
```

```
printf("    /*  source          = %d\n",source);
printf("    /*  Destination     = %d\n",dest);
printf("    /*  HopCount        = %d\n",hopCount);
printf("    /*  BroadcastID     = %d\n",broadcastID);
printf("    /*                                  /\n");
switch(nodeid[node_addr].status_of_node) {
case Undecided: { // Undecided
printf("    /*  Type of node            = Undecided          /\n");
break;
                }
case Member: { // Member
printf("    /*  Type of node            = Member             /\n");
break;
                        }
case Cluster_head: { // Cluster_head
printf("    /*  Type of node            = Cluster_head       /\n");
switch(Transmission) {
case ACTIVE: {
printf("    /*  Status          = Active            /\n");
printf("    /*                                  /\n");
break;
                                        }
case INACTIVE: {
printf("    /*  Status          = Inactive          /\n");
break;
                                        }
                }
 break;
                                        }
case Gateway: { // Bridge
printf("    /*  Type of node            = Bridge             /\n");
break;
                                }
                        }
printf("    /*************************************/\n");
break;
                        }

case ACK_RREQ: // Acknowledge from the rreq
                        {
printf("    /*************************************/\n");
printf("    /********** ACK_RREQ Packet **********/\n");
printf("    /*************************************/\n");
printf("    /*  From  = %d\n",previousHop);
printf("    /*  To    = %d\n",nextHop);
printf("    /*  Timer = %f\n",timer);
printf("    /-----------------------------------------/\n");
printf("    /*  Next_Cluster     = %d\n",next_cluster);
```

```
printf("    /* Destination      = %d\n",dest);
printf("    /*************************************/\n");
break;
                        }

case REPLY_PACKET_TYPE: // RREP received
                        {
op_pk_nfd_get(pk_ptr,"DestSeqNb",&destSeqNb);

printf("    /*************************************/\n");
if(nextHop > -1)
printf("    /****************** REPLY Packet ******/\n");
else
printf("    /********** BROADCAST HELLO Packet **/\n");
printf("    /*************************************/\n");
printf("    /* From  = %d\n",previousHop);
printf("    /* To    = %d\n",nextHop);
printf("    /* Timer = %f\n",timer);
printf("    /* ----------------------------------------------/\n");
printf("    /* source           = %d\n",source);
printf("    /* destination      = %d\n",dest);
printf("    /* Dest_Latitude    = %f\n",dest_latitude);
printf("    /* Dest_longitude   = %f\n",dest_longitude);
printf("    /* DestSeqNb        = %d\n",destSeqNb);
printf("    /*                                  /\n");
switch(nodeid[node_addr].status_of_node) {
case Undecided: { // Undecided
printf("    /* Type of node          = Undecided          /\n");
break;
                                }
case Member: { // Member
printf("    /* Type of node          = Member             /\n");
break;
                                }
case Cluster_head: { // Cluster_head
printf("    /* Type of node          = Cluster_head       /\n");
switch(Transmission) {
case ACTIVE: {
printf("    /* Status           = Active             /\n");
printf("    /*                                  /\n");
break;
                                }
case INACTIVE: {
printf("    /* Status           = Inactive           /\n");
break;
                                }
                        }
break;
```

```
                                        }
case Gateway: { // Bridge
printf("    /*  Type of node           = Bridge              /\n");
break;
                                }
                        }
printf("    /*************************************/\n");
break;
                        }

case HELLO_PACKET_TYPE:  // HELLO PACKET Received
                        {
op_pk_nfd_get(pk_ptr,"Type_of_node", &Type_of_node);
op_pk_nfd_get(pk_ptr,"Longitude", &longitude);
op_pk_nfd_get(pk_ptr,"Latitude", &latitude);
op_pk_nfd_get(pk_ptr,"Transmission", &Transmission);
op_pk_nfd_get(pk_ptr,"Num_Members", &num_members);
printf("\n    /*************************************/\n");
if(nextHop > -1)
printf("    /***************** REPLY HELLO Packet */\n");
else
printf("    /********* BROADCAST HELLO Packet ***/\n");
printf("    /*************************************/\n");
printf("    /*  From  = %d Source = %d\n", previousHop, source);
printf("    /*  To    = %d  Destination    = %d\n", nextHop, dest);
printf("    /*                                          /\n");
printf("    /*  Timer = %f\n",timer);
printf("    /* ---------------------------------------------/\n");
printf("    /*                                      /\n");
switch(nodeid[node_addr].status_of_node) {
case Undecided: { // Undecided
printf("    /*  Type of node           = Undecided           /\n");
break;
                        }
case Member: { // Member
printf("    /*  Type of node           = Member              /\n");
break;
                        }
case Cluster_head: { // Cluster_head
printf("    /*  Type of node           = Cluster_head        /\n");
printf("    /*  Number of Members      = %d\n",num_members);
switch(Transmission) {
case ACTIVE: {
printf("    /*  Status          = Active              /\n");
printf("    /*                                  /\n");
brreak;
                        }
case INACTIVE: {
```

```
printf("    /*  Status            = Inactive            /\n");
break;
                        }
                        }
break;
                }
case Gateway: { // Bridge
printf("    /*  Type of node          = Bridge            /\n");
break;
                        }
                        }
printf("    /*                                          /\n");
printf("    /*  Latitude = %f Longitude = %f\n",latitude, longitude);
printf("    /*****************************************/\n");
break;
                        }

case ACK_DATA:  // ACK_DATA packet received
                        {
printf("    /*****************************************/\n");
printf("    /********* ACK_DATA Packet ***********/\n");
printf("    /*****************************************/\n");
printf("    /*  From  = %d\n",previousHop);
printf("    /*  To    = %d\n",nextHop);
printf("    /*  Timer = %f\n",timer);
printf("    /*-----------------------------------------------/\n");
printf("    /*  Destination      = %d\n",dest);
printf("    /*  source           = %d\n",source);
printf("    /*  Dest_Latitude    = %f\n",dest_latitude);
printf("    /*  Dest_longitude   = %f\n",dest_longitude);
printf("    /*                                          /\n");
switch(nodeid[node_addr].status_of_node) {
case Undecided: { // Undecided
printf("    /*  Type of node          = Undecided         /\n");
break;
                        }
case Member: { // Member
printf("    /*  Type of node          = Member            /\n");
break;
                        }
case Cluster_head: { // Cluster_head
printf("    /*  Type of node          = Cluster_head      /\n");
switch(Transmission) {
case ACTIVE: {
printf("    /*  Status           = Active            /\n");
printf("    /*                                          /\n");
break;
                        }
```

```
case INACTIVE: {
printf("     /*  Status            = Inactive            /\n");
break;
                                    }
                            }
break;
                                    }
case Gateway: { // Bridge
printf("     /*  Type of node            = Bridge            /\n");
break;
                                }
                            }
printf("     /************************************/\n");
break;
                            }
                    }
            }
```

```
/*****************************************************************/
/***************** Check if pk sent by node within TR ************/
/*****************************************************************/

Boolean LORA-CBF_pk_is_in_tr(Packet * pk_ptr)
{
Objid     source_id;
double    x,y,z;
double    latitude,longitude,altitude;
double    x_source,y_source,z_source;
double    dist;
Boolean    result= OPC_FALSE;

/*
/* Check whether the source node of the packet is within
/* Range_predicted from the current node or not.
*/

// Read the Objid of the source node
op_pk_nfd_get(pk_ptr,"TR_source",&source_id);
//Get the position of the current node
op_ima_node_pos_get (node_id, &latitude,&longitude, &altitude, &x, &y, &z);
//Get the position of the source node
op_ima_node_pos_get (source_id, &latitude,&longitude, &altitude, &x_source,
&y_source, &z_source);
// Compute distance between the two nodes
dist = sqrt((x-x_source)*(x-x_source) + (y-y_source)*(y-y_source) + (z-z_source)*(z-
z_source));

if ((dist > 0.0) && (dist <= TR))
```

```
        {
result= OPC_TRUE;
        }

return result;
}

////////////////////////////////////////////////////////////
/********************** HANDLE HELLO MESSAGE ************/
////////////////////////////////////////////////////////////
void LORA-CBF_hello_msg_receive(Packet* hello_pk_ptr)

{

// var
int             i, nextHop, num_members;
int             source, dest, hopCount,Cluster_Neighbor;
int             previousHop,Type_of_node, transmission, Members;
double          latitude,longitude,altitude,x_pos,y_pos,z_pos;
double          Latitude_cluster,Longitude_cluster;
double          latitude_receiver, longitude_receiver;
Compcode        comp_code;

/*
/* This routine processes the upcoming hello message.
*/
// begin

// packet read
op_pk_nfd_get (hello_pk_ptr,"PreviousHop",&previousHop);
op_pk_nfd_get (hello_pk_ptr,"NextHop",&nextHop);
op_pk_nfd_get (hello_pk_ptr,"SRC",&source);
op_pk_nfd_get (hello_pk_ptr,"DEST",&dest);
// N.B. hop count field is incremented by 1 at reception
op_pk_nfd_get (hello_pk_ptr,"HopCount",&hopCount);
op_pk_nfd_set (hello_pk_ptr,"HopCount",hopCount+1);
op_pk_nfd_get (hello_pk_ptr,"HopCount",&hopCount);
op_pk_nfd_get (hello_pk_ptr,"Type_of_node",&Type_of_node);
op_pk_nfd_get (hello_pk_ptr,"Latitude",&latitude_receiver);
op_pk_nfd_get (hello_pk_ptr,"Longitude",&longitude_receiver);
op_pk_nfd_get (hello_pk_ptr,"Latitude_cluster",&Latitude_cluster);
op_pk_nfd_get (hello_pk_ptr,"Longitude_cluster",&Longitude_cluster);
op_pk_nfd_get (hello_pk_ptr,"Cluster_Neighbor",&Cluster_Neighbor);
op_pk_nfd_get (hello_pk_ptr,"Transmission", &transmission);
op_pk_nfd_get (hello_pk_ptr,"Num_Members", &num_members);

if (nextHop == BROADCAST)
```

```
        {
        printf("     > A BROADCAST Hello Packet Received (from node
%d)\n",source);
        }
else
        {
        printf("     > A Reply Hello Packet Received (from node %d)\n",source);
        }
node_id = op_topo_parent (op_id_self());
comp_code = op_ima_obj_pos_get (node_id, &latitude, &longitude, &altitude,
&x_pos, &y_pos, &z_pos);

/* This line tests the completion code, and upon detecting failure, calls op_sim_end () to
*/
/* immediately end the simulation and print and error message to the standard output
device,*/
/* and the opnet message area.                                                    */

if (comp_code == OPC_COMPCODE_FAILURE)
op_sim_end ("get attributes failed", "", "","");

switch(nodeid[node_addr].status_of_node) {

case Undecided:

{
if ((Type_of_node == Cluster_head) && (hopCount == 1))
        {
LORA-CBF_become_member_from_undecided(hello_pk_ptr, source,
latitude_receiver, longitude_receiver, latitude, longitude);
        }
else
        {
        op_pk_destroy (hello_pk_ptr);
        }
break;
} // End of the case Undecided

case Member:
{
if ((Type_of_node == Cluster_head) && (hopCount == 1))
        {
if (nodeid[node_addr].cluster_head == source)
                {
printf("     > The {{Member %d}} has received a Hello message from its {{Cluster
%d}}\n", node_addr, source);
LORA-CBF_become_member(hello_pk_ptr, source, latitude_receiver,
longitude_receiver, latitude, longitude);
```

```
                }
else
                {
if ((op_sim_time () - nodeid[node_addr].member_timer) < EXPIRATION_TIME)
                        {
printf("     > The {{Member %d}} has received a Hello message from the {{Cluster
%d}}\n", node_addr, source);
LORA-CBF_become_member_gateway(hello_pk_ptr, source, latitude_receiver,
longitude_receiver, latitude, longitude);
                        }
 else
                        {
 printf("     > The {{Member %d}} hasn't received any hello packet from its {{Cluster
 head %d}}\n",node_addr, nodeid[node_addr].cluster_head);
 printf("     > Changing its state to Member of the new {{Cluster head %d}}\n", source);

 LORA-CBF_become_member(hello_pk_ptr, source, latitude_receiver,
 longitude_receiver, latitude, longitude);
                        }
                }
        }
 else if ((Type_of_node == Member) && (hopCount == 1))
        {
 if (nodeid[node_addr].cluster_head != nodeid[source].cluster_head)
                {
 printf("     > The {{Member %d}} has received a Hello message from one {{Member
 %d}} of different cluster\n",node_addr, source);
 LORA-CBF_register_member(hello_pk_ptr, source, Latitude_cluster,
 Longitude_cluster, latitude, longitude, Cluster_Neighbor, latitude_receiver,
 longitude_receiver);
                }
 else
                {
 printf("     > The {{Member %d}} has received a Hello message from one {{Member
 %d}} of the same cluster\n",node_addr, source);
 LORA-CBF_register_member_and_cluster_only(hello_pk_ptr, source,
 Latitude_cluster, Longitude_cluster, latitude, longitude, Cluster_Neighbor,
 latitude_receiver, longitude_receiver);
                }
        }
 else
        {
        op_pk_destroy (hello_pk_ptr);
        }
 break;

 } // End of case Member
```

```
case Cluster_head:

{
 if ((Type_of_node == Cluster_head) && (hopCount == 1))
                {
printf("     > Receiving a Hello msg from {{Cluster_head %d }}\n",source);

                Members = 0;
                for (i=0; i<N; i++)

                        {
if ((Cluster_Member[node_addr][i].status == IN_SERVICE) && ((op_sim_time () -
Cluster_Member[node_addr][i].timer) < EXPIRATION_TIME))

                                {
Members++;
                                }

 else

                                {

                                }

                        }

if (Members <= num_members)

                        {
printf("     > There are two Cluster_head in the same Cluster\n");
printf("     > The {{Cluster_head %d}} has {{%d members}} and the {{Cluster_head
%d}} has {{%d members}}\n", node_addr, Members, source, num_members);
printf("     > Converting the Cluster_head %d into a member\n", node_addr);

LORA-CBF_member_from_cluster_head(hello_pk_ptr, source, latitude_receiver,
longitude_receiver, latitude, longitude);

                        }
                }

else if ((Type_of_node == Member) && (hopCount == 1))

        {

LORA-CBF_register_member_and_cluster_head(hello_pk_ptr, source,
latitude_receiver, longitude_receiver, latitude, longitude, Latitude_cluster,
Longitude_cluster, Cluster_Neighbor);
```

```
        }
else if ((Type_of_node == Undecided) && (hopCount == 1))
        {
printf("     > Receiving a Hello packet from {{undecided node %d}} \n", source);
op_pk_destroy (hello_pk_ptr);
        }
break;
}
}
}
```

```
////////////////////////////////////////////////////////////////////
/********* Become Member from Undecided ********************/
////////////////////////////////////////////////////////////////////
void LORA-CBF_become_member_from_undecided(Packet* hello_pk_ptr, int source,
double latitude_receiver, double longitude_receiver, double latitude, double longitude)
        {
int Type_of_node;
op_ev_cancel(hello_module.evt);
Cluster[node_addr].Cluster[source] = source;
nodeid[node_addr].cluster_head = source;
nodeid[node_addr].member_timer = op_sim_time ();
Cluster_Neighbour[node_addr][source].previa_latitude =
        Cluster_Neighbour[node_addr][source].latitude;
Cluster_Neighbour[node_addr][source].previa_longitude =
        Cluster_Neighbour[node_addr][source].longitude;
Cluster_Neighbour[node_addr][source].previo_timer =
        Cluster_Neighbour[node_addr][source].timer;
Cluster_Neighbour[node_addr][source].status = IN_SERVICE;
Cluster_Neighbour[node_addr][source].timer = op_sim_time();
Cluster_Neighbour[node_addr][source].latitude = latitude_receiver;
Cluster_Neighbour[node_addr][source].longitude = longitude_receiver;
```

```
printf("     > Registrating the attributes for the {{Cluster Head %d}} at the {{node
%d}} \n",Cluster[node_addr].Cluster[source], node_addr);

nodeid[node_addr].status_of_node = Member;
Type_of_node = Member;

op_pk_nfd_set (hello_pk_ptr, "HopCount",0);
op_pk_nfd_set (hello_pk_ptr, "SRC",node_addr);
op_pk_nfd_set (hello_pk_ptr, "DEST",node_addr);
op_pk_nfd_set (hello_pk_ptr, "Type_of_node", Type_of_node);
op_pk_nfd_set (hello_pk_ptr, "Latitude", latitude);
op_pk_nfd_set (hello_pk_ptr, "Longitude", longitude);
op_pk_nfd_set (hello_pk_ptr, "Cluster_Neighbor", source);
op_pk_nfd_set (hello_pk_ptr, "Latitude_cluster", latitude_receiver);
op_pk_nfd_set (hello_pk_ptr, "Longitude_cluster", longitude_receiver);

LORA-CBF_pk_send_to_mac_layer(hello_pk_ptr, BROADCAST);
nodeid[node_addr].evt_cluster = op_intrpt_schedule_self(op_sim_time () +
(CONVERT_CLUSTER + op_dist_outcome(transmit_dist)), CLUSTER_HEAD);

printf ("     - Packet sent back to Cluster Head (from node %d)\n",node_addr);
        }

////////////////////////////////////////////////////////////////////////
/********* LORA-CBF_become_Member **********************/
////////////////////////////////////////////////////////////////////////

void LORA-CBF_become_member(Packet* hello_pk_ptr, int source, double
latitude_receiver, double longitude_receiver, double latitude, double longitude)

        {

int Type_of_node;

op_ev_cancel(nodeid[node_addr].evt_cluster);

Cluster[node_addr].Cluster[source] = source;
nodeid[node_addr].cluster_head = source;
nodeid[node_addr].member_timer = op_sim_time ();

Cluster_Neighbour[node_addr][source].previa_latitude =
        Cluster_Neighbour[node_addr][source].latitude;
Cluster_Neighbour[node_addr][source].previa_longitude =
        Cluster_Neighbour[node_addr][source].longitude;
Cluster_Neighbour[node_addr][source].previo_timer =
        Cluster_Neighbour[node_addr][source].timer;
```

```
Cluster_Neighbour[node_addr][source].status = IN_SERVICE;
Cluster_Neighbour[node_addr][source].timer = op_sim_time ();

Cluster_Neighbour[node_addr][source].latitude = latitude_receiver;
Cluster_Neighbour[node_addr][source].longitude = longitude_receiver;

printf("     > Registrating the attributes for the {{Cluster Head %d}} at the {{node
%d}} \n",source, node_addr);

nodeid[node_addr].status_of_node = Member;
Type_of_node = Member;

op_pk_nfd_set (hello_pk_ptr, "HopCount",0);
op_pk_nfd_set (hello_pk_ptr, "Type_of_node", Type_of_node);
op_pk_nfd_set (hello_pk_ptr, "SRC", node_addr);
op_pk_nfd_set (hello_pk_ptr, "DEST", source);
op_pk_nfd_set (hello_pk_ptr, "Latitude", latitude);
op_pk_nfd_set (hello_pk_ptr, "Longitude", longitude);
op_pk_nfd_set (hello_pk_ptr, "Cluster_Neighbor", source);
op_pk_nfd_set (hello_pk_ptr, "Latitude_cluster", latitude_receiver);
op_pk_nfd_set (hello_pk_ptr, "Longitude_cluster", longitude_receiver);

LORA-CBF_pk_send_to_mac_layer(hello_pk_ptr, BROADCAST);

nodeid[node_addr].evt_cluster = op_intrpt_schedule_self(op_sim_time () +
(CONVERT_CLUSTER + op_dist_outcome(transmit_dist)), CLUSTER_HEAD);

printf ("     - Packet sent back to Cluster Head (from node %d)\n",node_addr);
        }

////////////////////////////////////////////////////////////////////////////////
/********** LORA-CBF_become_Member_from_gateway *********/
////////////////////////////////////////////////////////////////////////////////

void LORA-CBF_become_member_gateway(Packet* hello_pk_ptr, int source, double
latitude_receiver, double longitude_receiver, double latitude, double longitude)

        {
int Type_of_node;

op_ev_cancel(nodeid[node_addr].evt_cluster);

Cluster[node_addr].Cluster[source] = source;

Cluster_Neighbour[node_addr][source].previa_latitude =
        Cluster_Neighbour[node_addr][source].latitude;
Cluster_Neighbour[node_addr][source].previa_longitude =
        Cluster_Neighbour[node_addr][source].longitude;
```

```
Cluster_Neighbour[node_addr][source].previo_timer =
        Cluster_Neighbour[node_addr][source].timer;

Cluster_Neighbour[node_addr][source].status = IN_SERVICE;
Cluster_Neighbour[node_addr][source].timer = op_sim_time ();

Cluster_Neighbour[node_addr][source].latitude = latitude_receiver;
Cluster_Neighbour[node_addr][source].longitude = longitude_receiver;

printf("     > Registrating the attributes for the {{Cluster Head %d}} at the {{node
%d}} \n",source, node_addr);

nodeid[node_addr].status_of_node = Member;
Type_of_node = Member;

op_pk_nfd_set (hello_pk_ptr, "HopCount",0);
op_pk_nfd_set (hello_pk_ptr, "Type_of_node", Type_of_node);
op_pk_nfd_set (hello_pk_ptr, "SRC", node_addr);
op_pk_nfd_set (hello_pk_ptr, "DEST", source);
op_pk_nfd_set (hello_pk_ptr, "Latitude", latitude);
op_pk_nfd_set (hello_pk_ptr, "Longitude", longitude);
op_pk_nfd_set (hello_pk_ptr, "Cluster_Neighbor", source);
op_pk_nfd_set (hello_pk_ptr, "Latitude_cluster", latitude_receiver);
op_pk_nfd_set (hello_pk_ptr, "Longitude_cluster", longitude_receiver);

LORA-CBF_pk_send_to_mac_layer(hello_pk_ptr, BROADCAST);

nodeid[node_addr].evt_cluster = op_intrpt_schedule_self(op_sim_time () +
(CONVERT_CLUSTER + op_dist_outcome(transmit_dist)), CLUSTER_HEAD);

printf ("     - Packet sent back to Cluster Head (from node %d)\n",node_addr);

        }

////////////////////////////////////////////////////////////////////
/********** LORA-CBF_Member_from_Cluster_head **********/
////////////////////////////////////////////////////////////////////

void LORA-CBF_member_from_cluster_head (Packet* hello_pk_ptr, int source,
double latitude_receiver, double longitude_receiver, double latitude, double longitude)
        {
int Type_of_node;

op_ev_cancel(nodeid[node_addr].evt_cluster);

Cluster[node_addr].Cluster[source] = source;
nodeid[node_addr].cluster_head = source;
nodeid[node_addr].member_timer = op_sim_time ();
```

