NTNU – Trondheim
Norwegian University of
Science and Technology

Forwarding algorithms in vehicular networks

Amir Afroozeh

Master of Telematics - Communication Networks and Networked Services (2
Submission date: February 2014
Supervisor: Tor Kjetil Moseng, ITEM

Norwegian University of Science and Technology
Department of Telematics

NTNU – Trondheim
Norwegian University of
Science and Technology

Forwarding algorithms in vehicular networks

Amir Afroozeh

Submission date: February 2014
Responsible professor: Tor Kjetil Moseng
Supervisor:

Norwegian University of Science and Technology
Department of Telematics

Abstract

Vehicular ad hoc networks are a kind of mobile ad hoc network in which the mobile nodes are fast moving vehicles. The fast movement of vehicles causes considerable disconnections in network links and leads to changes in the topology of the network. Due to frequently changing topology in vehicular ad hoc networks, finding reliable routes which increases network performance is very important.

Finding reliable routes in vehicular ad hoc networks has received considerable attention by researchers. The goal of this thesis is to find the most stable routes by considering various mobility parameters by applying a fuzzy logic controller tool. Fuzzy logic is an artificial intelligence (AI) technique which is a form of multi-valued, as opposed to fixed and exact logic. Fuzzy logic has good performance in pattern classification and decision making systems. I considered the following parameters as the input to my fuzzy logic: the direction of two intermediate nodes, intermediate node's position, same directional node compared to source and destination and relative velocity compared to source and destination.

Different simulation scenarios are designed, tested and finally, the advantages of my proposed algorithm compared to normal AODV based on the various parameters, such as end to end delay and delivery rate, are presented. At the end, several requirements of quality of service that can be applied in vehicular networks are discussed and the future work is addressed.

Acknowledgement

This master project was carried out at the department of networks and computer (Telematics) at the Norwegian University of Science and Technology, NTNU. The author gratefully acknowledges Professor Tor Kjetil Moseng for his patience, motivation and valuable guidance during the process of completing the master thesis. I would also like to thank Ph.D. student Ameen Chilwan for sharing his knowledge and for his constant support during the project. Finally, I would like thank my three brothers, Ali, Azim and Omid, and especially my parents Masoumeh and Abbass for their support and understanding.

Amir Afroozeh
NTNU, Trondheim, Norway
February, 2014

Problem Description

Intelligent Transport Systems (ITS) is the utilization of ICT in the transport sector to improve safety and increase efficiency and convenience. One essential enabler for ITS is communication between vehicles and infrastructure, and from one vehicle to another. The vehicles communicate to each other in order to forward information. Information forwarding may, especially in congested areas, be challenging due to the shared media and QoS requirements.

The main aim of this thesis is to propose an information forwarding algorithm that efficiently disseminate information within the vehicle network based on existing literature. In addition, the algorithm is tested through simulations in realistic environment scenarios and analysed for different performance parameters such as delay and loss. Finally, the algorithm is be discussed in relation to different QoS requirements that may apply in vehicle networks.

Assignment Given: October, 2013

Supervisor: Prof. Tor Kjetil Moseng

Contents

List of Figures

List of Tables

Listings

Acronyms

ITS Intelligent Transport Systems

VANET Vehicular Ad-hoc Network

MANET Mobile Ad hoc Network

SUMO Simulation of Urban MObility

AODV Ad hoc On-Demand Distance Vector Routing

OSPF Open Shortest Path First

NS2 Network Simulator 2

NS3 Network Simulator 3

RREQ Route Request

RREP Route Reply

RREP Route Error

MF Membership Function

RSU RoadSide Unit

V2V Vehicle to Vehicle

V2I Vehicles to Infrastructure

WSN Wireless Sensor Networks

DOS Denial-Of-Service

GPS Global Positioning System

DVD Digital Video Disk

GSM Global System for Mobile Communications

QoS Quality of service

AOMDV On-Demand Multi-path Distance Vector

SAODV Secure version of AODV

NES Neighbor Elimination Scheme

AI Artificial Intelligence

ANN Artificial Neural Network

FFAR Four-inputs Fuzzy-logic-based AODV Routing protocol

FLS Fuzzy Logic System

NAM Network Animator

DSDV Destination-Sequenced Distance Vector routing

DSR Dynamic Source Routing

TORA Temporally-Ordered Routing Algorithm

TCP Transmission Control Protocol

Wi-Max Worldwide Interoperability for Microwave Access

GloMoSim Global Mobile Information System Simulator

NS3 Network Simulator 3

GHz gigahertz

GB gigabyte

RAM Random-Access Memory

UDP User Datagram Protocol

CBR Constant Bit Rate

MAC Media Access Control

Mbit megabit

dbm power ratio in decibels

m meter

cm centimeter

ICT Information and Communications Technology

LAN Local Area Network

AP Access Point

TSA Traffic Safety Applications

DMEMD Distributed MAC scheme for