```
Cluster_Neighbour[node_addr][source].previa_latitude =
        Cluster_Neighbour[node_addr][source].latitude;
Cluster_Neighbour[node_addr][source].previa_longitude =
        Cluster_Neighbour[node_addr][source].longitude;
Cluster_Neighbour[node_addr][source].previo_timer =
        Cluster_Neighbour[node_addr][source].timer;

Cluster_Neighbour[node_addr][source].status = IN_SERVICE;
Cluster_Neighbour[node_addr][source].timer = op_sim_time ();

Cluster_Neighbour[node_addr][source].latitude = latitude_receiver;
Cluster_Neighbour[node_addr][source].longitude = longitude_receiver;

printf("     > Registrating the attributes for the {{Cluster Head %d}} at the {{node
%d}} \n",source, node_addr);

nodeid[node_addr].status_of_node = Member;
Type_of_node = Member;

op_pk_nfd_set (hello_pk_ptr, "HopCount",0);
op_pk_nfd_set (hello_pk_ptr, "Type_of_node", Type_of_node);
op_pk_nfd_set (hello_pk_ptr, "SRC", node_addr);
op_pk_nfd_set (hello_pk_ptr, "DEST", source);
op_pk_nfd_set (hello_pk_ptr, "Latitude", latitude);
op_pk_nfd_set (hello_pk_ptr, "Longitude", longitude);
op_pk_nfd_set (hello_pk_ptr, "Cluster_Neighbor", source);
op_pk_nfd_set (hello_pk_ptr, "Latitude_cluster", latitude_receiver);
op_pk_nfd_set (hello_pk_ptr, "Longitude_cluster", longitude_receiver);

LORA-CBF_pk_send_to_mac_layer(hello_pk_ptr, BROADCAST);

nodeid[node_addr].evt_cluster = op_intrpt_schedule_self(op_sim_time () +
(CONVERT_CLUSTER + op_dist_outcome(transmit_dist)), CLUSTER_HEAD);

printf ("     - Packet sent back to Cluster Head (from node %d)\n",node_addr);
        }
```

```
////////////////////////////////////////////////////////////////////////
/****** LORA-CBF_Register_Member ******************/
////////////////////////////////////////////////////////////////////////

void LORA-CBF_register_member(Packet* hello_pk_ptr, int source, double
Latitude_cluster, double Longitude_cluster, double latitude, double longitude, int
Cluster_Neighbor, double latitude_receiver, double longitude_receiver)

        {
int                     Type_of_node;

if ((op_sim_time () - Member_Neighbour[node_addr][source].timer) >
MINIMUM_TIME)
                {
Member_Neighbor[node_addr].Member_Neighbor[source] = source;

Member_Neighbour[node_addr][source].previa_latitude =
        Member_Neighbour[node_addr][source].latitude;
Member_Neighbour[node_addr][source].previa_longitude =
        Member_Neighbour[node_addr][source].longitude;
Member_Neighbour[node_addr][source].previo_timer =
        Member_Neighbour[node_addr][source].timer;

Member_Neighbour[node_addr][source].status = IN_SERVICE;
Member_Neighbour[node_addr][source].timer = op_sim_time ();

Member_Neighbour[node_addr][source].latitude = latitude_receiver;
Member_Neighbour[node_addr][source].longitude = longitude_receiver;

printf("     > Registrating the attributes for the {{Member %d}} at the {{node %d}}
\n",source, node_addr);

if ((op_sim_time () - Cluster_Neighbour[node_addr][Cluster_Neighbor].timer) >
MINIMUM_TIME)
                        {

Cluster[node_addr].Cluster[Cluster_Neighbor] = Cluster_Neighbor;

Cluster_Neighbour[node_addr][Cluster_Neighbor].previa_latitude =
        Cluster_Neighbour[node_addr][Cluster_Neighbor].latitude;

Cluster_Neighbour[node_addr][Cluster_Neighbor].previa_longitude =
        Cluster_Neighbour[node_addr][Cluster_Neighbor].longitude;

Cluster_Neighbour[node_addr][Cluster_Neighbor].previo_timer =
        Cluster_Neighbour[node_addr][Cluster_Neighbor].timer;

Cluster_Neighbour[node_addr][Cluster_Neighbor].status = IN_SERVICE;
```

```
Cluster_Neighbour[node_addr][Cluster_Neighbor].timer = op_sim_time();
Cluster_Neighbour[node_addr][Cluster_Neighbor].latitude = Latitude_cluster;
Cluster_Neighbour[node_addr][Cluster_Neighbor].longitude = Longitude_cluster;

printf("     > Registrating the attributes for the {{Cluster Head %d}} at the {{node
%d}} \n",Cluster_Neighbor, node_addr);

nodeid[node_addr].status_of_node = Member;
Type_of_node = Member;

op_pk_nfd_set (hello_pk_ptr, "HopCount",0);
op_pk_nfd_set (hello_pk_ptr, "Type_of_node", Type_of_node);
op_pk_nfd_set (hello_pk_ptr, "SRC", node_addr);
op_pk_nfd_set (hello_pk_ptr, "DEST", source);
op_pk_nfd_set (hello_pk_ptr, "Latitude", latitude);
op_pk_nfd_set (hello_pk_ptr, "Longitude", longitude);
op_pk_nfd_set (hello_pk_ptr, "Cluster_Neighbor", Cluster_Neighbor);
op_pk_nfd_set (hello_pk_ptr, "Latitude_cluster", Latitude_cluster);
op_pk_nfd_set (hello_pk_ptr, "Longitude_cluster", Longitude_cluster);

LORA-CBF_pk_send_to_mac_layer(hello_pk_ptr, BROADCAST);

printf ("     - Packet sent back to node Cluster Head (from node %d)\n",node_addr);
                        }

                else
                        {
                        op_pk_destroy (hello_pk_ptr);
                        }
                }
        }

////////////////////////////////////////////////////////////////////////////
/****************** LORA-CBF_Register_Member_only *********/
////////////////////////////////////////////////////////////////////////////

void LORA-CBF_register_member_and_cluster_only(Packet* hello_pk_ptr, int source,
double Latitude_cluster, double Longitude_cluster, double latitude, double longitude,
int Cluster_Neighbor, double latitude_receiver, double longitude_receiver)
        {
if ((op_sim_time () - Member_Neighbour[node_addr][source].timer) >
MINIMUM_TIME)
                        {
Member_Neighbor[node_addr].Member_Neighbor[source] = source;
Member_Neighbour[node_addr][source].previa_latitude =
        Member_Neighbour[node_addr][source].latitude;
Member_Neighbour[node_addr][source].previa_longitude =
        Member_Neighbour[node_addr][source].longitude;
```

```
Member_Neighbour[node_addr][source].previo_timer =
        Member_Neighbour[node_addr][source].timer;

Member_Neighbour[node_addr][source].status = IN_SERVICE;
Member_Neighbour[node_addr][source].timer = op_sim_time ();

Member_Neighbour[node_addr][source].latitude = latitude_receiver;
Member_Neighbour[node_addr][source].longitude = longitude_receiver;

printf("     > Registrating the attributes for the {{Member %d}} at the {{node %d}}
\n",source, node_addr);
                        }
if ((op_sim_time () - Cluster_Neighbour[node_addr][Cluster_Neighbor].timer) >
MINIMUM_TIME)
                        {
Cluster[node_addr].Cluster[Cluster_Neighbor] = Cluster_Neighbor;
Cluster_Neighbour[node_addr][Cluster_Neighbor].previa_latitude =
        Cluster_Neighbour[node_addr][Cluster_Neighbor].latitude;
Cluster_Neighbour[node_addr][Cluster_Neighbor].previa_longitude =
        Cluster_Neighbour[node_addr][Cluster_Neighbor].longitude;
Cluster_Neighbour[node_addr][Cluster_Neighbor].previo_timer =
        Cluster_Neighbour[node_addr][Cluster_Neighbor].timer;

Cluster_Neighbour[node_addr][Cluster_Neighbor].status = IN_SERVICE;
Cluster_Neighbour[node_addr][Cluster_Neighbor].timer = op_sim_time();

Cluster_Neighbour[node_addr][Cluster_Neighbor].latitude = Latitude_cluster;
Cluster_Neighbour[node_addr][Cluster_Neighbor].longitude = Longitude_cluster;

printf("     > Registrating the attributes for the {{Cluster Head %d}} at the {{node
%d}} \n",Cluster_Neighbor, node_addr);

                        }
op_pk_destroy (hello_pk_ptr);

        }
```

```
////////////////////////////////////////////////////////////////////
/********* LORA-CBF_Register_Member and Cluster-head *******/
////////////////////////////////////////////////////////////////////

void LORA-CBF_register_member_and_cluster_head(Packet* hello_pk_ptr, int source,
double latitude_receiver, double longitude_receiver, double latitude, double longitude,
double Latitude_cluster, double Longitude_cluster, int Cluster_Neighbor)

        {
if ((Cluster_Neighbor != node_addr) && (Cluster_Neighbor != -1) &&
((op_sim_time () - Cluster_Neighbour[node_addr][Cluster_Neighbor].timer) >
MINIMUM_TIME))

                {
Cluster_Neighbour[node_addr][Cluster_Neighbor].previa_latitude =
        Cluster_Neighbour[node_addr][Cluster_Neighbor].latitude;
Cluster_Neighbour[node_addr][Cluster_Neighbor].previa_longitude =
        Cluster_Neighbour[node_addr][Cluster_Neighbor].longitude;
Cluster_Neighbour[node_addr][Cluster_Neighbor].previo_timer =
        Cluster_Neighbour[node_addr][Cluster_Neighbor].timer;

Cluster_Neighbour[node_addr][Cluster_Neighbor].status = IN_SERVICE;
        Cluster_Neighbour[node_addr][Cluster_Neighbor].timer = op_sim_time();

Cluster_Neighbour[node_addr][Cluster_Neighbor].latitude = Latitude_cluster;
Cluster_Neighbour[node_addr][Cluster_Neighbor].longitude = Longitude_cluster;

Clusterhead[node_addr].Ch[Cluster_Neighbor]= Cluster_Neighbor;

printf ("     > Registrating the {{ Cluster_head Neighbor %d}}  ***\n",
Cluster_Neighbor);
                }
if ((op_sim_time () - Cluster_Member[node_addr][source].timer) > MINIMUM_TIME)
                {
Cluster_Member[node_addr][source].previa_latitude =
        Cluster_Member[node_addr][source].latitude;
Cluster_Member[node_addr][source].previa_longitude =
        Cluster_Member[node_addr][source].longitude;
Cluster_Member[node_addr][source].previo_timer =
        Cluster_Member[node_addr][source].timer;

Cluster_Member[node_addr][source].status = IN_SERVICE;
Cluster_Member[node_addr][source].timer = op_sim_time ();

Cluster_Member[node_addr][source].latitude = latitude_receiver;
Cluster_Member[node_addr][source].longitude = longitude_receiver;

Clusterhead[node_addr].Member_id[source] = source;
```

```
printf ("     > Registrating the {{Member %d}}\n",source);
                }
op_pk_destroy (hello_pk_ptr);
}

////////////////////////////////////////////////////////////////////
/********* LORA-CBF_Register_Cluster-head ***************/
////////////////////////////////////////////////////////////////////

void LORA-CBF_register_cluster_head(Packet* hello_pk_ptr, int source, double
latitude_receiver, double longitude_receiver, double latitude, double longitude)
        {
Cluster_Neighbour[node_addr][source].previa_latitude =
        Cluster_Neighbour[node_addr][source].latitude;
Cluster_Neighbour[node_addr][source].previa_longitude =
        Cluster_Neighbour[node_addr][source].longitude;
Cluster_Neighbour[node_addr][source].previo_timer =
        Cluster_Neighbour[node_addr][source].timer;

Cluster_Neighbour[node_addr][source].status = IN_SERVICE;
Cluster_Neighbour[node_addr][source].timer = op_sim_time();

Cluster_Neighbour[node_addr][source].latitude = latitude_receiver;
Cluster_Neighbour[node_addr][source].longitude = longitude_receiver;

Clusterhead[node_addr].Ch[source]= source;

printf ("     > Registrating the {{ Cluster_head Neighbor %d}} **********\n", source);

op_pk_destroy (hello_pk_ptr);
        }
```

```
////////////////////////////////////////////////////////
///*****************  LORA-CBF_data_pk()  *****************/
////////////////////////////////////////////////////////

void LORA-CBF_data_pk(Packet* data_pk_ptr)
        {
int                     dest, nextCluster;
Boolean                 result_cluster = OPC_FALSE;
double                  latitude, longitude, altitude;
double                  x_pos, y_pos, z_pos;
double                  dest_latitude, dest_longitude;
Compcode                comp_code;


/*
/* Receive the upcoming data packet stream from the
/* upper layer and before to send it.
/* First the packet is placed in a queue and send RREQ
*/
// read packet
op_pk_nfd_get(data_pk_ptr,"DEST",&dest);

printf("     - Function LORA-CBF_data_pk(destination %d)\n",dest);

printf ("     > Check Routing table for the destination. \n");
if ((Location[dest].latitude != 0.0 && Location[dest].longitude != 0.0) &&
((op_sim_time () - Location[dest].timer) < EXPIRATION_TIME)  ||
(Cluster_Member[node_addr][dest].latitude != 0.0 &&
Cluster_Member[node_addr][dest].longitude != 0.0) && ((op_sim_time () -
Cluster_Member[node_addr][dest].timer) < EXPIRATION_TIME))
{
LORA-CBF_data_pk_queue(op_pk_copy(data_pk_ptr));

node_id = op_topo_parent (op_id_self());
comp_code = op_ima_obj_pos_get (node_id, &latitude, &longitude, &altitude,
&x_pos, &y_pos, &z_pos);

/* This line tests the completion code, and upon detecting failure, calls op_sim_end () to
*/
/* immediately end the simulation and print and error message to the standard output
device,*/
/* and the opnet message area.                                          */

if (comp_code == OPC_COMPCODE_FAILURE)
op_sim_end ("get attributes failed", "", "", "");

dest_latitude = Location[dest].latitude;
dest_longitude = Location[dest].longitude;
```

```
op_pk_nfd_set (data_pk_ptr,"Dest_latitude",dest_latitude);
op_pk_nfd_set (data_pk_ptr,"Dest_longitude",dest_longitude);
op_pk_nfd_set (data_pk_ptr,"DEST",dest);
op_pk_nfd_set (data_pk_ptr,"SRC",node_addr);
op_pk_nfd_set (data_pk_ptr,"Source_latitude",latitude);
op_pk_nfd_set (data_pk_ptr,"Source_longitude",longitude);
op_pk_nfd_set (data_pk_ptr,"Seq_number",Data_seq_number);
printf("     >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n");

if (nodeid[node_addr].status_of_node == Cluster_head)
        {
if (((Clusterhead[node_addr].Member_id[dest] == dest) || (node_addr == dest)) &&
((op_sim_time () - Cluster_Member[node_addr][dest].timer) < EXPIRATION_TIME))
        {
op_pk_nfd_set (data_pk_ptr, "Next_Cluster",node_addr);

// send data packet
LORA-CBF_pk_send_to_mac_layer(data_pk_ptr,dest);

delay_start = op_sim_time ();
        }
else
{
//Finding the next Cluster_head

nextCluster = nextCluster_dest_from_data(data_pk_ptr, latitude, longitude,
dest_latitude, dest_longitude, dest);

//Finding the next Member to the destination

Next_Member_dest_from_cluster(data_pk_ptr,latitude,longitude,dest_latitude,dest_lon
gitude,nextCluster);

delay_start = op_sim_time ();
                }
        }
else if (nodeid[node_addr].status_of_node == Member)
                        {
printf("     > Looking for the nearest cluster_head to the destination \n");

nextCluster = nextCluster_dest_from_data(data_pk_ptr, latitude, longitude,
dest_latitude, dest_longitude, dest);
result_cluster = predicting_position_next_cluster(nextCluster,latitude,longitude);

if (result_cluster == OPC_TRUE)
        {
```

```
printf ("     > The NextCluster {{%d}} has been seen for the Member {{%d}} \n",
nextCluster, node_addr);

op_pk_nfd_set (data_pk_ptr,"Type_of_node",nodeid[node_addr].status_of_node);
LORA-CBF_pk_send_to_mac_layer(data_pk_ptr,nextCluster);
delay_start = op_sim_time ();
        }
else
        {
printf("     > There is not way to trasmit directly to the next_Cluster {{%d}},\n",
nextCluster);
printf("     > instead we are looking for another gateway !!! \n");

// Finding the nearest Gateway
Next_Member_dest_data(data_pk_ptr,latitude,longitude,dest_latitude,dest_longitude,ne
xtCluster,dest);

delay_start = op_sim_time ();
                }
        }
}

else
        {
printf("     > Data packet is inserted into buffer.  \n");
// insert packet into buffer
LORA-CBF_data_pk_queue(data_pk_ptr);

latency_start = op_sim_time ();

//Generate RREQ
LORA-CBF_rreq_pk_generate(dest);
}

printf("     - Function LORA-CBF_data_pk() done\n");

} // End of LORA-CBF_data_pk
```

```
////////////////////////////////////////////////////////////
/************ BUBLE SORT ********************/
////////////////////////////////////////////////////////////

void bubbleSort(double pos[][2], int max)
{
        int i, j;
        double temp[1][2];

        for (i = 0; i < max; i++)
   {
                for (j = i + 1; j < max; j++)

           {
                        if (pos[j][0] < pos[i][0])

                   {
                        temp[0][0] = pos[i][0];
                        temp[0][1] = pos[i][1];
                        pos[i][0] = pos[j][0];
                        pos[i][1] = pos[j][1];
                        pos[j][0] = temp[0][0];
                        pos[j][1] = temp[0][1];
                }
        }
}
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
// void LORA-CBF_data_pk_queue(Packet* pack)                                    //
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//

void LORA-CBF_data_pk_queue(Packet* pk)
{
int destination;

/*
/* Queue data packet in the corresponding sub-queue.
*/

// Read the DEST field (packet's final destination) in order to have index of subqueue
op_pk_nfd_get(pk,"DEST",&destination);
// Enqueue packet in the correct subqueue
if (op_subq_pk_insert(destination,pk,OPC_QPOS_TAIL) != OPC_QINS_OK)
        {
printf("     > Fail to insert packet in subqueue \n");
op_pk_destroy(pk);
        }
```

```
else
        {
printf("     > %d data packets waiting for destination %d\n", LORA-
CBF_buffer_size_get(destination),destination);
        }
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
// int LORA-CBF_buffer_size_get(int destination)                              //
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//

int LORA-CBF_buffer_size_get(int destination)

        {
/*
/* Return the number of data packets waiting
/* for this destination.
        */
return op_subq_stat(destination, OPC_QSTAT_PKSIZE);
        }


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
// Boolean LORA-CBF_buffer_is_empty(int destination)                          //
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//

Boolean LORA-CBF_buffer_is_empty(int destination)

        {
/*
/* Check whether the given sub-queue (the one whose
/* index equals to destination) is empty or not.
/* Return OPC_TRUE if buffer is empty. OPC_FALSE otherwise.
*/

return op_subq_empty(destination);
        }
```

```
////////////////////////////////////////////////////////
/***************** HANDLE_DATA ***************************/
////////////////////////////////////////////////////////

void LORA-CBF_data_pk_receive(Packet* data_pk_ptr)
        {
// var
int             previousHop, dest, nextCluster;
int             source, seq_number, type_of_node;
double          source_latitude, source_longitude;
double          latitude, longitude, altitude, x_pos, y_pos, z_pos;
double          dest_longitude,dest_latitude;
Boolean         result, result_cluster;
Compcode        comp_code;
Packet*         ack_data_pk_ptr;

/*
/* This function receives the upcoming data packet from the MAC Layer.
*/

// Read data packet
op_pk_nfd_get (data_pk_ptr,"Type_of_node",&type_of_node);
op_pk_nfd_get (data_pk_ptr,"PreviousHop",&previousHop);
op_pk_nfd_get (data_pk_ptr,"SRC",&source);
op_pk_nfd_get (data_pk_ptr,"DEST",&dest);
op_pk_nfd_get (data_pk_ptr,"Next_Cluster",&nextCluster);
op_pk_nfd_get (data_pk_ptr,"Dest_longitude",&dest_longitude);
op_pk_nfd_get (data_pk_ptr,"Dest_latitude", &dest_latitude);
op_pk_nfd_get (data_pk_ptr,"Source_latitude", &source_latitude);
op_pk_nfd_get (data_pk_ptr,"Source_longitude",&source_longitude);
op_pk_nfd_get (data_pk_ptr,"Seq_number",&seq_number);

printf("     - Function LORA-CBF_data_pk_receive(destination is %d ) from (source
%d)\n",dest,source);

node_id = op_topo_parent (op_id_self());
comp_code = op_ima_obj_pos_get (node_id, &latitude, &longitude, &altitude,
&x_pos, &y_pos, &z_pos);

/* This line tests the completion code, and upon detecting failure, calls op_sim_end () to
*/
/* immediately end the simulation and print and error message to the standard output
device,*/
/* and the opnet message area.                                          */

if (comp_code == OPC_COMPCODE_FAILURE)
op_sim_end ("get attributes failed", "", "","");
printf(" >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n");
```

```
// check packet final destination

if (dest == node_addr)
{
// pk has reached its destination
printf("     > Data packet has reached its destination\n");

delay_stop = op_sim_time ();

delay = delay_stop - delay_start;

op_stat_write (average_delay_hndl,delay);

if(LORA-CBF_data_pk_is_fresh_enough(data_pk_ptr) == OPC_TRUE)
{
printf("     > Receiving the {{packet %d}}\n",seq_number);
op_stat_write (pk_received_hndl,seq_number);

latency = 0.0;

// send packet to upper layer
printf("     > Data packet sent to app_manager layer\n");

op_pk_send(data_pk_ptr,DISCARD_STRM);
}
else // DATA PAcket seen less than BROADCAST_RECORD_TIME ago
{

// DATA discarded
printf("     > DATA Packet already has been received\n");

op_pk_destroy(data_pk_ptr);
}
printf("     > The destination node now will send back an acknowledge packet to the
{{source node %d}} \n", source);

// Create ack_data packet
ack_data_pk_ptr = op_pk_create_fmt("ACK_DATA");

op_pk_nfd_set (ack_data_pk_ptr,"DEST", dest);
op_pk_nfd_set (ack_data_pk_ptr,"Dest_latitude",latitude);
op_pk_nfd_set (ack_data_pk_ptr,"Dest_longitude",longitude);
op_pk_nfd_set (ack_data_pk_ptr,"Next_Cluster",nextCluster);
op_pk_nfd_set (ack_data_pk_ptr,"SRC",source);
op_pk_nfd_set (ack_data_pk_ptr,"Source_latitude",source_latitude);
op_pk_nfd_set (ack_data_pk_ptr,"Source_longitude",source_longitude);
op_pk_nfd_set (ack_data_pk_ptr,"Seq_number",seq_number);
```

```
if (nodeid[node_addr].status_of_node == Cluster_head)
{
//Finding the nearest Cluster_head

nextCluster = nextCluster_source_from_ack_data(ack_data_pk_ptr, latitude, longitude,
source_latitude, source_longitude, source);

// Finding the nearest Gateway

Next_Member_source_from_cluster_ack_data(ack_data_pk_ptr,latitude,longitude,sourc
e_latitude,source_longitude,nextCluster);
                }
else if (nodeid[node_addr].status_of_node == Member)
                {
 printf("     > Looking for the nearest cluster_head to the source \n");

nextCluster = nextCluster_source_from_ack_data(ack_data_pk_ptr, latitude, longitude,
source_latitude, source_longitude, source);

result_cluster = predicting_position_next_cluster(nextCluster,latitude,longitude);

if (result_cluster == OPC_TRUE)
                {
printf ("     > The NextCluster {{%d}} has been seen for the Member {{%d}} \n",
nextCluster, node_addr);

LORA-CBF_pk_send_to_mac_layer(ack_data_pk_ptr,nextCluster);
        }
else
        {
printf("     > There is not way to trasmit directly to the next_Cluster {{%d}},\n",
nextCluster);
printf("     > instead we are looking for another gateway !!! \n");

// Finding the nearest Gateway

Next_Member_source_ack_data(ack_data_pk_ptr,latitude,longitude,source_latitude,sou
rce_longitude,nextCluster,source);
                                        }
                                }
        }
else if (nodeid[node_addr].status_of_node == Cluster_head)
        {
printf("     > Predicting the position of the destination \n");

result = predicting_position_destination(dest,latitude,longitude);
if (result == OPC_TRUE)
                {
```

```
printf("     > The destination node is in the same cluster !!! \n");
LORA-CBF_pk_send_to_mac_layer(data_pk_ptr,dest);
                    }
else
                              {
// Finding the next Cluster_head to the destination

nextCluster = nextCluster_dest_from_data(data_pk_ptr, latitude, longitude,
dest_latitude, dest_longitude, dest);

// Finding the nearest Gateway

Next_Member_dest_from_cluster(data_pk_ptr, latitude, longitude, dest_latitude,
dest_longitude, nextCluster);
                              }
                    }
else if ((nodeid[node_addr].status_of_node == Member) && (nextCluster !=
node_addr))
                    {
// Verifying if the next_cluster has been seen less than EXPIRATION_TIME

printf ("     > Verifying if the next_cluster has been seen for the Member less than
EXPIRATION TIME \n");

result_cluster = predicting_position_next_cluster(nextCluster, latitude, longitude);

if (result_cluster == OPC_TRUE)
                    {
printf ("     > The NextCluster {{%d}} has been seen for the Gateway {{%d}} \n",
nextCluster, node_addr);

LORA-CBF_pk_send_to_mac_layer(data_pk_ptr,nextCluster);
                    }
else
                    {
printf("     > There is not way to trasmit directly to the next_Cluster {{%d}},\n",
nextCluster);
printf("     > instead we are looking for another gateway !!! \n");

// Finding the nearest Gateway

Next_Member_dest_data(data_pk_ptr, latitude, longitude, dest_latitude, dest_longitude,
nextCluster,dest);
                              }
          }
printf("     - Function LORA-CBF_data_pk_receive() done\n");
  }
```

```
////////////////////////////////////////////////////////////////
/************** is DATA Packet fresh enough ************************/
////////////////////////////////////////////////////////////////

Boolean LORA-CBF_data_pk_is_fresh_enough(Packet* data_pk_ptr)

{
// var
Boolean     result = OPC_FALSE;
int         source, dest, seq_number;

/*
/* Check whether the BroadcastID of the request
/* is greater than the last seen. If so, RREP is
/* automatically processed. Else, node checks
/* whether RREP packet was seen less than
/* RECORD_BROADCAST_TIME seconds ago. If so, RREP
/* must be discarded. Else, it is processed.
*/

// Read RREQ fields
op_pk_nfd_get (data_pk_ptr,"Seq_number",&seq_number);
op_pk_nfd_get (data_pk_ptr,"SRC",&source);
op_pk_nfd_get (data_pk_ptr,"DEST",&dest);

if((Dataseen[source][dest].sequence_number < seq_number) ||
((Dataseen[source][dest].sequence_number==seq_number) &&
(op_sim_time()> Dataseen[source][dest].expirationTime)))

{
result = OPC_TRUE;
// update RequestSeen repository
Dataseen[source][dest].sequence_number=seq_number;
Dataseen[source][dest].expirationTime=op_sim_time()+BROADCAST_RECORD_TI
ME;
                }
return result;
}
```

```
////////////////////////////////////////////////////////////
/******* HANDLE ACK_DATA ************************/
////////////////////////////////////////////////////////////

void LORA-CBF_ack_data_pk_receive(Packet* ack_data_pk_ptr)
        {
int             source, dest, nextCluster, previous_hop;
int             type_of_node;
double          dest_longitude, dest_latitude, source_latitude, source_longitude;
double          latitude, longitude, altitude, x_pos, y_pos, z_pos;
Ici*            ici_ptr;
Compcode        comp_code;
Boolean         result, result_cluster = OPC_FALSE;
Packet*         data_pk_ptr;

/*
/* This function receives the upcoming data packet from the MAC Layer.
*/

// Read data packet

op_pk_nfd_get (ack_data_pk_ptr,"Type_of_node",&type_of_node);
op_pk_nfd_get (ack_data_pk_ptr,"PreviosHop", &previous_hop);
op_pk_nfd_get (ack_data_pk_ptr,"SRC",&source);
op_pk_nfd_get (ack_data_pk_ptr,"DEST",&dest);
op_pk_nfd_get (ack_data_pk_ptr,"Next_Cluster",&nextCluster);
op_pk_nfd_get (ack_data_pk_ptr,"Dest_longitude",&dest_longitude);
op_pk_nfd_get (ack_data_pk_ptr,"Dest_latitude", &dest_latitude);
op_pk_nfd_get (ack_data_pk_ptr,"Source_latitude", &source_latitude);
op_pk_nfd_get (ack_data_pk_ptr,"Source_longitude",&source_longitude);

printf("     - Function LORA-CBF_ack_data_pk_receive(source is %d ) from
(destination %d)\n",source,dest);

node_id = op_topo_parent (op_id_self());
comp_code = op_ima_obj_pos_get (node_id, &latitude, &longitude, &altitude,
&x_pos, &y_pos, &z_pos);

/* This line tests the completion code, and upon detecting failure, calls op_sim_end () to
*/
/* immediately end the simulation and print and error message to the standard output
device,*/
/* and the opnet message area.                                          */

if (comp_code == OPC_COMPCODE_FAILURE)
op_sim_end ("get attributes failed", "", "", "");
printf(" >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n");
// check packet final destination
```

```
if (source == node_addr)
        {
// pk has reached its destination
printf("     > ACK_DATA packet has reached its destination\n");

if(LORA-CBF_ack_data_pk_is_fresh_enough(ack_data_pk_ptr) == OPC_TRUE
        {
Data_seq_number++;

op_pk_destroy (ack_data_pk_ptr);

data_pk_ptr = LORA-CBF_data_pk_dequeue(dest);

op_pk_destroy (data_pk_ptr);

printf("     > Schedule a notice to serve the buffer \n");
printf("     > Updating the position of the {{Destination node %d}}\n",dest);

Location[dest].latitude = dest_latitude;
Location[dest].longitude = dest_longitude;
Location[dest].timer = op_sim_time ();

// Serve buffer
ici_ptr = op_ici_create ("LORA-CBF_notice_to_serve");
op_ici_attr_set (ici_ptr, "dest", dest);
op_ici_install (ici_ptr);
op_intrpt_schedule_self (op_sim_time(), REP_CODE);
        }
else // ACK DATA PAcket seen less than BROADCAST_RECORD_TIME ago
        {
// ACK DATA discarded
printf("     > ACK DATA Packet already has been received\n");

op_pk_destroy(ack_data_pk_ptr);
        }
}
else if ((nodeid[node_addr].status_of_node == Member) && (nextCluster !=
node_addr))
        {
// Verifying if the next_cluster has been seen less than EXPIRATION_TIME

result_cluster = predicting_position_next_cluster(nextCluster,latitude,longitude);

if (result_cluster == OPC_TRUE)
        {
printf ("     > The NextCluster {{%d}} has been seen for the Gateway {{%d}} \n",
nextCluster, node_addr);
```

```
LORA-CBF_pk_send_to_mac_layer(ack_data_pk_ptr,nextCluster);
        }
else
        {
printf("     > There is not way to trasmit directly to the next_Cluster {{%d}},\n",
nextCluster);

printf("     > instead we are looking for another gateway !!! \n");

// Finding the nearest Gateway

Next_Member_source_ack_data(ack_data_pk_ptr,latitude,longitude,source_latitude,sou
rce_longitude,nextCluster,source);
                }
        }
else if (nodeid[node_addr].status_of_node == Cluster_head)
        {
printf("     > Predicting the position of the source \n");

result = predicting_position_source(source, latitude, longitude);

if (result == OPC_TRUE)
        {
printf("     > ACK_DATA has reached its cluster destination !!! \n");

op_pk_nfd_set (ack_data_pk_ptr, "Next_Cluster",node_addr);

LORA-CBF_pk_send_to_mac_layer(ack_data_pk_ptr,source);
        }
else
                {
printf("     > The {{Cluster_head %d}} is looking for the nearest Cluster to the
{{source %d}}\n", node_addr, source);

nextCluster =
nextCluster_source_from_ack_data(ack_data_pk_ptr,latitude,longitude,source_latitude,
source_longitude,source);

// Finding the nearest Gateway

Next_Member_source_from_cluster_ack_data(ack_data_pk_ptr,latitude,longitude,sourc
e_latitude,source_longitude,nextCluster);
                }
        }
printf("     - Function LORA-CBF ack_data_pk_receive() done\n");
        }
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
// Packet* LORA-CBF_data_pk_dequeue(int destination)                                //
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//

Packet* LORA-CBF_data_pk_dequeue(int destination)
{
Packet* data_pk_ptr;

/*
/* Return the first packet (first in) from
/* the indicated data buffer.
*/

// If buffer not empty
if (LORA-CBF_buffer_is_empty(destination) == OPC_FALSE)
        {
// get data packet from queue
data_pk_ptr = op_subq_pk_remove(destination,OPC_QPOS_HEAD);
        }
else

printf("     > ERROR: buffer empty \n");

return data_pk_ptr;
}

////////////////////////////////////////////////////////////////
/***************** Generate RREQ ************************/
////////////////////////////////////////////////////////////////

void LORA-CBF_rreq_pk_generate(int destination)
{
double                  latitude, longitude, altitude, x_pos, y_pos, z_pos;
Packet*                 rreq_pk_ptr;
Compcode                comp_code;
/*
/* Initiate the discovery process by generating a RREQ for a given
/* destination.
*/
printf("     - Function LORA-CBF_rreq_pk_generate(destination %d)\n", destination);

// Create a request type packet
rreq_pk_ptr = op_pk_create_fmt("LORA-CBF_RREQ");

node_id = op_topo_parent (op_id_self());
comp_code = op_ima_obj_pos_get (node_id, &latitude, &longitude, &altitude,
&x_pos, &y_pos, &z_pos);
```

```
/* This line tests the completion code, and upon detecting failure, calls op_sim_end () to
*/
/* immediately end the simulation and print and error message to the standard output
device,*/
/* and the opnet message area.                                                    */

if (comp_code == OPC_COMPCODE_FAILURE)
op_sim_end ("get attributes failed", "", "","");

// set RREQ fields
op_pk_nfd_set (rreq_pk_ptr,"TTL",RequestSent[destination].ttl);
op_pk_nfd_set (rreq_pk_ptr,"DEST",destination);
op_pk_nfd_set (rreq_pk_ptr,"BroadcastID",myBroadcastID);
op_pk_nfd_set (rreq_pk_ptr,"SRC",node_addr);
op_pk_nfd_set (rreq_pk_ptr,"Source_latitude",latitude);
op_pk_nfd_set (rreq_pk_ptr,"Source_longitude",longitude);

//increment node's boadcastID
myBroadcastID++;

// send to mac layer
printf("     > RREQ packet has been generated \n"); LORA-
CBF_pk_send_to_mac_layer(rreq_pk_ptr, BROADCAST);

printf("     - Function LORA-CBF_rreq_pk_generate() done\n");
        }

////////////////////////////////////////////////////////////
/********************** HANDLE REQUEST ******************/
////////////////////////////////////////////////////////////

void LORA-CBF_rreq_pk_receive(Packet* rreq_pk_ptr)
{
int                 dest, type_of_node, previous_hop;
double              latitude, longitude, altitude, x_pos, y_pos, z_pos;
Boolean             result= OPC_FALSE;
Compcode            comp_code;

/*
/* Process the received RREQ. This procedure decides
/* whether node should generate a RREP packet for the
/* requested destination or simply forward it to the
/* neighbouring nodes.
/* Note that RREQ is automatically discarded if seen
/* less than BROADCAST_RECORD_TIME seconds ago
*/
printf("     - Function LORA-CBF_rreq_pk_receive()\n");
```

```
// reading packet

op_pk_nfd_get (rreq_pk_ptr, "Type_of_node", &type_of_node);
op_pk_nfd_get (rreq_pk_ptr, "PreviousHop", &previous_hop);
op_pk_nfd_get (rreq_pk_ptr, "DEST", &dest);

node_id = op_topo_parent (op_id_self());
comp_code = op_ima_obj_pos_get (node_id, &latitude, &longitude, &altitude,
&x_pos, &y_pos, &z_pos);

/* This line tests the completion code, and upon detecting failure, calls op_sim_end () to
*/
/* immediately end the simulation and print and error message to the standard output
device,*/
/* and the opnet message area.
*/

if (comp_code == OPC_COMPCODE_FAILURE)
op_sim_end ("get attributes failed", "", "","");

if (nodeid[node_addr].status_of_node == Member)
        {
// check if RREQ has already been seen
if(LORA-CBF_rreq_pk_is_fresh_enough(rreq_pk_ptr) == OPC_TRUE)
        {
//request NEVER seen OR SEEN more than BROADCAST_RECORD_TIME ago

printf("     > RREQ is fresh enough: request handled \n");

// the node is an intermediate node

LORA-CBF_rreq_pk_forward(rreq_pk_ptr);
        }
else //request seen less than BROADCAST_RECORD_TIME ago
        {
// RREQ discarded
printf("     > Request seen less than BRODCAST_RECORD_TIME ago: RREQ
discarded\n");
op_pk_destroy(rreq_pk_ptr);
        }
}

else if (nodeid[node_addr].status_of_node == Cluster_head)
{
// check if RREQ has already been seen
if(LORA-CBF_rreq_pk_is_fresh_enough(rreq_pk_ptr) == OPC_TRUE)
        {
//request NEVER seen OR SEEN more than BROADCAST_RECORD_TIME ago
```

```
printf("     > RREQ is fresh enough: request handled \n");

if (((Clusterhead[node_addr].Member_id[dest] == dest) && ((op_sim_time () -
Cluster_Member[node_addr][dest].timer) < EXPIRATION_TIME)) || (node_addr ==
dest))
        {
        result = OPC_TRUE;
        }
else

        {
        }

if (result == OPC_TRUE)
        {
printf("     > RREQ has reached its Cluster_head destination: node replies\n");
LORA-CBF_rrep_pk_generate_from_destination(rreq_pk_ptr);
}
else  // the node is an intermediate node
        {
LORA-CBF_rreq_pk_forward(rreq_pk_ptr);
        }
}
else //request seen less than BROADCAST_RECORD_TIME ago

{
// RREQ discarded
printf("     > Request seen less than BRODCAST_RECORD_TIME ago: RREQ
discarded\n");
op_pk_destroy(rreq_pk_ptr);
}
printf("     - Function LORA-CBF_rreq_pk_receive() done\n");
}

else

        {
        op_pk_destroy (rreq_pk_ptr);
        }
}
```

```
////////////////////////////////////////////////////////////
/************** is request fresh enough ********************/
////////////////////////////////////////////////////////////

Boolean LORA-CBF_rreq_pk_is_fresh_enough(Packet* rreq_pk_ptr)
        {
// var
Boolean    result = OPC_FALSE;
int        source, dest, broadcastID;

/*
/* Check whether the BroadcastID of the request
/* is greater than the last seen. If so, RREQ is
/* automatically processed. Else, node checks
/* whether RREQ packet was seen less than
/* RECORD_BROADCAST_TIME seconds ago. If so, RREQ
/* must be discarded. Else, it is processed.
*/

// Read RREQ fields
op_pk_nfd_get (rreq_pk_ptr,"BroadcastID",&broadcastID);
op_pk_nfd_get (rreq_pk_ptr,"SRC",&source);
op_pk_nfd_get (rreq_pk_ptr,"DEST",&dest);

if((RequestSeen[source][dest].broadcastID < broadcastID) ||
((RequestSeen[source][dest].broadcastID==broadcastID) && (op_sim_time()>
RequestSeen[source][dest].expirationTime)))
                {
result = OPC_TRUE;
// update RequestSeen repository
RequestSeen[source][dest].broadcastID=broadcastID;
RequestSeen[source][dest].expirationTime=op_sim_time()+BROADCAST_RECORD_
TIME;
                }
return result;
        }
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
// void LORA-CBF_rreq_pk_forward(Packet* rreq_pk_ptr)                        //
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//

void LORA-CBF_rreq_pk_forward(Packet* rreq_pk_ptr)
        {
int             ttl, hopCount;
double          latitude, longitude, altitude, x_pos, y_pos, z_pos;
Compcode        comp_code;

/*
/* Node does not have a fresh enough entry
/* (or does not have an entry at all) to answer
/* the received RREQ, so it decides to forward
/* it. Node increments the Hop Count field
/* and decrements the TTL field.
/* Note that packet is sent to mac layer only if
/* TTL > 0.
*/

op_pk_nfd_get (rreq_pk_ptr,"TTL",&ttl);
op_pk_nfd_get (rreq_pk_ptr,"HopCount",&hopCount);

if(ttl > 1)
                {
printf("     > Node forwards request\n",ttl);

node_id = op_topo_parent (op_id_self());
comp_code = op_ima_obj_pos_get (node_id, &latitude, &longitude, &altitude,
&x_pos, &y_pos, &z_pos);

/* This line tests the completion code, and upon detecting failure, calls op_sim_end () to
*/
/* immediately end the simulation and print and error message to the standard output
device,*/
/* and the opnet message area.                                              */

if (comp_code == OPC_COMPCODE_FAILURE)
op_sim_end ("get attributes failed", "", "", "");

op_pk_nfd_set (rreq_pk_ptr,"TTL",ttl-1);
op_pk_nfd_set (rreq_pk_ptr,"HopCount",hopCount+1);

// send to mac
LORA-CBF_pk_send_to_mac_layer(rreq_pk_ptr, BROADCAST);
        }
else
        {
```

```
printf("     > TTL expired: rreq destroyed\n");
op_pk_destroy(rreq_pk_ptr);
        }
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
// void LORA-CBF_rrep_pk_generate_from_destination(Packet* rreq_pk_ptr)  //
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//

void LORA-CBF_rrep_pk_generate_from_destination(Packet* rreq_pk_ptr)
        {

Packet*             rrep_pk_ptr;
int                 previousHop,nextCluster, dest;
int                 source;
double              source_latitude,source_longitude;
double              latitude, longitude, altitude, x_pos, y_pos, z_pos;
double              dest_latitude, dest_longitude;
Compcode            comp_code;

/*
/* This function is called when a RREQ has reached its
/* destination and node wants to reply.
/* Once the RREQ packet is generated, the procedure unicat
/* it back to the node upstream (node from which RREQ was
/* received).
*/

// Read the received RREQ packet
op_pk_nfd_get (rreq_pk_ptr,"PreviousHop",&previousHop);
op_pk_nfd_get (rreq_pk_ptr,"SRC",&source);
op_pk_nfd_get (rreq_pk_ptr,"DEST", &dest);
op_pk_nfd_get (rreq_pk_ptr,"Source_latitude",&source_latitude);
op_pk_nfd_get (rreq_pk_ptr,"Source_longitude",&source_longitude);

// Create reply packet
rrep_pk_ptr = op_pk_create_fmt("LORA-CBF_RREP");

// Assign source and dest to the reply packet
op_pk_nfd_set(rrep_pk_ptr,"HopCount",0);
op_pk_nfd_set(rrep_pk_ptr,"Source_longitude",source_longitude);
op_pk_nfd_set(rrep_pk_ptr,"Source_latitude",source_latitude);
op_pk_nfd_set(rrep_pk_ptr,"SRC",source);
op_pk_nfd_set(rrep_pk_ptr,"DEST", dest);

op_pk_destroy (rreq_pk_ptr);
```

```
 //increment node's boadcastID
mySeqNb++;

op_pk_nfd_set(rrep_pk_ptr,"DestSeqNb",mySeqNb);

printf("     > RREP has been generated from {{Cluster %d}}: unicast it to {{Source
%d}} \n",node_addr,source);

node_id = op_topo_parent (op_id_self());
comp_code = op_ima_obj_pos_get (node_id, &latitude, &longitude, &altitude,
&x_pos, &y_pos, &z_pos);

/* This line tests the completion code, and upon detecting failure, calls op_sim_end () to
*/
/* immediately end the simulation and print and error message to the standard output
device,*/
/* and the opnet message area.                                          */

if (comp_code == OPC_COMPCODE_FAILURE)
op_sim_end ("get attributes failed", "", "","");
printf("     >>\n");

if (Clusterhead[node_addr].Member_id[dest] == dest)
{
dest_latitude = Cluster_Member[node_addr][dest].latitude;
dest_longitude = Cluster_Member[node_addr][dest].longitude;

op_pk_nfd_set(rrep_pk_ptr,"Dest_latitude",dest_latitude);
op_pk_nfd_set(rrep_pk_ptr,"Dest_longitude",dest_longitude);
                }
else if (node_addr == dest)
        {
op_pk_nfd_set(rrep_pk_ptr,"Dest_latitude",latitude);
op_pk_nfd_set(rrep_pk_ptr,"Dest_longitude",longitude);
        }
// Finding the nearest cluster to the source

nextCluster = nextCluster_source_from_rrep(rrep_pk_ptr, latitude, longitude,
source_latitude, source_longitude, source);


// Finding the farest Gateway

Next_Member_source_from_cluster_rrep(rrep_pk_ptr,latitude,longitude,
source_latitude, source_longitude, nextCluster);

        }
```

```
////////////////////////////////////////////////////////////
/********************** HANDLE REPLY ********************/
////////////////////////////////////////////////////////////

void  LORA-CBF_rrep_pk_receive(Packet* rrep_pk_ptr)
{
// var
int             source, nextCluster;
int             hopCount,nextHop, type_of_node, previous_hop;
int             dest;
double          source_latitude,source_longitude;
double          dest_longitude,dest_latitude;
int             seq_nb;
Boolean         result, result_cluster = OPC_FALSE;
double          latitude, longitude, altitude, x_pos, y_pos, z_pos;
Compcode        comp_code;
Packet*         ack_rreq_pk_ptr;
Ici*            ici_ptr;

/*
/* Receive RREP and decide whether to update or create
/* the corresponding entry and whether to forward or
/* discard the received RREP.
*/

// begin
printf("     - Function LORA-CBF_rrep_pk_receive()\n");
// packet read

op_pk_nfd_get (rrep_pk_ptr,"Type_of_node",&type_of_node);
op_pk_nfd_get (rrep_pk_ptr,"PreviousHop",&previous_hop);
op_pk_nfd_get (rrep_pk_ptr,"HopCount",&hopCount);
op_pk_nfd_set (rrep_pk_ptr,"HopCount",hopCount+1);
op_pk_nfd_get (rrep_pk_ptr,"NextHop",&nextHop);
op_pk_nfd_get (rrep_pk_ptr,"HopCount",&hopCount);
op_pk_nfd_get (rrep_pk_ptr,"Dest_longitude",&dest_longitude);
op_pk_nfd_get (rrep_pk_ptr,"Dest_latitude", &dest_latitude);
op_pk_nfd_get (rrep_pk_ptr,"Source_longitude",&source_longitude);
op_pk_nfd_get (rrep_pk_ptr,"Source_latitude",&source_latitude);
op_pk_nfd_get (rrep_pk_ptr,"SRC",&source);
op_pk_nfd_get (rrep_pk_ptr,"DEST",&dest);
op_pk_nfd_get (rrep_pk_ptr,"Next_Cluster",&nextCluster);
op_pk_nfd_get (rrep_pk_ptr,"DestSeqNb",&seq_nb);

node_id = op_topo_parent (op_id_self());
comp_code = op_ima_obj_pos_get (node_id, &latitude, &longitude, &altitude,
&x_pos, &y_pos, &z_pos);
```

```
/* This line tests the completion code, and upon detecting failure, calls op_sim_end () to
*
/* immediately end the simulation and print and error message to the standard output
device,*/
/* and the opnet message area.                                                    */

if (comp_code == OPC_COMPCODE_FAILURE)
op_sim_end ("get attributes failed", "", "","");

printf(" >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n");

if (node_addr == source)
        {
// check if RREP has already been seen
if(LORA-CBF_rrep_pk_is_fresh_enough(rrep_pk_ptr) == OPC_TRUE)
                {
//request NEVER seen OR SEEN more than BROADCAST_RECORD_TIME ago

printf("     > RREP is fresh enough: request handled \n");

// destroy RREP

 printf("     > Reply has reached its cluster destination !!! \n");
op_pk_destroy (rrep_pk_ptr);

printf("     > Schedule a notice to serve the buffer \n");

printf("     > Updating the position of the {{Destination node %d}}\n",dest);

Location[dest].latitude = dest_latitude;
Location[dest].longitude = dest_longitude;
Location[dest].timer = op_sim_time ();

// Serve buffer

ici_ptr = op_ici_create ("LORA-CBF_notice_to_serve");
op_ici_attr_set (ici_ptr, "dest", dest);
op_ici_install (ici_ptr);
op_intrpt_schedule_self (op_sim_time(), REP_CODE);

printf ("     - Function LORA-CBF_rrep_pk_receive ()  done \n ");
                }
else //Reply seen less than BROADCAST_RECORD_TIME ago
                {
// RREP discarded
printf("     > Reply seen less than BRODCAST_RECORD_TIME ago: RREP
discarded\n");
op_pk_destroy(rrep_pk_ptr);
```

```
                }
        }
else if (nodeid[node_addr].status_of_node == Member)
        {
// Verifying if the next_cluster has been seen less than EXPIRATION_TIME

printf ("     > Verifying if the next_cluster has been seen for the Member less than
EXPIRATION TIME \n");

result_cluster = predicting_position_next_cluster(nextCluster, latitude, longitude);

if (result_cluster == OPC_TRUE)
                {
printf ("     > The NextCluster {{%d}} has been seen for the Member {{%d}} \n",
nextCluster, node_addr);

LORA-CBF_pk_send_to_mac_layer(rrep_pk_ptr,nextCluster);
                }
else
                {
printf("     > There is not way to trasmit directly to the next_Cluster {{%d}},\n",
nextCluster);
printf("     > instead we are looking for another gateway !!! \n");

Next_Member_source_rrep(rrep_pk_ptr,latitude,longitude, source_latitude,
source_longitude, nextCluster, source);
                }
        }
else if (nodeid[node_addr].status_of_node == Cluster_head)
        {
if (((Clusterhead[node_addr].Member_id[source] == source) || (node_addr == source))
&& ((op_sim_time () - Cluster_Member[node_addr][source].timer) <
EXPIRATION_TIME))
                        {
result = OPC_TRUE;
                        }
else
        {
        }
if (result == OPC_TRUE)
        {
// check if RREP has already been seen
if(LORA-CBF_rrep_pk_is_fresh_enough(rrep_pk_ptr) == OPC_TRUE)
        {
//request NEVER seen OR SEEN more than BROADCAST_RECORD_TIME ago

printf("     > RREP is fresh enough: request handled \n");
```

```
// destroy RREP
printf("     > Reply has reached its cluster destination !!! \n");

// Create ack_rreq packet
ack_rreq_pk_ptr = op_pk_create_fmt("ACK_RREQ");

op_pk_nfd_set (ack_rreq_pk_ptr,"DEST", dest);
op_pk_nfd_set (ack_rreq_pk_ptr,"Dest_latitude",dest_latitude);
op_pk_nfd_set (ack_rreq_pk_ptr,"Dest_longitude",dest_longitude);
op_pk_nfd_set (ack_rreq_pk_ptr,"Next_Cluster",node_addr);
op_pk_nfd_set (ack_rreq_pk_ptr,"SRC",source);
op_pk_nfd_set (ack_rreq_pk_ptr,"DestSeqNb",seq_nb);

LORA-CBF_pk_send_to_mac_layer(ack_rreq_pk_ptr,source);

op_pk_destroy (rrep_pk_ptr);

printf ("     - Function LORA-CBF_rrep_pk_receive ()  done \n ");
                }
else //reply seen less than BROADCAST_RECORD_TIME ago
                {
// RREP discarded
printf("     > Reply seen less than BRODCAST_RECORD_TIME ago: RREP
discarded\n");
op_pk_destroy(rrep_pk_ptr);
}
}
else
                {
printf("     > The {{Cluster_head %d}} is looking for the nearest Cluster to the
{{source %d}}\n", node_addr, source);

nextCluster = nextCluster_source_from_rrep(rrep_pk_ptr, latitude, longitude,
source_latitude, source_longitude, source);

// Finding the nearest Gateway

Next_Member_source_from_cluster_rrep(rrep_pk_ptr,latitude,longitude,
source_latitude, source_longitude, nextCluster);
                }
        }
else

        {
op_pk_destroy (rrep_pk_ptr);
        }
}
```

```
////////////////////////////////////////////////////////////
/************** is reply fresh enough **********************/
////////////////////////////////////////////////////////////

Boolean LORA-CBF_rrep_pk_is_fresh_enough(Packet* rrep_pk_ptr)

        {
// var
Boolean     result = OPC_FALSE;
int         source, dest, myseqNb;
/*
/* Check whether the BroadcastID of the request
/* is greater than the last seen. If so, RREP is
/* automatically processed. Else, node checks
/* whether RREP packet was seen less than
/* RECORD_BROADCAST_TIME seconds ago. If so, RREP
/* must be discarded. Else, it is processed.
*/

// Read RREP fields
op_pk_nfd_get (rrep_pk_ptr,"DestSeqNb",&myseqNb);
op_pk_nfd_get (rrep_pk_ptr,"SRC",&source);
op_pk_nfd_get (rrep_pk_ptr,"DEST",&dest);

if((ReplySeen[source][dest].sequence_number < myseqNb) ||
((ReplySeen[source][dest].sequence_number==myseqNb) && (op_sim_time()>
ReplySeen[source][dest].expirationTime)))
                {
result = OPC_TRUE;
// update ReplySeen repository
ReplySeen[source][dest].sequence_number=myseqNb;

ReplySeen[source][dest].expirationTime=op_sim_time()+BROADCAST_RECORD_TI
ME;
                }
return result;
        }
```

```
////////////////////////////////////////////////////////
/********************** BUBLE SORT **********************/
////////////////////////////////////////////////////////

void bubbleSortfar(double pos[][2], int max)

{
int             i, j;
double          temp[1][2];

for (i = 0; i < max; i++)

   {
for (j = i + 1; j < max; j++)

          {
if (pos[j][0] > pos[i][0])

                  {
                        temp[0][0] = pos[i][0];
                        temp[0][1] = pos[i][1];
                        pos[i][0] = pos[j][0];
                        pos[i][1] = pos[j][1];
                        pos[j][0] = temp[0][0];
                        pos[j][1] = temp[0][1];
                        }
               }
        }
}

 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
 //       int LORA-CBF_buffer_serve(int destination)                          //
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//

void LORA-CBF_buffer_serve(int destination)

{
Packet*         data_pk_ptr;
int             nextCluster;
double          latitude, longitude, altitude, x_pos, y_pos, z_pos;
double          dest_latitude, dest_longitude;
Boolean         result_cluster = OPC_FALSE;
Compcode        comp_code;

/*
/* This routine is called when a new entry is available for
/* a given destination.
*/
```

```
printf("     - Function LORA-CBF_buffer_serve(destination %d)\n",destination);

node_id = op_topo_parent (op_id_self());
comp_code = op_ima_obj_pos_get (node_id, &latitude, &longitude, &altitude,
&x_pos, &y_pos, &z_pos);

/* This line tests the completion code, and upon detecting failure, calls op_sim_end () to
*/
/* immediately end the simulation and print and error message to the standard output
device,*/
/* and the opnet message area.                                              */

if (comp_code == OPC_COMPCODE_FAILURE)
op_sim_end ("get attributes failed", "", "","");

latency_stop = op_sim_time ();

latency = latency_stop - latency_start;

op_stat_write (latency_hndl, latency);

// get packet from sub-queue

printf("     > Get packet from buffer\n");

data_pk_ptr = op_pk_create_fmt("LORA-CBF_DATA");

data_pk_ptr = LORA-CBF_data_pk_dequeue(destination);
LORA-CBF_data_pk_queue(op_pk_copy (data_pk_ptr));

dest_latitude = Location[destination].latitude;
dest_longitude = Location[destination].longitude;

op_pk_nfd_set (data_pk_ptr,"Dest_latitude",dest_latitude);
op_pk_nfd_set (data_pk_ptr,"Dest_longitude",dest_longitude);
op_pk_nfd_set (data_pk_ptr,"DEST",destination);
op_pk_nfd_set (data_pk_ptr,"SRC",node_addr);
op_pk_nfd_set (data_pk_ptr,"Source_latitude",latitude);
op_pk_nfd_set (data_pk_ptr,"Source_longitude",longitude);
op_pk_nfd_set (data_pk_ptr,"Seq_number",Data_seq_number);

printf(" >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n");

if (nodeid[node_addr].status_of_node == Cluster_head)
        {
if (((Clusterhead[node_addr].Member_id[destination] == destination) || (node_addr ==
destination)) && ((op_sim_time () - Cluster_Member[node_addr][destination].timer) <
EXPIRATION_TIME))
```

```
                {
op_pk_nfd_set (data_pk_ptr, "Next_Cluster",node_addr);

// send data packet
LORA-CBF_pk_send_to_mac_layer(data_pk_ptr,destination);

delay_start = op_sim_time ();
                }

else
                {
//Finding the next Cluster_head

nextCluster = nextCluster_dest_from_data(data_pk_ptr, latitude, longitude,
dest_latitude, dest_longitude, destination);

//Finding the next Member to the destination

Next_Member_dest_from_cluster(data_pk_ptr,latitude,longitude,dest_latitude,dest_lon
gitude,nextCluster);

delay_start = op_sim_time ();
                }
        }
else if (nodeid[node_addr].status_of_node == Member)
        {
printf("     > Looking for the nearest cluster_head to the destination \n");

nextCluster = nextCluster_dest_from_data(data_pk_ptr, latitude, longitude,
dest_latitude, dest_longitude, destination);
result_cluster = predicting_position_next_cluster(nextCluster,latitude,longitude);

if (result_cluster == OPC_TRUE)
                {
printf ("     > The NextCluster {{%d}} has been seen for the Member {{%d}} \n",
nextCluster, node_addr);

LORA-CBF_pk_send_to_mac_layer(data_pk_ptr,nextCluster);

delay_start = op_sim_time ();
                }

else

                {
printf("     > There is not way to trasmit directly to the next_Cluster {{%d}},\n",
nextCluster);
printf("     > instead we are looking for another gateway !!! \n");
```

```
// Finding the nearest Gateway

Next_Member_dest_data(data_pk_ptr,latitude,longitude,dest_latitude,dest_longitude,ne
xtCluster, destination);

delay_start = op_sim_time ();

                }
        }

printf("     - Function LORA-CBF_buffer_serve() done\n");

}

////////////////////////////////////////////////////////////
/************** is reply fresh enough **********************/
////////////////////////////////////////////////////////////

Boolean LORA-CBF_ack_rreq_pk_is_fresh_enough(Packet* ack_rreq_pk_ptr)

        {

// var
Boolean    result = OPC_FALSE;
int        source, dest, myseqNb;
/*
/* Check whether the BroadcastID of the request
/* is greater than the last seen. If so, RREP is
/* automatically processed. Else, node checks
/* whether RREP packet was seen less than
/* RECORD_BROADCAST_TIME seconds ago. If so, RREP
/* must be discarded. Else, it is processed.
*/

// Read RREP fields
op_pk_nfd_get (ack_rreq_pk_ptr,"DestSeqNb",&myseqNb);
op_pk_nfd_get (ack_rreq_pk_ptr,"SRC",&source);
op_pk_nfd_get (ack_rreq_pk_ptr,"DEST",&dest);

if((ReplySeen[source][dest].sequence_number < myseqNb) ||
((ReplySeen[source][dest].sequence_number==myseqNb) && (op_sim_time()>
ReplySeen[source][dest].expirationTime)))
                {
result = OPC_TRUE;
// update ReplySeen repository
ReplySeen[source][dest].sequence_number=myseqNb;
```

```
ReplySeen[source][dest].expirationTime=op_sim_time()+BROADCAST_RECORD_TI
ME;
                }
return result;
        }

////////////////////////////////////////////////////////////////////
/************** HANDLE ACK_RREQ *****************/
////////////////////////////////////////////////////////////////////

void LORA-CBF_ack_rreq_pk_receive(Packet* ack_rreq_pk_ptr)
{
// var

int                     cluster;
int                     dest;
double                  dest_longitude,dest_latitude;
Ici*                    ici_ptr;

/*
/* Receive ack_rreq from its cluster_head
*/

// begin
printf("     - Function LORA-CBF_ack_rreq_pk_receive()\n");
// packet read
op_pk_nfd_get (ack_rreq_pk_ptr,"DEST",&dest);
op_pk_nfd_get (ack_rreq_pk_ptr,"Dest_longitude", &dest_longitude);
op_pk_nfd_get (ack_rreq_pk_ptr,"Dest_latitude", &dest_latitude);
op_pk_nfd_get (ack_rreq_pk_ptr,"Next_Cluster", &cluster);

// check if ACK_RREQ has already been seen
if(LORA-CBF_ack_rreq_pk_is_fresh_enough(ack_rreq_pk_ptr) == OPC_TRUE)
{
printf("     > ack_rreq has reached its source !!!\n");

op_pk_destroy (ack_rreq_pk_ptr);

printf("     > Schedule a notice to serve the buffer \n");

printf("     > Updating the position of the {{Destination node %d}}\n",dest);

Location[dest].latitude = dest_latitude;
Location[dest].longitude = dest_longitude;
Location[dest].timer = op_sim_time ();
nodeid[node_addr].cluster_head = cluster;

// Serve buffer
```

```
ici_ptr = op_ici_create ("LORA-CBF_notice_to_serve");
op_ici_attr_set (ici_ptr, "dest", dest);
op_ici_install (ici_ptr);
op_intrpt_schedule_self (op_sim_time(), REP_CODE);

printf("     - Function LORA-CBF_ack_rreq_pk_receive() done\n");
        }
else //ACK_RREQ seen less than BROADCAST_RECORD_TIME ago
{
// ACK_RREQ discarded
printf("     > ACK_RREQ seen less than BRODCAST_RECORD_TIME ago:
ACK_RREQ discarded\n");
op_pk_destroy(ack_rreq_pk_ptr);
        }
}

 ////////////////////////////////////////////////////////////////
 /********** NextCluster Source ****************/
////////////////////////////////////////////////////////////////

int nextCluster_source_from_rrep(Packet* rrep_pk_ptr, double latitude, double
longitude, double source_latitude, double source_longitude, int source)
        {
int             max, j,i, nextCluster;
double          latitude_neighbour, longitude_neighbour;
double          dist_neighbour_source;
double          pos[N][2];

max = j = 0;

for (i=0; i<N; i++)
                {
if ((Cluster_Neighbour[node_addr][i].status == IN_SERVICE) && ((op_sim_time () -
Cluster_Neighbour[node_addr][i].timer) < EXPIRATION_TIME))
        {
latitude_neighbour = Cluster_Neighbour[node_addr][i].latitude;
longitude_neighbour = Cluster_Neighbour[node_addr][i].longitude;

printf("     > Cluster_neighbour %d\n", i);

dist_neighbour_source = sqrt((longitude_neighbour-
source_longitude)*(longitude_neighbour-source_longitude) + (latitude_neighbour-
source_latitude)*(latitude_neighbour-source_latitude));

//dist_node_source = sqrt((longitude-source_longitude)*(longitude-source_longitude) +
(latitude-source_latitude)*(latitude-source_latitude));

pos[j][0]= dist_neighbour_source;
```

```
pos[j][1] = (double)i;
j++;
max = j;

        }
}
if (max == 0)
        {
printf ("     > There is not another Cluster_head to transmit the packet !!!  \n");

        }
else if (max >= 1)
        {
bubbleSort(pos, max);
nextCluster = pos[0][1];

printf("     > The nearest Cluster_head to the source {{%d}} is {{%d}} \n", source,
nextCluster);

op_pk_nfd_set (rrep_pk_ptr, "Next_Cluster",nextCluster);
        }
return nextCluster;
}

 ////////////////////////////////////////////////////////////////
 /************ Next Member Source ****************/
////////////////////////////////////////////////////////////////

void Next_Member_source_rrep(Packet* rrep_pk_ptr, double latitude, double
longitude, double source_latitude, double source_longitude, int nextCluster, int source)
{
int             max, i, j, nextHop;
double          latitude_gateway, longitude_gateway, factor;
double          dist_next_gateway_source;
double          interval_time, interval_latitude, interval_longitude;
double          future_latitude, future_longitude, dist_predicted;
double          pos[N][2];

max = j = 0;

for (i=0; i<N; i++)
        {
if ((Member_Neighbour[node_addr][i].status == IN_SERVICE) && ((op_sim_time () -
Member_Neighbour[node_addr][i].timer) < EXPIRATION_TIME))
                {
latitude_gateway = Member_Neighbour[node_addr][i].latitude;
longitude_gateway = Member_Neighbour[node_addr][i].longitude;
```

```
printf("      > Member %d \n", i);

//dist_node_source = sqrt((longitude-source_longitude)*(longitude-source_longitude) +
(latitude-source_latitude)*(latitude-source_latitude));

dist_next_gateway_source = sqrt((longitude_gateway-
source_longitude)*(longitude_gateway-source_longitude) + (latitude_gateway-
source_latitude)*(latitude_gateway-source_latitude));

//dist_node_next_gateway = sqrt((longitude-longitude_gateway)*(longitude-
longitude_gateway) + (latitude-latitude_gateway)*(latitude-latitude_gateway));

pos[j][0]= dist_next_gateway_source;

pos[j][1] = (double)i;
j++;
max = j;
        }
}
if (max == 0)
        {
        }
else if (max >= 1)
        {
bubbleSort(pos, max);

printf("      > Predicting the position of the next Hop \n");

for (i=0; i<j; i++)
                {
nextHop = pos[i][1];

printf("      > nextHop =%d\n",nextHop);

if ((Member_Neighbour[node_addr][nextHop].previa_latitude != 0.0) && (nextCluster
!= nextHop))
                {
printf("      > next Hop has previa latitude %d %f\n", nextHop,
Member_Neighbour[node_addr][nextHop].previa_latitude);

interval_time = Member_Neighbour[node_addr][nextHop].timer -
Member_Neighbour[node_addr][nextHop].previo_timer;
interval_latitude = (Member_Neighbour[node_addr][nextHop].latitude -
Member_Neighbour[node_addr][nextHop].previa_latitude)/interval_time;
interval_longitude = (Member_Neighbour[node_addr][nextHop].longitude -
Member_Neighbour[node_addr][nextHop].previa_longitude)/interval_time;
```

```
if ((op_sim_time () - Member_Neighbour[node_addr][nextHop].timer) >
MINIMUM_DELAY)
                                {
printf("     > Next Hop has been seen %d %f\n", nextHop,
Member_Neighbour[node_addr][nextHop].timer);

factor = factor_Member_Neighbour(nextHop);

future_latitude = (Member_Neighbour[node_addr][nextHop].latitude +
(interval_latitude * factor));
future_longitude = (Member_Neighbour[node_addr][nextHop].longitude +
(interval_longitude * factor));
dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("     > Next Hop has distance %d %f\n", nextHop, dist_predicted);

if ((dist_predicted < Range_predicted) || (Range_predicted == dist_predicted))
        {
nextHop = pos[i][1];
break;
        }
}
else
{
printf("     > Next Hop has been seen %d %f\n", nextHop,
Member_Neighbour[node_addr][nextHop].timer);

future_latitude = Member_Neighbour[node_addr][nextHop].latitude;
future_longitude = Member_Neighbour[node_addr][nextHop].longitude;

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("     > Next Hop has distance %d %f\n", nextHop, dist_predicted);

if ((dist_predicted < Range_predicted) || (Range_predicted == dist_predicted))
        {
nextHop = pos[i][1];
break;
        }
}
}
else
        {
        }
}
        }
```

```
printf("     > The nearest Gateway to the Next_Cluster {{%d}} is {{%d}}
\n",nextCluster, nextHop);
 // send reply packet
 LORA-CBF_pk_send_to_mac_layer(rrep_pk_ptr,nextHop);
}

 ////////////////////////////////////////////////////////////////////////////
 /************ Next Member Source from Cluster ************/
 ////////////////////////////////////////////////////////////////////////////

 void Next_Member_source_from_cluster_rrep(Packet* rrep_pk_ptr, double latitude,
 double longitude, double source_latitude, double source_longitude, int nextCluster)
 {
 int             max, i, j, nextHop;
 double          latitude_gateway, longitude_gateway, factor;
 double          dist_next_gateway_source;
 double interval_time, interval_latitude, interval_longitude;
 double          future_latitude, future_longitude, dist_predicted;
 double pos[N][2];

 max = j = 0;

 for (i=0; i<N; i++)
        {
 if ((Cluster_Member[node_addr][i].status == IN_SERVICE) && ((op_sim_time () -
 Cluster_Member[node_addr][i].timer) < EXPIRATION_TIME))
                {
 latitude_gateway = Cluster_Member[node_addr][i].latitude;
 longitude_gateway = Cluster_Member[node_addr][i].longitude;

 printf("     > Member %d \n", i);

 //dist_node_source = sqrt((longitude-source_longitude)*(longitude-source_longitude) +
 (latitude-source_latitude)*(latitude-source_latitude));

 dist_next_gateway_source = sqrt((longitude_gateway-
 source_longitude)*(longitude_gateway-source_longitude) +
 (latitude_gateway-source_latitude)*(latitude_gateway-source_latitude));

 //dist_node_next_gateway = sqrt((longitude-longitude_gateway)*(longitude-
 longitude_gateway) + (latitude-latitude_gateway)*(latitude-latitude_gateway));

 pos[j][0]= dist_next_gateway_source;
 pos[j][1] = (double)i;
 j++;
 max = j;
                }
        }
```

```
if (max == 0)

        {
        }
else if (max >= 1)
        {
bubbleSort(pos, max);

printf("     > Predicting the position of the next Hop \n");

for (i=0; i<j; i++)
                {
nextHop = pos[i][1];

printf("     > Next Hop =%d\n", nextHop);

if ((Cluster_Member[node_addr][nextHop].previa_latitude != 0.0) && (nextCluster !=
nextHop))
                        {
printf("     > Next Hop has previa latitude %d %f\n", nextHop,
Cluster_Member[node_addr][nextHop].previa_latitude);

interval_time = Cluster_Member[node_addr][nextHop].timer -
Cluster_Member[node_addr][nextHop].previo_timer;
interval_latitude = (Cluster_Member[node_addr][nextHop].latitude -
Cluster_Member[node_addr][nextHop].previa_latitude)/interval_time;
interval_longitude = (Cluster_Member[node_addr][nextHop].longitude -
Cluster_Member[node_addr][nextHop].previa_longitude)/interval_time;

if ((op_sim_time () - Cluster_Member[node_addr][nextHop].timer) >
MINIMUM_DELAY)
                {
printf("     > Next Hop has been seen %d %f \n", nextHop,
Cluster_Member[node_addr][nextHop].timer);

factor = factor_Cluster_Member(nextHop);

future_latitude = (Cluster_Member[node_addr][nextHop].latitude + (interval_latitude *
factor));
future_longitude = (Cluster_Member[node_addr][nextHop].longitude +
(interval_longitude * factor));

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("     > Next Hop =%d, distance =%.16f\n", nextHop, dist_predicted);

if ((dist_predicted < Range_predicted) || (dist_predicted == Range_predicted))
```

```
                                        {
nextHop = pos[i][1];
break;
                                        }
                                }

else
                                {
printf("      > Next Hop has been seen %d %f \n", nextHop,
Cluster_Member[node_addr][nextHop].timer);

future_latitude = Cluster_Member[node_addr][nextHop].latitude;
future_longitude = Cluster_Member[node_addr][nextHop].longitude;

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("      > Next Hop =%d, distance =%f\n", nextHop, dist_predicted);

if ((dist_predicted < Range_predicted) || (dist_predicted == Range_predicted))
                                        {
nextHop = pos[i][1];
break;
                                        }
                                }
                        }

else
                        {

                        }
                }
        }
printf("      > The nearest Gateway to the Next_Cluster {{%d}} is {{%d}}
\n",nextCluster, nextHop);

// send reply packet
LORA-CBF_pk_send_to_mac_layer(rrep_pk_ptr,nextHop);
}
```

```
////////////////////////////////////////////////////////////////////////////
/********** Predicting the position next Cluster **************/
////////////////////////////////////////////////////////////////////////////

Boolean predicting_position_next_cluster(int nextCluster, double latitude, double
longitude)
        {
double interval_time, interval_latitude, interval_longitude, factor;
double          future_latitude, future_longitude, dist_predicted;
Boolean         result_cluster = OPC_FALSE;

if ((Cluster_Neighbour[node_addr][nextCluster].status == IN_SERVICE) &&
((op_sim_time () - Cluster_Neighbour[node_addr][nextCluster].timer) <
EXPIRATION_TIME))
                {
printf("     > Predicting the position of the nextCluster %d\n", nextCluster);

if (Cluster_Neighbour[node_addr][nextCluster].previa_latitude != 0.0)
                        {
printf("     > Next Cluster has previa latitude %d %f\n", nextCluster,
Cluster_Neighbour[node_addr][nextCluster].previa_latitude);

interval_time = Cluster_Neighbour[node_addr][nextCluster].timer -
Cluster_Neighbour[node_addr][nextCluster].previo_timer;
interval_latitude = (Cluster_Neighbour[node_addr][nextCluster].latitude -
Cluster_Neighbour[node_addr][nextCluster].previa_latitude)/interval_time;

interval_longitude = (Cluster_Neighbour[node_addr][nextCluster].longitude -
Cluster_Neighbour[node_addr][nextCluster].previa_longitude)/interval_time;

if ((op_sim_time () - Cluster_Neighbour[node_addr][nextCluster].timer) >
MINIMUM_DELAY)
                                {
printf("     > Next Cluster has been seen %d %f\n", nextCluster,
Cluster_Neighbour[node_addr][nextCluster].timer);

factor = factor_Cluster_Neighbour(nextCluster);

future_latitude = (Cluster_Neighbour[node_addr][nextCluster].latitude +
(interval_latitude * factor));

future_longitude = (Cluster_Neighbour[node_addr][nextCluster].longitude +
(interval_longitude * factor));

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) *  (longitude - future_longitude));

printf("     > Next Cluster has distance %d %f\n", nextCluster, dist_predicted);
```

```
        }

else
        {
printf("     > Next Cluster has been seen %d %f\n", nextCluster,
Cluster_Neighbour[node_addr][nextCluster].timer);

future_latitude = Cluster_Neighbour[node_addr][nextCluster].latitude;
future_longitude = Cluster_Neighbour[node_addr][nextCluster].longitude;

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) *  (longitude - future_longitude));

printf("     > Next Cluster has distance %d %f\n", nextCluster, dist_predicted);
                                        }
                                }
else
                                {
printf("     > Next Cluster has previa latitude %d %f\n", nextCluster,
Cluster_Neighbour[node_addr][nextCluster].previa_latitude);

future_latitude = Cluster_Neighbour[node_addr][nextCluster].latitude;
future_longitude = Cluster_Neighbour[node_addr][nextCluster].longitude;

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) *  (longitude - future_longitude));

printf("     > Next Cluster has distance %d %f\n", nextCluster, dist_predicted);
                                }
                        }
if ((dist_predicted < Range_predicted) || (Range_predicted == dist_predicted))

                                {
result_cluster = OPC_TRUE;
                                }
return result_cluster;
                        }
```

```
////////////////////////////////////////////////////////////////////////
/************ NextCluster Destination ******************/
////////////////////////////////////////////////////////////////////////

int nextCluster_dest_from_data(Packet* data_pk_ptr, double latitude, double longitude,
double dest_latitude, double dest_longitude, int destination)
        {
int             max, j,i, nextCluster;
double          latitude_neighbour, longitude_neighbour;
double          dist_neighbour_dest;
double          pos[N][2];

max = j = 0;

for (i=0; i<N; i++)
                {
if ((Cluster_Neighbour[node_addr][i].status == IN_SERVICE) && ((op_sim_time () -
Cluster_Neighbour[node_addr][i].timer) < EXPIRATION_TIME))
                        {
latitude_neighbour = Cluster_Neighbour[node_addr][i].latitude;
longitude_neighbour = Cluster_Neighbour[node_addr][i].longitude;

printf("     > Cluster_neighbour %d\n", i);

dist_neighbour_dest = sqrt((longitude_neighbour-
dest_longitude)*(longitude_neighbour-dest_longitude) + (latitude_neighbour-
dest_latitude)*(latitude_neighbour-dest_latitude));

//dist_node_dest = sqrt((longitude-dest_longitude)*(longitude-dest_longitude) +
(latitude-dest_latitude)*(latitude-dest_latitude));

pos[j][0]= dist_neighbour_dest;
pos[j][1] = (double)i;
j++;
max = j;
                        }
                }

if (max == 0)
                {
printf ("     > There is not another Cluster_head to transmit the packet !!!  \n");
                }
else if (max >= 1)
                {
bubbleSort(pos, max);

nextCluster = pos[0][1];
```

```
printf("     > The nearest Cluster_head to the destination {{%d}} is {{%d}}
\n",destination, nextCluster);

op_pk_nfd_set (data_pk_ptr, "Next_Cluster",nextCluster);
                }
return nextCluster;
        }

 ////////////////////////////////////////////////////////////////////////////////
/***** Next Member Destination from Cluster ************/
////////////////////////////////////////////////////////////////////////////////

void Next_Member_dest_from_cluster(Packet* data_pk_ptr, double latitude, double
longitude, double dest_latitude, double dest_longitude, int nextCluster)

{

int             max, i, j, nextHop;
double          latitude_gateway, longitude_gateway, factor;
double          dist_next_gateway_dest;
double interval_time, interval_latitude, interval_longitude;
double          future_latitude, future_longitude, dist_predicted;
double pos[N][2];

max = j = 0;

for (i=0; i<N; i++)
        {
if ((Cluster_Member[node_addr][i].status == IN_SERVICE) && ((op_sim_time () -
Cluster_Member[node_addr][i].timer) < EXPIRATION_TIME))
                {
latitude_gateway = Cluster_Member[node_addr][i].latitude;
longitude_gateway = Cluster_Member[node_addr][i].longitude;

printf("     > Member %d \n", i);

//dist_node_dest = sqrt((longitude-dest_longitude)*(longitude-dest_longitude) +
(latitude-dest_latitude)*(latitude-dest_latitude));

dist_next_gateway_dest = sqrt((longitude_gateway-
dest_longitude)*(longitude_gateway-dest_longitude) + (latitude_gateway-
dest_latitude)*(latitude_gateway-dest_latitude));

//dist_node_next_gateway = sqrt((longitude-longitude_gateway)*(longitude-
longitude_gateway) + (latitude-latitude_gateway)*(latitude-latitude_gateway));

pos[j][0]= dist_next_gateway_dest;
pos[j][1] = (double)i;
```

```
j++;
max = j;
                }
        }
if (max == 0)
        {
        }
else if (max >= 1)
        {
bubbleSort(pos, max);

printf("     > Predicting the position of the next Hop \n");

for (i=0; i<j; i++)
                {
nextHop = pos[i][1];

printf("     > nextHop %d\n", nextHop);

if ((Cluster_Member[node_addr][nextHop].previa_latitude != 0.0) && (nextCluster !=
nextHop))
                        {
printf("     > next Hop has previa latitude %d %f\n",nextHop,
Cluster_Member[node_addr][nextHop].previa_latitude);

interval_time = Cluster_Member[node_addr][nextHop].timer -
Cluster_Member[node_addr][nextHop].previo_timer;

interval_latitude = (Cluster_Member[node_addr][nextHop].latitude -
Cluster_Member[node_addr][nextHop].previa_latitude)/interval_time;

interval_longitude = (Cluster_Member[node_addr][nextHop].longitude -
Cluster_Member[node_addr][nextHop].previa_longitude)/interval_time;

if ((op_sim_time () - Cluster_Member[node_addr][nextHop].timer) >
MINIMUM_DELAY)
                                {
printf("     > next Hop has been seen %d %f\n", nextHop,
Cluster_Member[node_addr][nextHop].timer);
factor = factor_Cluster_Member(nextHop);

future_latitude = (Cluster_Member[node_addr][nextHop].latitude + (interval_latitude *
factor));

future_longitude = (Cluster_Member[node_addr][nextHop].longitude +
(interval_longitude * factor));
```

```
dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("      > next Hop has distance %d %f\n", nextHop, dist_predicted);

if ((dist_predicted < Range_predicted) || (Range_predicted == dist_predicted))
                                        {
nextHop = pos[i][1];
break;
                                        }
                                }
else
                                {
printf("      > next Hop has been seen %d %f\n", nextHop,
Cluster_Member[node_addr][nextHop].timer);

future_latitude = Cluster_Member[node_addr][nextHop].latitude;
future_longitude = Cluster_Member[node_addr][nextHop].longitude;

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("      > next Hop has distance %d %f\n", nextHop, dist_predicted);

if ((dist_predicted < Range_predicted) || (Range_predicted == dist_predicted))
                                        {
nextHop = pos[i][1];
break;
                                        }
                                }
                        }
else
                        {

                        }
                }
        }

printf("      > The nearest Gateway to the Next_Cluster {{%d}} is {{%d}}
\n",nextCluster, nextHop);

// send reply packet
LORA-CBF_pk_send_to_mac_layer(data_pk_ptr,nextHop);
        }
```

```
////////////////////////////////////////////////////////////////////////////
/********** NextCluster Source ******************/
////////////////////////////////////////////////////////////////////////////

int nextCluster_source_from_ack_data(Packet* ack_data_pk_ptr, double latitude,
double longitude, double source_latitude, double source_longitude, int source)
        {
int             max, j,i, nextCluster;
double          latitude_neighbour, longitude_neighbour;
double          dist_neighbour_source;
double          pos[N][2];

max = j = 0;

for (i=0; i<N; i++)
                {
if ((Cluster_Neighbour[node_addr][i].status == IN_SERVICE) && ((op_sim_time () -
Cluster_Neighbour[node_addr][i].timer) < EXPIRATION_TIME))
                        {
latitude_neighbour = Cluster_Neighbour[node_addr][i].latitude;
longitude_neighbour = Cluster_Neighbour[node_addr][i].longitude;

printf("     > Cluster_neighbour %d\n", i);

dist_neighbour_source = sqrt((longitude_neighbour-
source_longitude)*(longitude_neighbour-source_longitude) + (latitude_neighbour-
source_latitude)*(latitude_neighbour-source_latitude));

//dist_node_source = sqrt((longitude-source_longitude)*(longitude-source_longitude) +
(latitude-source_latitude)*(latitude-source_latitude));

pos[j][0]= dist_neighbour_source;
pos[j][1] = (double)i;
j++;

max = j;
                        }
                }
if (max == 0)
                {
printf ("     > There is not another Cluster_head to transmit the packet !!!  \n");
                }

else if (max >= 1)
                {
bubbleSort(pos, max);
nextCluster = pos[0][1];
```

```
printf("     > The nearest Cluster_head to the Source {{%d}} is {{%d}} \n",source,
nextCluster);

op_pk_nfd_set (ack_data_pk_ptr, "Next_Cluster",nextCluster);
                }
return nextCluster;
        }

 ////////////////////////////////////////////////////////////////////////////////
 /****** Next Member Source from Cluster ***********/
////////////////////////////////////////////////////////////////////////////////

void Next_Member_source_from_cluster_ack_data(Packet* ack_data_pk_ptr, double
latitude, double longitude, double source_latitude, double source_longitude, int
nextCluster)
{
int             max, i, j, nextHop, source;
double          latitude_gateway, longitude_gateway, factor;
double          ist_node_source, dist_next_gateway_source, dist_node_next_gateway;
double interval_time, interval_latitude, interval_longitude;
double          future_latitude, future_longitude, dist_predicted;
double pos[N][2];

max = j = 0;

for (i=0; i<N; i++)
        {
if ((Cluster_Member[node_addr][i].status == IN_SERVICE) && ((op_sim_time () -
Cluster_Member[node_addr][i].timer) < EXPIRATION_TIME))
                {
latitude_gateway = Cluster_Member[node_addr][i].latitude;
ongitude_gateway = Cluster_Member[node_addr][i].longitude;

printf("     > Member %d \n", i);

//dist_node_source = sqrt((longitude-source_longitude)*(longitude-source_longitude) +
(latitude-source_latitude)*(latitude-source_latitude));

dist_next_gateway_source = sqrt((longitude_gateway-
source_longitude)*(longitude_gateway-source_longitude) + (latitude_gateway-
source_latitude)*(latitude_gateway-source_latitude));

//dist_node_next_gateway = sqrt((longitude-longitude_gateway)*(longitude-
longitude_gateway) + (latitude-latitude_gateway)*(latitude-latitude_gateway));

pos[j][0]= dist_next_gateway_source;
pos[j][1] = (double)i;
j++;
```

```
max = j;
                }
        }
if (max == 0)

        {
        }

else if (max >= 1)
        {
bubbleSort(pos, max);

printf("     > Predicting the position of the next Hop \n");

for (i=0; i<j; i++)
                {
nextHop = pos[i][1];

printf("     > nextHop =%d\n",nextHop);

if ((Cluster_Member[node_addr][nextHop].previa_latitude != 0.0) && (nextCluster !=
nextHop))
                        {
printf("     > next Hop has previa latitude %d %f\n",nextHop,
Cluster_Member[node_addr][nextHop].previa_latitude);

interval_time = Cluster_Member[node_addr][nextHop].timer -
Cluster_Member[node_addr][nextHop].previo_timer;

interval_latitude = (Cluster_Member[node_addr][nextHop].latitude -
Cluster_Member[node_addr][nextHop].previa_latitude)/interval_time;

interval_longitude = (Cluster_Member[node_addr][nextHop].longitude -
Cluster_Member[node_addr][nextHop].previa_longitude)/interval_time;

if ((op_sim_time () - Cluster_Member[node_addr][nextHop].timer) >
MINIMUM_DELAY)
                                {
printf("     > next Hop has been seen %d %f\n",nextHop,
Cluster_Member[node_addr][nextHop].timer);

factor = factor_Cluster_Member(nextHop);

future_latitude = (Cluster_Member[node_addr][nextHop].latitude + (interval_latitude *
factor));

future_longitude = (Cluster_Member[node_addr][nextHop].longitude +
(interval_longitude * factor));
```

```
dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("     > next Hop has distance %d %f\n",nextHop, dist_predicted);

if ((dist_predicted < Range_predicted) || (Range_predicted == dist_predicted))
                                        {
nextHop = pos[i][1];
break;
                                        }
                                }

else
                                {
printf("     > next Hop has been seen %d %f\n",nextHop,
Cluster_Member[node_addr][nextHop].timer);

future_latitude = Cluster_Member[node_addr][nextHop].latitude;
future_longitude = Cluster_Member[node_addr][nextHop].longitude;

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("     > next Hop has distance %d %f\n",nextHop, dist_predicted);

if ((dist_predicted < Range_predicted) || (Range_predicted == dist_predicted))
                                        {
nextHop = pos[i][1];
break;
                                        }
                                }
                        }

else
                        {

                        }
                }
        }
printf("     > The nearest Gateway to the Next_Cluster {{%d}} is {{%d}}
\n",nextCluster, nextHop);
// send reply packet
LORA-CBF_pk_send_to_mac_layer(ack_data_pk_ptr,nextHop);
}
```

```
////////////////////////////////////////////////////////////////
/****************** Next Member Source *****************/
////////////////////////////////////////////////////////////////

void Next_Member_source_ack_data(Packet* ack_data_pk_ptr, double latitude, double
longitude, double source_latitude, double source_longitude, int nextCluster, int source)

{
int             max, i, j, nextHop;
double          latitude_gateway, longitude_gateway, factor;
double          dist_node_source, dist_next_gateway_source,
                dist_node_next_gateway;
double          interval_time, interval_latitude, interval_longitude;
double          future_latitude, future_longitude, dist_predicted;
double          pos[N][2];

max = j = 0;

for (i=0; i<N; i++)

        {
if ((Member_Neighbour[node_addr][i].status == IN_SERVICE) && ((op_sim_time () -
Member_Neighbour[node_addr][i].timer) < EXPIRATION_TIME))
                {
latitude_gateway = Member_Neighbour[node_addr][i].latitude;
longitude_gateway = Member_Neighbour[node_addr][i].longitude;

printf("      > Member %d \n", i);

//dist_node_source = sqrt((longitude-source_longitude)*(longitude-source_longitude) +
(latitude-source_latitude)*(latitude-source_latitude));

dist_next_gateway_source = sqrt((longitude_gateway-
source_longitude)*(longitude_gateway-source_longitude) + (latitude_gateway-
source_latitude)*(latitude_gateway-source_latitude));

//dist_node_next_gateway = sqrt((longitude-longitude_gateway)*(longitude-
longitude_gateway) + (latitude-latitude_gateway)*(latitude-latitude_gateway));

pos[j][0]= dist_next_gateway_source;
pos[j][1] = (double)i;
j++;
max = j;
                }
        }
if (max == 0)
        {
        }
```

```
else if (max >= 1)
        {
bubbleSort(pos, max);

printf("     > Predicting the position of the next Hop \n");

for (i=0; i<j; i++)
                {
nextHop = pos[i][1];
printf("     > next Hop %d\n",nextHop);

if ((Member_Neighbour[node_addr][nextHop].previa_latitude != 0.0) && (nextCluster
!= nextHop))
                        {
printf("     > next Hop has previa latitude %d %f\n",nextHop,
Member_Neighbour[node_addr][nextHop].previa_latitude);

interval_time = Member_Neighbour[node_addr][nextHop].timer -
Member_Neighbour[node_addr][nextHop].previo_timer;

interval_latitude = (Member_Neighbour[node_addr][nextHop].latitude -
Member_Neighbour[node_addr][nextHop].previa_latitude)/interval_time;

interval_longitude = (Member_Neighbour[node_addr][nextHop].longitude -
Member_Neighbour[node_addr][nextHop].previa_longitude)/interval_time;

if ((op_sim_time () - Member_Neighbour[node_addr][nextHop].timer) >
MINIMUM_DELAY)
                                {
printf("     > next Hop has been seen %d %f\n", nextHop,
Member_Neighbour[node_addr][nextHop].timer);

factor = factor_Member_Neighbour(nextHop);

future_latitude = (Member_Neighbour[node_addr][nextHop].latitude +
(interval_latitude * factor));

future_longitude = (Member_Neighbour[node_addr][nextHop].longitude +
(interval_longitude * factor));

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("     > next Hop has distance %d %f\n", nextHop, dist_predicted);

if ((dist_predicted < Range_predicted) || (Range_predicted == dist_predicted))
                                        {
nextHop = pos[i][1];
```

```
break;
                                        }
                                    }

else
                                    {
printf("     > next Hop has been seen %d %f\n", nextHop,
Member_Neighbour[node_addr][nextHop].timer);

future_latitude = Member_Neighbour[node_addr][nextHop].latitude;
future_longitude = Member_Neighbour[node_addr][nextHop].longitude;

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("     > next Hop has distance %d %f\n", nextHop, dist_predicted);

if ((dist_predicted < Range_predicted) || (Range_predicted == dist_predicted))
                                        {
nextHop = pos[i][1];
break;
                                        }
                                    }
                                }

else

                                {

                                }
                            }
        }
printf("     > The nearest Gateway to the Next_Cluster {{%d}} is {{%d}}
\n",nextCluster, nextHop);

// send reply packet
LORA-CBF_pk_send_to_mac_layer(ack_data_pk_ptr,nextHop);
        }
```

```
////////////////////////////////////////////////////////////////////////////////
/******* Predicting the position of the destination *****/
////////////////////////////////////////////////////////////////////////////////

Boolean predicting_position_destination(int dest, double latitude, double longitude)
        {
double          interval_time, interval_latitude, interval_longitude, factor;
double          future_latitude, future_longitude, dist_predicted;
Boolean         result = OPC_FALSE;

if ((Clusterhead[node_addr].Member_id[dest] == dest) && ((op_sim_time () -
Cluster_Member[node_addr][dest].timer) < EXPIRATION_TIME))
                {
if (Cluster_Member[node_addr][dest].previa_latitude != 0.0)
                        {
printf("      > Destination has previa latitude %d %f\n", dest,
Cluster_Member[node_addr][dest].previa_latitude);

interval_time = Cluster_Member[node_addr][dest].timer -
Cluster_Member[node_addr][dest].previo_timer;

interval_latitude = (Cluster_Member[node_addr][dest].latitude -
Cluster_Member[node_addr][dest].previa_latitude)/interval_time;

interval_longitude = (Cluster_Member[node_addr][dest].longitude -
Cluster_Member[node_addr][dest].previa_longitude)/interval_time;

f ((op_sim_time () - Cluster_Member[node_addr][dest].timer) > MINIMUM_DELAY)
                                {
printf("  > Destination has been seen %d %f\n", dest,
Cluster_Member[node_addr][dest].timer);

factor = factor_Cluster_Member(dest);

future_latitude = (Cluster_Member[node_addr][dest].latitude + (interval_latitude *
factor));
future_longitude = (Cluster_Member[node_addr][dest].longitude + (interval_longitude
* factor));

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("      > Destination has distance %d %f\n",dest, dist_predicted);
                                }
else
                                {
printf("      > Destination has been seen %d %f\n", dest,
Cluster_Member[node_addr][dest].timer);
```

```
future_latitude = Cluster_Member[node_addr][dest].latitude;
future_longitude = Cluster_Member[node_addr][dest].longitude;

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
 (longitude - future_longitude) * (longitude - future_longitude));

printf("      > Destination has distance %d %f\n",dest, dist_predicted);
                                        }
                                }

else
                                {
printf("     > Destination has previa latitude %d %f\n", dest,
Cluster_Member[node_addr][dest].previa_latitude);

future_latitude = Cluster_Member[node_addr][dest].latitude;
future_longitude = Cluster_Member[node_addr][dest].longitude;

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("      > Destination has distance %d %f\n",dest, dist_predicted);
                                }
                        }
if ((dist_predicted < Range_predicted) || (Range_predicted == dist_predicted))
                                {
result = OPC_TRUE;
                                }
return result;
        }
```

```
////////////////////////////////////////////////////////////////////////////
/********* Next Member Destination ********************/
////////////////////////////////////////////////////////////////////////////

void Next_Member_dest_data(Packet* data_pk_ptr, double latitude, double longitude,
double dest_latitude, double dest_longitude, int nextCluster, int dest)
{
int             max, i, j, nextHop;
double          latitude_gateway, longitude_gateway, factor;
double          dist_node_dest, dist_next_gateway_dest, dist_node_next_gateway;
double          interval_time, interval_latitude, interval_longitude;
double          future_latitude, future_longitude, dist_predicted;
double pos[N][2];

max = j = 0;

for (i=0; i<N; i++)

        {
if ((Member_Neighbour[node_addr][i].status == IN_SERVICE) && ((op_sim_time () -
Member_Neighbour[node_addr][i].timer) < EXPIRATION_TIME))
                {
latitude_gateway = Member_Neighbour[node_addr][i].latitude;
longitude_gateway = Member_Neighbour[node_addr][i].longitude;

printf("     > Member %d \n", i);

//dist_node_dest = sqrt((longitude-dest_longitude)*(longitude-dest_longitude) +
(latitude-dest_latitude)*(latitude-dest_latitude));

dist_next_gateway_dest = sqrt((longitude_gateway-
dest_longitude)*(longitude_gateway-dest_longitude) + (latitude_gateway-
dest_latitude)*(latitude_gateway-dest_latitude));

//dist_node_next_gateway = sqrt((longitude-longitude_gateway)*(longitude-
longitude_gateway) + (latitude-latitude_gateway)*(latitude-latitude_gateway));

pos[j][0]= dist_next_gateway_dest;
pos[j][1] = (double)i;
j++;
max = j;
                }
        }
if (max == 0)

        {
        }
```

```
else if (max >= 1)
        {
bubbleSort(pos, max);

printf("     > Predicting the position of the next Hop \n");

for (i=0; i<j; i++)
                {
nextHop = pos[i][1];
printf("     > nextHop =%d\n",nextHop);

if ((Member_Neighbour[node_addr][nextHop].previa_latitude != 0.0) && (nextCluster
!= nextHop))
                        {
printf("     > Next Hop has previa latitude %d %f\n", nextHop,
Member_Neighbour[node_addr][nextHop].previa_latitude);

interval_time = Member_Neighbour[node_addr][nextHop].timer -
Member_Neighbour[node_addr][nextHop].previo_timer;

interval_latitude = (Member_Neighbour[node_addr][nextHop].latitude -
Member_Neighbour[node_addr][nextHop].previa_latitude)/interval_time;

interval_longitude = (Member_Neighbour[node_addr][nextHop].longitude -
Member_Neighbour[node_addr][nextHop].previa_longitude)/interval_time;

if ((op_sim_time () - Member_Neighbour[node_addr][nextHop].timer) >
MINIMUM_DELAY)
                                {
printf("     > Next Hop has been seen %d %f\n", nextHop,
Member_Neighbour[node_addr][nextHop].timer);

factor = factor_Member_Neighbour(nextHop);
future_latitude = (Member_Neighbour[node_addr][nextHop].latitude +
(interval_latitude * factor));

future_longitude = (Member_Neighbour[node_addr][nextHop].longitude +
(interval_longitude * factor));

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("     > next Hop has distance %d %f\n",nextHop, dist_predicted);

if ((dist_predicted < Range_predicted) || (Range_predicted == dist_predicted))
                                        {
nextHop = pos[i][1];
break;
```

```
                                        }
                                    }
else

                                    {
printf("     > Next Hop has been seen %d %f\n", nextHop,
Member_Neighbour[node_addr][nextHop].timer);

future_latitude = Member_Neighbour[node_addr][nextHop].latitude;
future_longitude = Member_Neighbour[node_addr][nextHop].longitude;

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("     > next Hop has distance %d %f\n",nextHop, dist_predicted);

if ((dist_predicted < Range_predicted) || (Range_predicted == dist_predicted))
                                        {
nextHop = pos[i][1];
break;
                                        }
                                    }
                                }

else

                                {
                                }
                            }
                        }

 printf("     > The nearest Gateway to the Next_Cluster {{%d}} is {{%d}}
\n",nextCluster, nextHop);

 // send reply packet
 LORA-CBF_pk_send_to_mac_layer(data_pk_ptr,nextHop);
 }
```

```
////////////////////////////////////////////////////////////////////////////////
/*******  Predicting the position of the source  ******/
////////////////////////////////////////////////////////////////////////////////

Boolean predicting_position_source(int source, double latitude, double longitude)

        {
double          interval_time, interval_latitude, interval_longitude, factor;
double          future_latitude, future_longitude, dist_predicted;
Boolean         result = OPC_FALSE;

if ((Clusterhead[node_addr].Member_id[source] == source) && ((op_sim_time () -
Cluster_Member[node_addr][source].timer) < EXPIRATION_TIME))
                {
if (Cluster_Member[node_addr][source].previa_latitude != 0.0)
                        {
printf("      > Source has previa latitude %d %f\n", source,
Cluster_Member[node_addr][source].previa_latitude);

interval_time = Cluster_Member[node_addr][source].timer -
Cluster_Member[node_addr][source].previo_timer;

interval_latitude = (Cluster_Member[node_addr][source].latitude -
Cluster_Member[node_addr][source].previa_latitude)/interval_time;

interval_longitude = (Cluster_Member[node_addr][source].longitude -
Cluster_Member[node_addr][source].previa_longitude)/interval_time;

if ((op_sim_time () - Cluster_Member[node_addr][source].timer) >
MINIMUM_DELAY)
                                {
printf("      > Source has been seen %d %f\n", source,
Cluster_Member[node_addr][source].timer);

factor = factor_source(source);

future_latitude = (Cluster_Member[node_addr][source].latitude + (interval_latitude *
factor));
future_longitude = (Cluster_Member[node_addr][source].longitude +
(interval_longitude * factor));

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("      > Source has distance %d %f\n", source, dist_predicted);
                                }
                        else
                                {
```

```
printf("      > Source has been seen %d %f\n", source,
Cluster_Member[node_addr][source].timer);

future_latitude = Cluster_Member[node_addr][source].latitude;

future_longitude = Cluster_Member[node_addr][source].longitude;

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("      > Source has distance %d %f\n", source, dist_predicted);
                                        }
                                }
                        else
                                {
printf("      > Source has previa latitude %d %f\n", source,
Cluster_Member[node_addr][source].previa_latitude);

future_latitude = Cluster_Member[node_addr][source].latitude;
future_longitude = Cluster_Member[node_addr][source].longitude;

dist_predicted = sqrt((latitude - future_latitude) * (latitude - future_latitude) +
(longitude - future_longitude) * (longitude - future_longitude));

printf("      > Source has distance %d %f\n", source, dist_predicted);
                                }
                        }
if ((dist_predicted < Range_predicted) || (Range_predicted == dist_predicted))
                                {
result = OPC_TRUE;
                                }

        return result;

        }
```

```
////////////////////////////////////////////////////////////////
/****** Is ACK DATA Packet fresh enough *********/
////////////////////////////////////////////////////////////////

Boolean LORA-CBF_ack_data_pk_is_fresh_enough(Packet* ack_data_pk_ptr)
        {
// var
Boolean     result = OPC_FALSE;
int         source, dest, seq_number;
/*
/* Check whether the BroadcastID of the request
/* is greater than the last seen. If so, RREP is
/* automatically processed. Else, node checks
/* whether RREP packet was seen less than
/* RECORD_BROADCAST_TIME seconds ago. If so, RREP
/* must be discarded. Else, it is processed.
*/

// Read RREQ fields
op_pk_nfd_get (ack_data_pk_ptr,"Seq_number",&seq_number);
op_pk_nfd_get (ack_data_pk_ptr,"SRC",&source);
op_pk_nfd_get (ack_data_pk_ptr,"DEST",&dest);

if((Ackseen[dest][source].sequence_number < seq_number) ||
((Ackseen[dest][source].sequence_number==seq_number) && (op_sim_time()>
Ackseen[dest][source].expirationTime)))
                {
result = OPC_TRUE;
// update RequestSeen repository
Ackseen[dest][source].sequence_number=seq_number;

Ackseen[dest][source].expirationTime=op_sim_time()+BROADCAST_RECORD_TIM
E;
                }
return result;
        }
```

```
////////////////////////////////////////////////////////////////////////////
/************ Factor Member Neighbour ***************/
////////////////////////////////////////////////////////////////////////////

double factor_Member_Neighbour(int nextHop)
        {
double diference, convert, factor;

diference = (op_sim_time () - Member_Neighbour[node_addr][nextHop].timer);

convert = diference - (floor(diference));
if (convert <= 0.5)
        {
factor = floor(op_sim_time () - Member_Neighbour[node_addr][nextHop].timer);

if (factor == 0.0)
                {
factor = 1.0;
                }
        }
else
        {
factor = ceil(op_sim_time () - Member_Neighbour[node_addr][nextHop].timer);
        }
printf("     > factor = %f\n",factor);
return (factor);
}

////////////////////////////////////////////////////////////////////////////
/******** Factor Cluster Neighbour**********************/
////////////////////////////////////////////////////////////////////////////

double factor_Cluster_Neighbour(int nextCluster)
        {
double diference, convert, factor;

diference = (op_sim_time () - Cluster_Neighbour[node_addr][nextCluster].timer);

convert = diference - (floor(diference));

if (convert <= 0.5)
        {
factor = floor(op_sim_time () - Cluster_Neighbour[node_addr][nextCluster].timer);

if (factor == 0.0)
                {
factor = 1.0;
                }
```

```
        }
else
        {
factor = ceil(op_sim_time () - Cluster_Neighbour[node_addr][nextCluster].timer);
        }

printf("     > factor = %f\n",factor);

return (factor);
}

 ////////////////////////////////////////////////////////////////////////////////
 /****** Factor Cluster Member DESTINATION***************/
////////////////////////////////////////////////////////////////////////////////

double factor_Cluster_Member(int dest)
        {
double diference, convert, factor;

diference = (op_sim_time () - Cluster_Member[node_addr][dest].timer);

convert = diference - (floor(diference));

if (convert <= 0.5)
        {
factor = floor(op_sim_time () - Cluster_Member[node_addr][dest].timer);

if (factor == 0.0)
                {
factor = 1.0;
                }
        }
else
        {
factor = ceil(op_sim_time () - Cluster_Member[node_addr][dest].timer);
        }
printf("     > factor = %f\n",factor);

return (factor);

}
```

```
////////////////////////////////////////////////////////////////////////////
/******* Factor Cluster Member SOURCE************/
////////////////////////////////////////////////////////////////////////////

double factor_source(int source)
        {
double diference, convert, factor;

diference = (op_sim_time () - Cluster_Member[node_addr][source].timer);

convert = diference - (floor(diference));

if (convert <= 0.5)
        {
factor = floor(op_sim_time () - Cluster_Member[node_addr][source].timer);

if (factor == 0.0)
                {
factor = 1.0;
                }
        }
else
        {
factor = ceil(op_sim_time () - Cluster_Member[node_addr][source].timer);
        }
printf("     > factor = %f\n",factor);
return (factor);
}

/* End of Function Block */

/* Undefine optional tracing in FIN/FOUT/FRET */
/* The FSM has its own tracing code and the other */
/* functions should not have any tracing.                */
#undef FIN_TRACING
#define FIN_TRACING

#undef FOUTRET_TRACING
#define FOUTRET_TRACING

#if defined (__cplusplus)
extern "C" {
#endif
void LORA-CBF_clusters_and_members_only (void);
Compcode LORA-CBF_clusters_and_members_only_init (void **);
void LORA-CBF_clusters_and_members_only_diag (void);
void LORA-CBF_clusters_and_members_only_terminate (void);
void LORA-CBF_clusters_and_members_only_svar (void *, const char *, char **);
```

```
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif

/* Process model interrupt handling procedure */

void LORA-CBF_clusters_and_members_only (void)
        {
int _block_origin = 0;
FIN (LORA-CBF_clusters_and_members_only ());
if (1)
                {
double                  first_hello_intrvl;
double                  altitude,latitude,longitude,x_pos,y_pos,z_pos;
double                  dest_latitude, dest_longitude;
double                  source_latitude, source_longitude;
Compcode                comp_code;
Packet*                 pkptr;
int                     type, i, j, nextHop, previousHop, source,
                        Cluster_Neighbor, dest, max;
Ici*                    buffer;
int                     destination,cluster, direction;
int                     Members, num_members;

FSM_ENTER (LORA-CBF_clusters_and_members_only)

FSM_BLOCK_SWITCH
                        {
/*---------------------------------------------------------*/
/** state (Init) enter executives **/
FSM_STATE_ENTER_FORCED_NOLABEL (0, "Init", "LORA-
CBF_clusters_and_members_only [Init enter execs]")

FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only [Init enter
execs]", state0_enter_exec)
        {
/* Initialization of the process model.                    */
/* All the attributes are loaded in this routine and       */
/* a self interruption is scheduled to initiate the        */
/* the first Hello Interval                                */

node_id = op_topo_parent (op_id_self());
op_ima_obj_attr_get(node_id,"Wireless LAN MAC Address",&node_addr);

op_ima_obj_attr_get(op_id_self(),"ACTIVE_ROUTE_TIMEOUT",&ACTIVE_ROUT
E_TIMEOUT);
op_ima_obj_attr_get(op_id_self(),"HELLO_MODE",      &HELLO_MODE);
op_ima_obj_attr_get(op_id_self(),"TTL",             &TTL_START);
```

```
op_ima_obj_attr_get(op_id_self(),"TR",          &TR);

/* Constants */

HELLO_INTERVAL                  = 2.0;
CLUSTER_INTERVAL                = 1.0;
EXPIRATION_TIME                 = 3.0;
CONVERT_CLUSTER                 = 2.0;
BROADCAST_RECORD_TIME           = 3.0;
mySeqNb                         = 0;
ackNb                           = 0;
TIMER_RENEW                     = 1;
Data_seq_number                 = 1;
//Range_predicted               = 0.001346;               //150m
//Range_predicted               = 0.00179;                //200m
Range_predicted                 = 0.0026962602390916;     //300m
//Range_predicted               = 0.003143;               //350m
Packet_Retransmit               = 0;
pk_received                     = 0;
pk_transmited                   = 0;
tot_delay                       = 0.0;
average_delay                   = 0.0;
delay                           = 0.0;
latency                         = 0.0;
MINIMUM_TIME                    = 0.8;
MINIMUM_DELAY                   = 0.2;

nodeid[node_addr].transmission  = INACTIVE;
nodeid[node_addr].latitude      = 0.0;
nodeid[node_addr].longitude     = 0.0;
nodeid[node_addr].direction     = 0;

transmit_dist = op_dist_load("uniform_double", 0.0, 0.5);

/* Set priorities*/
op_intrpt_priority_set(OPC_INTRPT_STRM,RCV_STRM,20);
op_intrpt_priority_set(OPC_INTRPT_STRM,SRC_STRM,5);

for (i=0; i<N; i++)
{
RequestSent[i].sequence_number=0;
RequestSent[i].status=OFF;
RequestSent[i].ttl=TTL_START;
for (j=0; j<N; j++)
{
RequestSeen[i][j].expirationTime = -1;
RequestSeen[i][j].broadcastID = -1;
ReplySeen[i][j].expirationTime = -1;
```

```
ReplySeen[i][j].sequence_number = -1;
Dataseen[i][j].sequence_number = -1;
Ackseen[i][j].sequence_number =-1;
Register_Cluster[i][j][i].timer = 0.0;
}
}

for (i=0; i<N; i++)
{
Location[i].latitude = 0.0;
Location[i].longitude = 0.0;
Cluster_Neighbour[node_addr][i].latitude = 0.0;
Cluster_Neighbour[node_addr][i].longitude = 0.0;
Cluster_Neighbour[node_addr][i].timer = 0.0;
Cluster_Member[node_addr][i].latitude = 0.0;
Cluster_Member[node_addr][i].longitude = 0.0;
Member_Neighbour[node_addr][i].timer = 0.0;
Cluster_Neighbour[node_addr][i].timer = 0.0;
}

for (i=0; i<N; i++)

{
for (j=0; j<N; j++)
{
Retransmission[i][j].counter = 0;
Clusterhead[i].Member_id[j] = 0;
Clusterhead[i].Gw[j] = 0;
Cluster_Neighbour[i][j].status = DOWN;
        }
}

retransmissions_hndl = op_stat_reg ("Retransmissions", OPC_STAT_INDEX_NONE,
OPC_STAT_GLOBAL);
latency_hndl = op_stat_reg ("Latency", OPC_STAT_INDEX_NONE,
OPC_STAT_GLOBAL);
average_delay_hndl = op_stat_reg ("Average_delay", OPC_STAT_INDEX_NONE,
OPC_STAT_GLOBAL);
pk_received_hndl = op_stat_reg ("Pk_received", OPC_STAT_INDEX_NONE,
OPC_STAT_GLOBAL);
pk_transmited_hndl = op_stat_reg ("Pk_transmited", OPC_STAT_INDEX_NONE,
OPC_STAT_GLOBAL);
eficiency_hndl = op_stat_reg ("Eficiency", OPC_STAT_INDEX_NONE,
OPC_STAT_GLOBAL);
buffer_packets_hndl = op_stat_reg ("Buffer_packets", OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
```

```
// Format the Hello Module and schedule the first broadcast of hello msg
if(HELLO_MODE)
        {
/*init the hello_module */
hello_module.hello_msg_template = op_pk_create_fmt("HELLO_PACKET");

/* trigger first intrpt for hello broadcast */
hello_dist = op_dist_load ("uniform_double", 0.0,1.0);
first_hello_intrvl = op_dist_outcome(hello_dist);
hello_module.evt=
op_intrpt_schedule_self(op_sim_time()+first_hello_intrvl,UNDECIDED);
                        }
                                }
FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only [Init
enter execs]", state0_enter_exec)

/** state (Init) exit executives **/
FSM_STATE_EXIT_FORCED (0, "Init", "LORA-CBF_clusters_and_members_only
[Init exit execs]")

FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only [Init exit
execs]", state0_exit_exec)
                                {
                                }
FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only [Init
exit execs]", state0_exit_exec)


/** state (Init) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "Init", "idle")
/*---------------------------------------------------------*/

/** state (idle) enter executives **/
FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "idle", "LORA-
CBF_clusters_and_members_only [idle enter execs]")

FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only [idle enter
execs]", state1_enter_exec)
                                {
                                }

FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only [idle
enter execs]", state1_enter_exec)

/** blocking after enter executives of unforced state. **/
FSM_EXIT (3,LORA-CBF_clusters_and_members_only)
```

```
/** state (idle) exit executives **/
FSM_STATE_EXIT_UNFORCED (1, "idle", "LORA-
CBF_clusters_and_members_only [idle exit execs]")

FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only [idle exit
execs]", state1_exit_exec)
                                {
                                }
FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only [idle
exit execs]", state1_exit_exec)

/** state (idle) transition processing **/
FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only [idle trans
conditions]", state1_trans_conds)

FSM_INIT_COND (Type_Undecided)
FSM_TEST_COND (Type_Cluster_head)
FSM_TEST_COND (LOWER_LAYER_ARVL)
FSM_TEST_COND (UPPER_LAYER_ARVL)
FSM_TEST_COND (Notice_To_Serve_Buffer)
FSM_DFLT_COND
FSM_TEST_LOGIC ("idle")
FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only [idle
trans conditions]", state1_trans_conds)

FSM_TRANSIT_SWITCH
                                {
FSM_CASE_TRANSIT (0, 2, state2_enter_exec, ;, "Type_Undecided", "", "idle",
"Undecided")
FSM_CASE_TRANSIT (1, 3, state3_enter_exec, ;, "Type_Cluster_head", "", "idle",
"Cluster_head")
FSM_CASE_TRANSIT (2, 4, state4_enter_exec, ;, "LOWER_LAYER_ARVL", "",
"idle", "Receiving")
FSM_CASE_TRANSIT (3, 5, state5_enter_exec, ;, "UPPER_LAYER_ARVL", "",
"idle", "Transmiting")
FSM_CASE_TRANSIT (4, 6, state6_enter_exec, ;, "Notice_To_Serve_Buffer", "",
"idle", "Buffer")
FSM_CASE_TRANSIT (5, 1, state1_enter_exec, ;, "default", "", "idle", "idle")
                                }
/*---------------------------------------------------------*/

/** state (Undecided) enter executives **/
FSM_STATE_ENTER_FORCED (2, state2_enter_exec, "Undecided", "LORA-
CBF_clusters_and_members_only [Undecided enter execs]")

FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only
[Undecided enter execs]", state2_enter_exec)
                                {
```

```
/* Broadcast a Hello Message and schedule a self */
/* interrupt to the next hello interval          */

if (HELLO_MODE)
                                        {
printf("\n      {{node %d}} Node Undecided send a Hello msg :: ", node_addr);

node_id = op_topo_parent (op_id_self());
comp_code = op_ima_obj_pos_get (node_id, &latitude, &longitude, &altitude,
&x_pos, &y_pos, &z_pos);

/* This line tests the completion code, and upon detecting failure, calls op_sim_end () to
*/
/* immediately end the simulation and print and error message to the standard output
device,*/
/* and the opnet message area.                                          */

if (comp_code == OPC_COMPCODE_FAILURE)
op_sim_end ("get attributes failed", "", "","");

nodeid[node_addr].previa_latitude = nodeid[node_addr].latitude;
nodeid[node_addr].previa_longitude = nodeid[node_addr].longitude;
nodeid[node_addr].latitude = latitude;
nodeid[node_addr].longitude = longitude;
nodeid[node_addr].timer = op_sim_time ();

/* Assign source and dest to the Hello packet */

op_pk_nfd_set(hello_module.hello_msg_template,"SRC",node_addr);

op_pk_nfd_set(hello_module.hello_msg_template,"DEST",node_addr);

op_pk_nfd_set(hello_module.hello_msg_template,"HopCount",0);

op_pk_nfd_set(hello_module.hello_msg_template,"Type_of_node",Undecided);

op_pk_nfd_set(hello_module.hello_msg_template,"Latitude",latitude);

op_pk_nfd_set(hello_module.hello_msg_template,"Longitude",longitude);

nodeid[node_addr].status_of_node = Undecided;

/* Send Packet */

LORA-CBF_pk_send_to_mac_layer(op_pk_copy(hello_module.hello_msg_template),
BROADCAST);

/* Update hello interval */
```

```
hello_module.evt = op_intrpt_schedule_self(op_sim_time () + HELLO_INTERVAL,
CLUSTER_HEAD);

printf("     - Next hello intrpt scheduled \n");
                                        }
                                }

FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only
[Undecided enter execs]", state2_enter_exec)

/** state (Undecided) exit executives **/
FSM_STATE_EXIT_FORCED (2, "Undecided", "LORA-
CBF_clusters_and_members_only [Undecided exit execs]")

FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only
[Undecided exit execs]", state2_exit_exec)
                                {
                                }
FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only
[Undecided exit execs]", state2_exit_exec)


/** state (Undecided) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "Undecided", "idle")
/*---------------------------------------------------------*/


/** state (Cluster_head) enter executives **/
FSM_STATE_ENTER_FORCED (3, state3_enter_exec, "Cluster_head", "LORA-
CBF_clusters_and_members_only [Cluster_head enter execs]")

FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only
[Cluster_head enter execs]", state3_enter_exec)
                                {
/* Broadcast a Hello Message and schedule a self */
/* interrupt to the next hello interval         */

if (HELLO_MODE)
                                        {
printf("\n     {{node %d}} Node Cluster_head send a Hello msg :: \n", node_addr);

/* Update the Type of node */

nodeid[node_addr].status_of_node = Cluster_head;

node_id = op_topo_parent (op_id_self());
```

```
comp_code = op_ima_obj_pos_get (node_id, &latitude, &longitude, &altitude,
&x_pos, &y_pos, &z_pos);

/* This line tests the completion code, and upon detecting failure, calls op_sim_end () to
*/
/* immediately end the simulation and print and error message to the standard output
device,*/
/* and the opnet message area.
*/

if (comp_code == OPC_COMPCODE_FAILURE)
op_sim_end ("get attributes failed", "", "","");

if ((nodeid[node_addr].transmission == ACTIVE && (op_sim_time () -
nodeid[node_addr].active_timer <= EXPIRATION_TIME)))
                                                {
op_pk_nfd_set (hello_module.hello_msg_template,"Transmission",ACTIVE);
                                                }

else
                                                {
op_pk_nfd_set (hello_module.hello_msg_template,"Transmission",INACTIVE);
                                                }

Members = 0;

for (i=0; i<N; i++)

                                                {
if ((Cluster_Member[node_addr][i].status == IN_SERVICE) && ((op_sim_time () -
Cluster_Member[node_addr][i].timer) < CLUSTER_INTERVAL))
                                                                        {
Members++;
                                                                        }

else
                                                                        {

                                                                        }

                                                                }
op_pk_nfd_set (hello_module.hello_msg_template,"HopCount", 0);
op_pk_nfd_set (hello_module.hello_msg_template,"Num_Members", Members);
op_pk_nfd_set (hello_module.hello_msg_template,"Type_of_node",Cluster_head);
op_pk_nfd_set (hello_module.hello_msg_template,"Latitude",latitude);
op_pk_nfd_set (hello_module.hello_msg_template,"Longitude",longitude);
```

```
/* Send Packet */

LORA-CBF_pk_send_to_mac_layer (op_pk_copy (hello_module.hello_msg_template),
BROADCAST);

/* Update hello interval from Cluster */

nodeid[node_addr].evt_cluster = op_intrpt_schedule_self(op_sim_time () +
CLUSTER_INTERVAL, CLUSTER_HEAD);

printf("      - Next hello intrpt scheduled from Cluster_head %d \n", node_addr);
                                        }
                                }
FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only
[Cluster_head enter execs]", state3_enter_exec)

/** state (Cluster_head) exit executives **/
FSM_STATE_EXIT_FORCED (3, "Cluster_head", "LORA-
CBF_clusters_and_members_only [Cluster_head exit execs]")
FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only
[Cluster_head exit execs]", state3_exit_exec)
                                {
                                }
FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only
[Cluster_head exit execs]", state3_exit_exec)

/** state (Cluster_head) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "Cluster_head", "idle")
/*---------------------------------------------------------*/

/** state (Receiving) enter executives **/
FSM_STATE_ENTER_FORCED (4, state4_enter_exec, "Receiving", "LORA-
CBF_clusters_and_members_only [Receiving enter execs]")

FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only [Receiving
enter execs]", state4_enter_exec)
                                {
 /* This state receives the incoming packet stream from   */
/* the lower layer. It first checks the type of the      */
/* received packet then, calls the appropriate function  */
/* to proceed.                                           */
/* N.B. promiscuous listenning is blocked at this state  */

pkptr = op_pk_get(RCV_STRM);
if (LORA-CBF_pk_is_in_tr(pkptr) == OPC_TRUE)
                                {
op_pk_nfd_get(pkptr,"Type",&type);
op_pk_nfd_get(pkptr,"NextHop",&nextHop);
```

```
op_pk_nfd_get(pkptr,"PreviousHop",&previousHop);
op_pk_nfd_get(pkptr,"SRC",&source);

// check packet type
switch(type)
                                        {
case HELLO_PACKET_TYPE:  // HELLO packet received
                                                {
if (nextHop == BROADCAST)
                                                        {
printf("\n      {{ node %d }}", node_addr);
LORA-CBF_pk_print(pkptr);
LORA-CBF_hello_msg_receive(pkptr);
                                                        }

else
                                                        {
op_pk_destroy(pkptr);
                                                        }
break;
                                                }

case REQUEST_PACKET_TYPE: // RREQ received
                                                {
if(source != node_addr)
                                                        {
printf("\n      {{ node %d }}\n      - A REQUEST pk has been received from mac
layer\n", node_addr);

LORA-CBF_pk_print(pkptr);
LORA-CBF_rreq_pk_receive(pkptr);
                                                        }
else
                                                        {
// request has looped back to source: RREQ discarded.
op_pk_destroy(pkptr);
                                                        }
break;
                                                }
case ACK_RREQ: // Acknowledge from the rreq packet
                                                {
if (nextHop == node_addr)
                                                        {
```

```
// It is an acknowledge packet

printf(" \n     {{node %d}}\n     - An ack_rreq packet has been received \n",
node_addr);
LORA-CBF_pk_print(pkptr);
LORA-CBF_ack_rreq_pk_receive(pkptr);
                                                    }
else
                                                        {
op_pk_destroy(pkptr);
                                                        }
break;
                                                    }

case REPLY_PACKET_TYPE:  // RREP received
                                                    {

if(nextHop == node_addr)
                                                        {
printf("\n     {{ node %d }}\n     - A REPLY pk has been received from mac layer\n",
node_addr);

LORA-CBF_pk_print(pkptr);
LORA-CBF_rrep_pk_receive(pkptr);
                                                        }
else
                                                         {
// Reply has looped back to source: RREP discarded.
op_pk_destroy(pkptr);
                                                         }
break;
                                                    }

case DATA_PACKET_TYPE: // data received
                                                    {

if(nextHop != node_addr)
                                                        {
op_pk_destroy(pkptr);
                                                        }
else
                                                        {
printf("\n     {{ node %d }}\n     - A DATA pk has been received from mac layer\n",
node_addr);

LORA-CBF_pk_print(pkptr);
LORA-CBF_data_pk_receive(pkptr);
                                                        }
```

```
break;
                                                    }

case ACK_DATA:  //Acknowledge from the data packet

                                                    {

if (nextHop == node_addr)
                                                    {
// It is an acnowledge packet

printf (" \n     {{node %d}}\n     - An ack_data_pk has been received \n", node_addr);
LORA-CBF_pk_print(pkptr);
                                                    LORA-
CBF_ack_data_pk_receive(pkptr);
                                                    }

else

                                                    {
op_pk_destroy(pkptr);
                                                    }
break;
                                                }

                                                } /* end switch(type)*/

                                            }

else

                                            {
op_pk_destroy(pkptr);
                                            }
                                    }

FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only
[Receiving enter execs]", state4_enter_exec)

/** state (Receiving) exit executives **/
FSM_STATE_EXIT_FORCED (4, "Receiving", "LORA-
CBF_clusters_and_members_only [Receiving exit execs]")

FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only [Receiving
exit execs]", state4_exit_exec)
                                    {
                                    }
```

```
FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only
[Receiving exit execs]", state4_exit_exec)


/** state (Receiving) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "Receiving", "idle")
/*---------------------------------------------------------*/

/** state (Transmiting) enter executives **/
FSM_STATE_ENTER_FORCED (5, state5_enter_exec, "Transmiting", "LORA-
CBF_clusters_and_members_only [Transmiting enter execs]")

FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only
[Transmiting enter execs]", state5_enter_exec)
                                {
/* This state receives incoming packet stream from upper */
/* layer. */
pkptr = op_pk_get(SRC_STRM);
op_pk_nfd_get(pkptr,"DEST",&dest);

pk_transmited++;

op_stat_write (pk_transmited_hndl, pk_transmited);

// set buffer size to 20 packet at a time

if(LORA-CBF_buffer_size_get(dest) < 20)
                                        {
// data from internal source
printf("\n     {{ node %d }} LORA-CBF_data_pk :: \n", node_addr);

/ call routing function

LORA-CBF_data_pk(pkptr);
                                        }
else
                                        {
op_pk_destroy(pkptr);
                                        }
                                }

FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only
[Transmiting enter execs]", state5_enter_exec)

/** state (Transmiting) exit executives **/
FSM_STATE_EXIT_FORCED (5, "Transmiting", "LORA-
CBF_clusters_and_members_only [Transmiting exit execs]")
```

```
FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only
[Transmiting exit execs]", state5_exit_exec)
                                        {
                                        }
FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only
[Transmiting exit execs]", state5_exit_exec)


/** state (Transmiting) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "Transmiting", "idle")
/*---------------------------------------------------------*/

/** state (Buffer) enter executives **/
FSM_STATE_ENTER_FORCED (6, state6_enter_exec, "Buffer", "LORA-
CBF_clusters_and_members_only [Buffer enter execs]")

FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only [Buffer
enter execs]", state6_enter_exec)
                                        {
/* This state dequeue a data packet from the buffer and */
/* sends tl the mac layer. */

printf ("     {{ node %d }} @ Buffer :: \n", node_addr);

// Read the ICI associated to the interruption

buffer = op_intrpt_ici();
op_ici_attr_get (buffer, "dest", &destination);
op_ici_destroy (buffer);

// Check if the buffer is empty or not

if (LORA-CBF_buffer_is_empty(destination) == OPC_FALSE)
                                                {
printf ("     - The destination {{ %d }} has {{ %d }} packets waiting in the buffer \n",
destination, LORA-CBF_buffer_size_get(destination));

LORA-CBF_buffer_serve(destination);

op_stat_write (buffer_packets_hndl, LORA-CBF_buffer_size_get(destination));

Packet_Retransmit++;

op_stat_write (retransmissions_hndl,Packet_Retransmit- 1);

                                        }
else
                                        {
```

```
printf ("      - Buffer is empty !!! \n");

                                        }
                                }

FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only [Buffer
enter execs]", state6_enter_exec)

/** state (Buffer) exit executives **/
FSM_STATE_EXIT_FORCED (6, "Buffer", "LORA-
CBF_clusters_and_members_only [Buffer exit execs]")

FSM_PROFILE_SECTION_IN ("LORA-CBF_clusters_and_members_only [Buffer
exit execs]", state6_exit_exec)
                                {
                                }
FSM_PROFILE_SECTION_OUT ("LORA-CBF_clusters_and_members_only [Buffer
exit execs]", state6_exit_exec)

/** state (Buffer) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "Buffer", "idle")
/*---------------------------------------------------------*/
                        }

FSM_EXIT (0,LORA-CBF_clusters_and_members_only)
                }
        }

#if defined (__cplusplus)
        extern "C" {
#endif
extern VosT_Fun_Status Vos_Catmem_Register (const char * , int ,
VosT_Void_Null_Proc, VosT_Address *);
extern VosT_Address Vos_Catmem_Alloc (VosT_Address, size_t);
extern VosT_Fun_Status Vos_Catmem_Dealloc (VosT_Address);
#if defined (__cplusplus)
        }
#endif

Compcode
LORA-CBF_clusters_and_members_only_init (void ** gen_state_pptr)
        {
int _block_origin = 0;
static VosT_Address  obtype = OPC_NIL;

FIN (LORA-CBF_clusters_and_members_only_init (gen_state_pptr))
```

```
if (obtype == OPC_NIL)
                {
/* Initialize memory management */
if (Vos_Catmem_Register ("proc state vars (LORA-
CBF_clusters_and_members_only)",
sizeof (LORA-CBF_clusters_and_members_only_state), Vos_Vnop, &obtype) ==
VOSC_FAILURE)
                        {
FRET (OPC_COMPCODE_FAILURE)
                        }
                }

*gen_state_pptr = Vos_Catmem_Alloc (obtype, 1);
if (*gen_state_pptr == OPC_NIL)
                {
FRET (OPC_COMPCODE_FAILURE)
                }
else
                {
/* Initialize FSM handling */
((LORA-CBF_clusters_and_members_only_state *)(*gen_state_pptr))->current_block
= 0;

FRET (OPC_COMPCODE_SUCCESS)
                }
        }

void LORA-CBF_clusters_and_members_only_diag (void)
        {
/* No Diagnostic Block */
        }
void LORA-CBF_clusters_and_members_only_terminate (void)
        {
int _block_origin = __LINE__;

FIN (LORA-CBF_clusters_and_members_only_terminate (void))

Vos_Catmem_Dealloc (pr_state_ptr);

FOUT
        }

/* Undefine shortcuts to state variables to avoid */
/* syntax error in direct access to fields of */
/* local variable prs_ptr in LORA-CBF_clusters_and_members_only_svar function. */
#undef node_id
#undef node_addr
#undef DEBUG
```

```
#undef HELLO_MODE
#undef TTL_START
#undef TR
#undef hello_module
#undef hello_dist
#undef HELLO_INTERVAL
#undef CLUSTER_INTERVAL
#undef RequestSent
#undef RequestSeen
#undef myBroadcastID
#undef NET_DIAMETER
#undef NODE_TRAVERSAL_TIME
#undef NET_TRAVERSAL_TIME
#undef BROADCAST_RECORD_TIME
#undef ACTIVE_ROUTE_TIMEOUT
#undef MY_ROUTE_TIMEOUT
#undef mySeqNb
#undef TIMER_RENEW
#undef ackNb
#undef Location
#undef Retransmission
#undef Data_seq_number
#undef ReplySeen
#undef transmit_dist
#undef Range_predicted
#undef retransmissions_hndl
#undef Packet_Retransmit
#undef latency_hndl
#undef latency_start
#undef delay
#undef tot_delay
#undef average_delay
#undef average_delay_hndl
#undef buffer_packets_hndl
#undef buffer_time
#undef latency_stop
#undef pk_received_hndl
#undef EXPIRATION_TIME
#undef eficiency_hndl
#undef pk_transmited_hndl
#undef MINIMUM_TIME
#undef MINIMUM_DELAY
#undef Dataseen
#undef Ackseen
#undef CONVERT_CLUSTER

void LORA-CBF_clusters_and_members_only_svar (void * gen_ptr, const char *
var_name, char ** var_p_ptr)
```

```
        {
LORA-CBF_clusters_and_members_only_state                *prs_ptr;

FIN (LORA-CBF_clusters_and_members_only_svar (gen_ptr, var_name, var_p_ptr))

if (var_name == OPC_NIL)
                {
*var_p_ptr = (char *)OPC_NIL;
FOUT
                }
prs_ptr = (LORA-CBF_clusters_and_members_only_state *)gen_ptr;

if (strcmp ("node_id" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->node_id);
FOUT
                }
if (strcmp ("node_addr" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->node_addr);
FOUT
                }
if (strcmp ("DEBUG" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->DEBUG);
FOUT
                }
if (strcmp ("HELLO_MODE" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->HELLO_MODE);
FOUT
                }
if (strcmp ("TTL_START" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->TTL_START);
FOUT
                }
if (strcmp ("TR" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->TR);
FOUT
                }
if (strcmp ("hello_module" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->hello_module);
FOUT
                }
if (strcmp ("hello_dist" , var_name) == 0)
```

```
                {
*var_p_ptr = (char *) (&prs_ptr->hello_dist);
FOUT
                }
if (strcmp ("HELLO_INTERVAL" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->HELLO_INTERVAL);
FOUT
                }
if (strcmp ("CLUSTER_INTERVAL" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->CLUSTER_INTERVAL);
FOUT
                }
if (strcmp ("RequestSent" , var_name) == 0)
                {
*var_p_ptr = (char *) (prs_ptr->RequestSent);
FOUT
                }
if (strcmp ("RequestSeen" , var_name) == 0)
                {
*var_p_ptr = (char *) (prs_ptr->RequestSeen);
FOUT
                }
if (strcmp ("myBroadcastID" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->myBroadcastID);
FOUT
                }
if (strcmp ("NET_DIAMETER" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->NET_DIAMETER);
FOUT
                }
if (strcmp ("NODE_TRAVERSAL_TIME" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->NODE_TRAVERSAL_TIME);
FOUT
                }
if (strcmp ("NET_TRAVERSAL_TIME" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->NET_TRAVERSAL_TIME);
FOUT
                }
if (strcmp ("BROADCAST_RECORD_TIME" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->BROADCAST_RECORD_TIME);
FOUT
```

```
                }
if (strcmp ("ACTIVE_ROUTE_TIMEOUT" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->ACTIVE_ROUTE_TIMEOUT);
FOUT
                }
if (strcmp ("MY_ROUTE_TIMEOUT" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->MY_ROUTE_TIMEOUT);
FOUT
                }
if (strcmp ("mySeqNb" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->mySeqNb);
FOUT
                }
if (strcmp ("TIMER_RENEW" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->TIMER_RENEW);
FOUT
                }
if (strcmp ("ackNb" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->ackNb);
FOUT
                }
if (strcmp ("Location" , var_name) == 0)
                {
*var_p_ptr = (char *) (prs_ptr->Location);
FOUT
                }
if (strcmp ("Retransmission" , var_name) == 0)
                {
*var_p_ptr = (char *) (prs_ptr->Retransmission);
FOUT
                }
if (strcmp ("Data_seq_number" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->Data_seq_number);
FOUT
                }
if (strcmp ("ReplySeen" , var_name) == 0)
                {
*var_p_ptr = (char *) (prs_ptr->ReplySeen);
FOUT
                }
if (strcmp ("transmit_dist" , var_name) == 0)
                {
```

```
*var_p_ptr = (char *) (&prs_ptr->transmit_dist);
FOUT
                }
if (strcmp ("Range_predicted" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->Range_predicted);
FOUT
                }
if (strcmp ("retransmissions_hndl" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->retransmissions_hndl);
FOUT
                }
if (strcmp ("Packet_Retransmit" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->Packet_Retransmit);
FOUT
                }
if (strcmp ("latency_hndl" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->latency_hndl);
FOUT
                }
if (strcmp ("latency_start" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->latency_start);
FOUT
                }
if (strcmp ("delay" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->delay);
FOUT
                }
if (strcmp ("tot_delay" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->tot_delay);
FOUT
                }
if (strcmp ("average_delay" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->average_delay);
FOUT
                }
if (strcmp ("average_delay_hndl" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->average_delay_hndl);
FOUT
                }
```

```
if (strcmp ("buffer_packets_hndl" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->buffer_packets_hndl);
FOUT
                }
if (strcmp ("buffer_time" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->buffer_time);
FOUT
                }
if (strcmp ("latency_stop" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->latency_stop);
FOUT
                }
if (strcmp ("pk_received_hndl" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->pk_received_hndl);
FOUT
                }
if (strcmp ("EXPIRATION_TIME" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->EXPIRATION_TIME);
FOUT
                }
if (strcmp ("eficiency_hndl" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->eficiency_hndl);
FOUT
                }
if (strcmp ("pk_transmited_hndl" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->pk_transmited_hndl);
FOUT
                }
if (strcmp ("MINIMUM_TIME" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->MINIMUM_TIME);
FOUT
                }
if (strcmp ("MINIMUM_DELAY" , var_name) == 0)
                {
*var_p_ptr = (char *) (&prs_ptr->MINIMUM_DELAY);
FOUT
                }
if (strcmp ("Dataseen" , var_name) == 0)
                {
*var_p_ptr = (char *) (prs_ptr->Dataseen);
```

```
FOUT
        }
if (strcmp ("Ackseen" , var_name) == 0)
        {
*var_p_ptr = (char *) (prs_ptr->Ackseen);
FOUT
        }
if (strcmp ("CONVERT_CLUSTER" , var_name) == 0)
        {
*var_p_ptr = (char *) (&prs_ptr->CONVERT_CLUSTER);
FOUT
        }
*var_p_ptr = (char *)OPC_NIL;

FOUT
}
```

APPENDICE C: STATISTICAL ANALYSIS FOR AD-HOC WIRELESS NETWORKS IN INTER-VEHICULAR COMMUNICATION

C.1 INTRODUCTION

To meet performance goals prescribed for user traffic, a network must be able to adapt its behaviour to accommodate for changes in its intrinsic properties of connectivity and capacity **[24] [99]**. In the following sections, we will describe these properties in more detail. Performance goals in inter-vehicular networks will normally include statistical analysis of vehicular traffic, investigation of topological dynamic ranges and throughput.

C.2 STATISTICAL ANALYSIS OF VEHICULAR TRAFFIC

Chapter 5 presented previously, an introduction to vehicular traffic theory. Here we will look at the statistical distribution of velocity, time gaps and distances, which are three useful indicators that permit us to simulate a realistic environment of vehicles on a motorway.

These quantities are described by random variables v, τ, and d. The value of velocity in our application has a normal distribution **[100]**. Therefore, the probability density function (pdf) and the probability distribution function (PDF) can be written as

$$p_v = \frac{1}{\sigma\sqrt{2\pi}} \cdot e - \frac{(v-\mu)^2}{2\sigma^2} \quad \text{and} \quad P(v \le V) = \frac{1}{\sigma\sqrt{2\pi}} \int_0^v e - \frac{(v-\mu)^2}{2\sigma^2} dv \qquad \text{(C.1)}$$

whereby μ and σ are the mean and standard deviation of velocity. We have assumed that the vehicles drive at an average speed of 150 km/h on a motorway with low traffic density (undisturbed traffic), where vehicles with a speed of 190 km/h are considered to

be at the upper limit. These premises are reasonable because there is no regulation above this speed. Based on this assumption, the standard deviation is σ = 40 Km/h. The normal distribution guarantees that 68.26 % of the vehicles considered will be driven at between 110 km/h to 190 km/h on a motorway (Figure C.1).

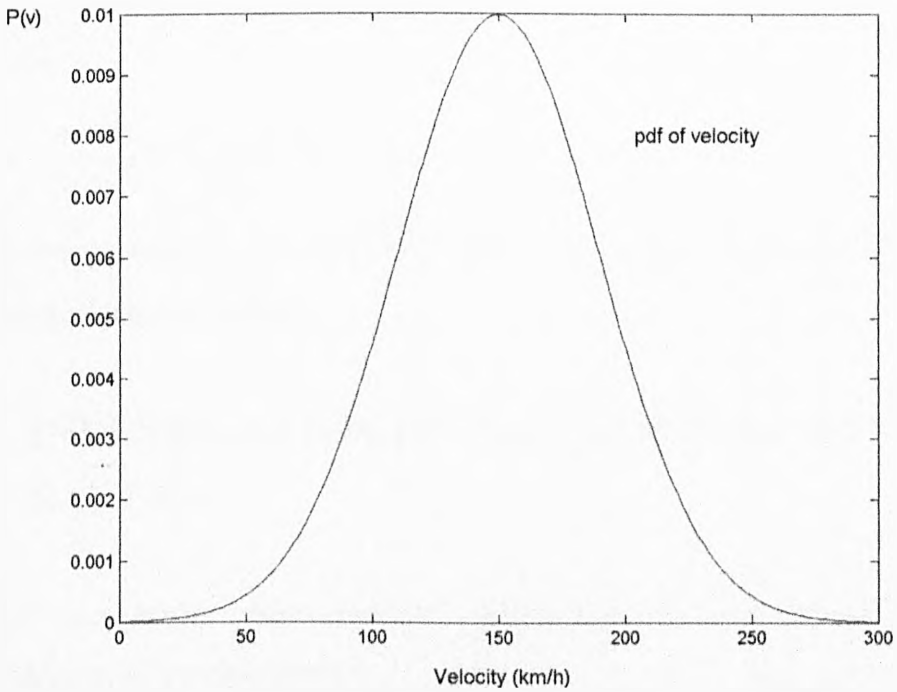


Figure C.1: probability density function of Velocity.

On the other hand, the Probability Distribution Function (PDF) cannot be solved analytically. Thus, the following results are obtained by numerical computation with MATLAB. We have considered limits of $\mu-\sigma$ to $\mu+\sigma$ for our distribution. Results for the PDF have provided $P(v \le \mu-\sigma) = 15.73\%$ and $P(v \le \mu+\sigma) = 83.49\%$. This means that approximately 15 % of the vehicles deviate from the upper and lower limit.

According to **[101]**, cars can be assumed to have arrivals that have a Poisson distribution. This then means that the time gaps between cars are distributed according to the probability density function **[102]** shown in equation C.2.

$$p_\tau(\tau) = q \cdot e^{-q\tau} \qquad \text{(C.2)}$$

In C.2, q represents the traffic flow in vehicles per second.

Using the distribution of the time gap between vehicles, it is possible to obtain the pdf of vehicle separation d, substituting equation (5.1) found in Chapter 5.

$q = \frac{1}{\tau_m} = v_m \cdot \left(\frac{1}{\frac{300}{\rho_{veh}} - l_m} \right)$ in (C.2), we obtain the probability density function of vehicle separation.

$$p_d(d) = \frac{q}{v_m} \cdot e^{-q\frac{d}{v_m}} = \frac{\rho_{veh}}{300} \cdot e^{-\frac{\rho_{veh}}{300} \cdot d} \qquad \text{(C.3)}$$

Note that equation (C.3) neglects the value of l_m, because it is insignificant here. l_m is the average length of vehicles.

C.3 INVESTIGATION OF TOPOLOGICAL DYNAMIC RANGES

When considering inter-vehicular communication, one should consider the circumstances of the endpoints.

A typical factor is the direction of travel. For example, if they are in the same lane, on the same carriageway or on opposite carriageways. The purpose of the analysis is to derive the probabilities of two endpoints having time to set up a link, given the properties of the specific protocols concerned.

To illustrate a simple case, let us assume that the velocity of both endpoints is constant and all vehicles are driving in the same direction. Let vehicle A be a reference point with R_{comm} the distance between two cars changing from $d = -R_{comm}$ to $d = R_{comm}$. Because, the communication distance d is given by $d = 2 \cdot R_{comm}$, we can calculate the probability distribution function (PDF) of communication duration as:

$$P_t(t) = \frac{4 \cdot R_{comm}}{\sigma_{\Delta v}\sqrt{2\pi}} \cdot \frac{1}{t^2} \cdot e^{\frac{\left(\frac{2 \cdot Rcomm}{t} - \mu_{\Delta v}\right)^2}{2\sigma_{\Delta v}^2}} \qquad \text{for } t \geq 0 \qquad \text{(C.4)}$$

in which the subscript Δv refers to the velocity difference in equation C.4.

According to the probability density function of communication duration, the vehicles can remain connected for 27 seconds, considering only one direction. However, this is with rather low probability (Figure C.2).

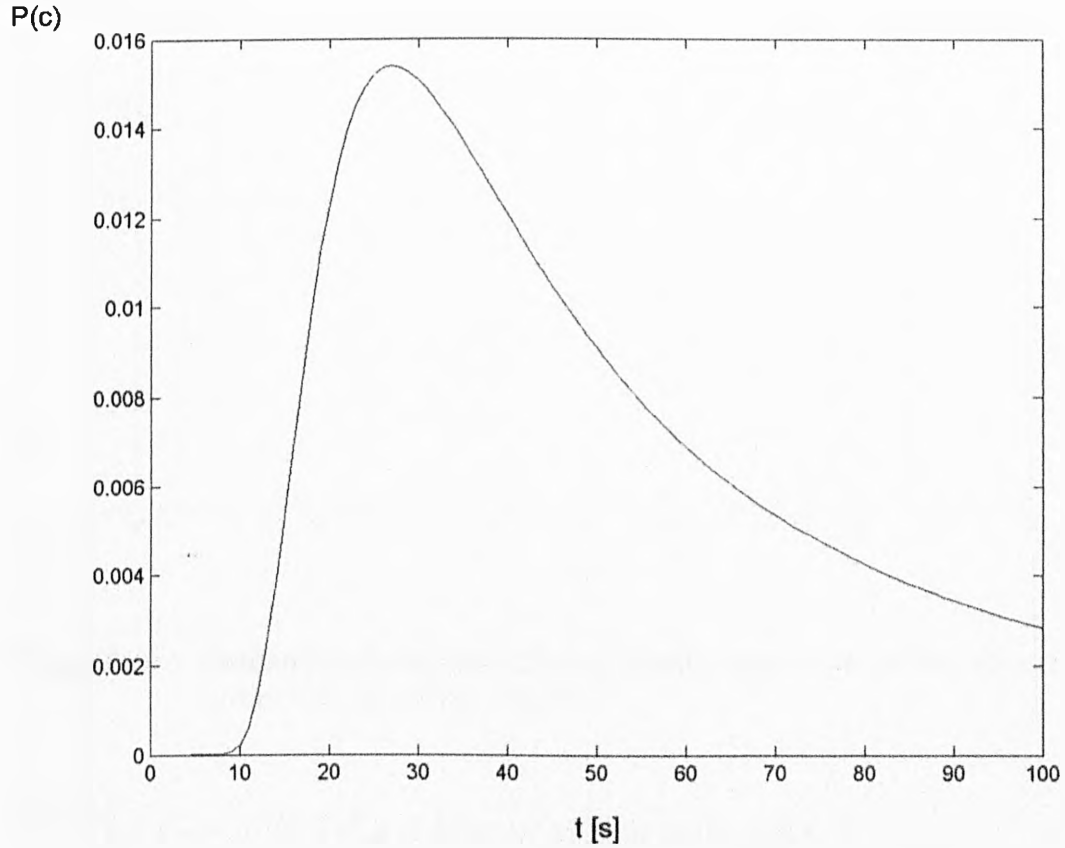


Figure C.2: ***Probability density function of communication duration considering only one direction.***

Now, we continue our discussion by considering the oncoming traffic scenario. The main difference is the $\mu_{\Delta v,on} = \mu_2 + \mu_1$, in which $\mu_{\Delta v,on}$ is the average of a velocity difference considering oncoming traffic, and μ_2 and μ_1 are the average values of velocity. To calculate communication duration, one simply has to substitute $\mu_{\Delta v}$ by $\mu_{\Delta v,on}$ in the equation (C.4), to obtain the probability density function for oncoming traffic on a motorway (Figure C.3).

The pdf for communication durations shows that the probability of remaining connected for 10 seconds for oncoming traffic is highly probably at a speed of 150 km/h.

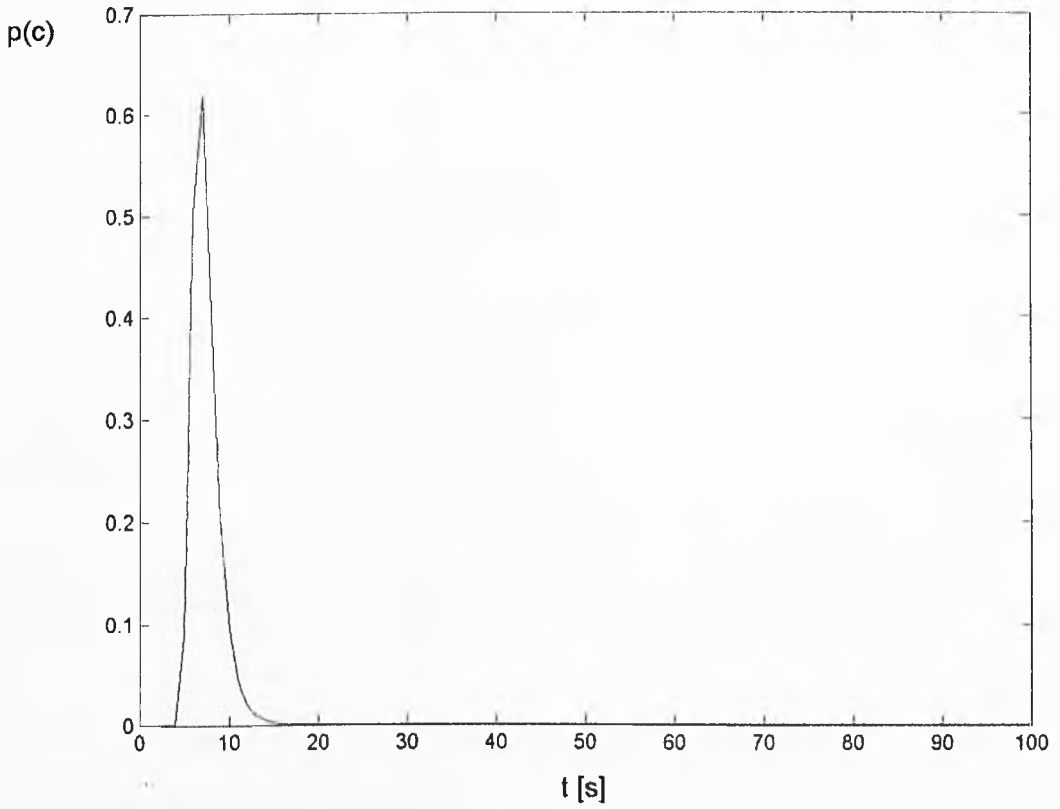


Figure C.3: ***Probability density function of communication duration between vehicles in oncoming traffic.***

C.4 The Capacity of Ad-hoc wireless networks

Due to spatial separation, several nodes can carry out wireless transmissions simultaneously and the capacity of ad-hoc wireless networks is affected by the mutual interference of concurrent transmissions between nodes **[103]**. The study of the capacity of wireless ad hoc networks has received significant attention recently, according to **[104]**, **[105]** and **[106]**. Basically, this research considers n identical randomly located nodes, each capable of transmitting at W bits/sec and using a fixed range. The throughput capacity $\lambda(n)$ obtainable by each node for a randomly chosen destination is:

$$\lambda(n) = \frac{W}{\sqrt{n \log n}} \text{ bits/sec} \qquad \text{(C.5)}$$

Equation (C.5) defines the throughput capacity for each node in a wireless ad-hoc network for a randomly chosen destination under a non-interference protocol. Here, packets are sent from node to node in a multi-hop fashion until they reach their final

destination. Also, they can be buffered at intermediate nodes while awaiting transmission.

C.5 THROUGHPUT IN IEEE WIRELESS NETWORKS

In order to determine the throughput of IEEE 802.11 and IEEE 802.11b, it is necessary to analyze the MAC layer. The format of a data packet consists of the overhead (preamble and header) and the data segment. The time to transmit this packet is shown below:

$$t_T = DIFS + Overhead + \frac{Data(bits)}{Rate(bits/\sec)} + SIFS + ACK \qquad \text{(C.6)}$$

DIFS, SIFS and ACK frames are considered here because they are necessary to ensure the correct reception of a packet.

SIFS = 10 μs, DIFS = 50 μs, ACK = 112 μs

Overhead = Preamble + Header = 144 μs + 48 μs = 192 μs

Rate = 1 and 11 Mbps

We will consider a data packet of 1000 bytes.

Using equation C.6, we can determine the time required to transmit a data packet of 1000 bytes using IEEE 802.11 and IEEE 802.11b wireless cards.

$$t_T = 50\mu s + 192\mu s + \frac{(1000*8)bits}{1*10^6 bits/s} + 10\mu s + 112\mu s$$

t_T = 8.36 ms for IEEE 802.11 and 1.09 ms for IEEE 802.11b

The throughput for wireless radios is given by:

$$\text{Throughput} = \frac{Data}{t_T} \cdot (1 - PER)\,(\text{bits/sec}) \qquad \text{(C.7)}$$

$$PER = 1 - (1 - P_e)^{number\_of\_bits\_in\_the\_packet} \quad (C.8)$$

The equation (C.7) expresses the throughput, considering the time required to transmit a data packet and the probability of a packet being received in error (PER). In (C.8) we can observe that the PER of the system is related to the packet size, and the greater the packet size, the greater the PER. This is because the packet needs more time to travel from the source to the destination, and this increases the probability of interference.

The probability of bit error (P_e), for IEEE 802.11 wireless radios is given by:

$$P_e = \frac{1}{2 \cdot \left(1 + \frac{E_b}{N_0}\right)} \quad (C.9)$$

where E_b represents the energy of the bit at the receiver and N_0 is the noise power density.

For example, the throughput for IEEE 802.11 wireless radios is shown in Figure C.5, which is reached at a distance of 300m at 955.46 kbps. According to equation C.6 the time to transmit a packet of 1000 Bytes is equal to 8.36 ms. Another way to verify the result obtained is by using the following equation:

$$Throughput = \frac{Data(bit/\sec)}{t_T} \quad (C.10)$$

$$Throughput = \frac{1000 * 8bits}{8.36ms}, \qquad \text{Throughput} = 956.9 \text{ kbps}$$

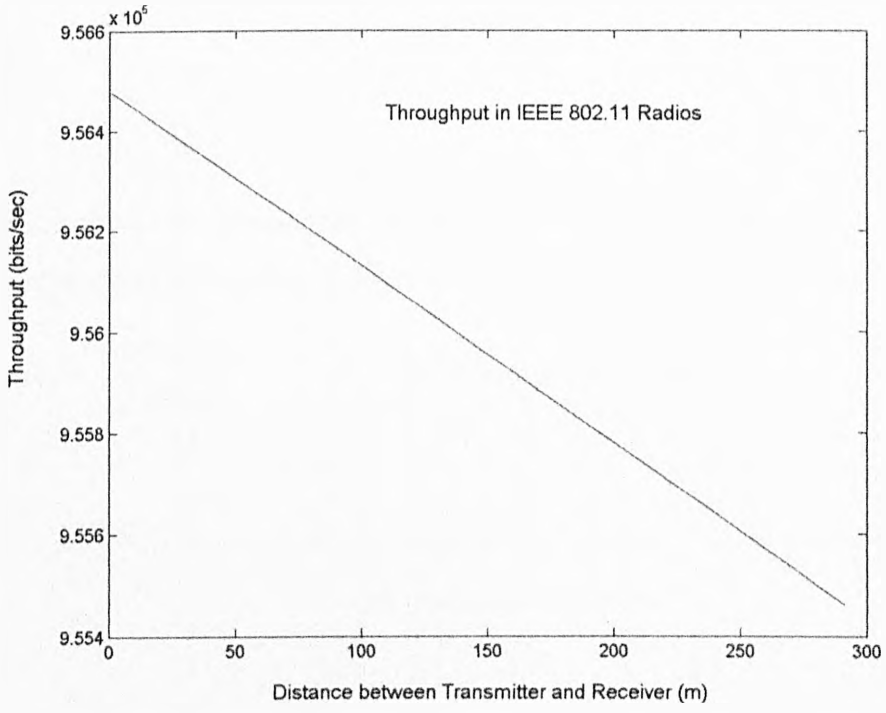


Figure C.5: Throughput for IEEE 802.11 wireless radios in terms of distance to receiver

Similarly, the Bit Error Probability (P_e), for IEEE 802.11b radios can be expressed in the following equation:

$$Pe = \int_0^{\infty} \left[1 - \int_{-x}^{\infty} \left(\frac{1}{\sqrt{2\pi}} \int_{-(v+x)}^{v+x} \exp\left(-\frac{y^2}{2}\right) dy \right)^{\frac{M}{2}-1} \cdot \exp\left(-\frac{v^2}{2}\right) dv \right] \cdot \frac{1}{2 \cdot p} \exp\left(-\frac{x}{2 \cdot p}\right) dx \qquad \text{(C.11)}$$

using the next approximation for Pe:

$$Pe = \frac{2^{k-1}}{2^k - 1} \left(\sum_{m=1}^{M-1} \frac{(-1)^{m+1} \binom{M-1}{m}}{1 + m + m8\Gamma} \right) \qquad \text{(C.12)}$$

where:

$M = 2^{k-1}$, k is the number of bits in the symbol

$$\Gamma = \sqrt{\frac{2Eb}{No}} \qquad \text{(C.13)}$$

Figure C.6 shows the throughput for IEEE 802.11b wireless radios. Here, the throughput reached at distance of 300m is 7.32 Mbps, employing the equation (C.10).

$$Throughput = \frac{1000x8bits}{1.09ms}, \qquad \text{Throughput} = 7.3 \text{ Mbps}$$

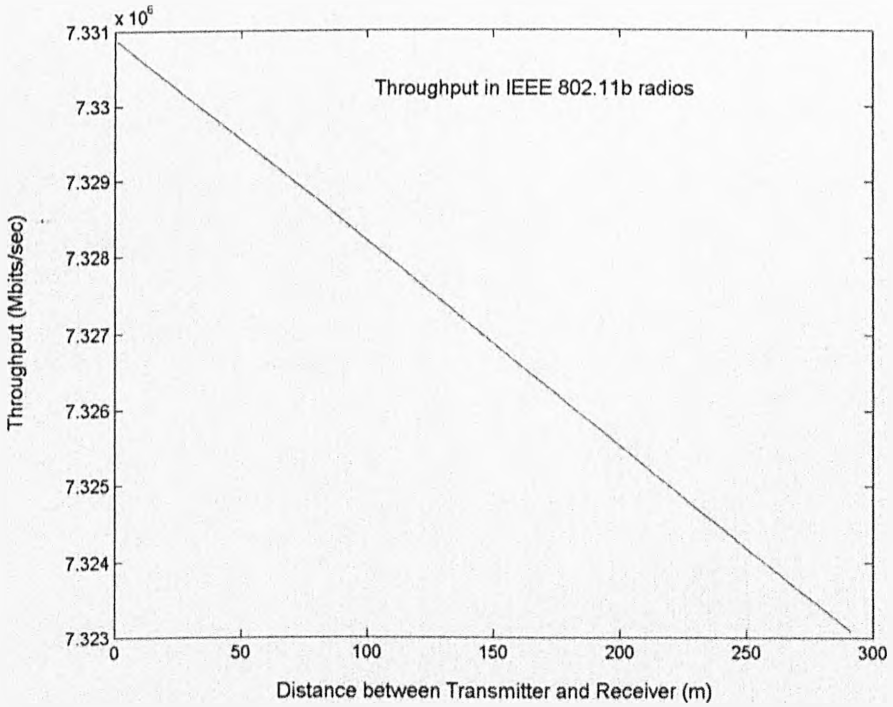


Figure C.6: Throughput for IEEE 802.11b wireless card in terms of distance to the receiver.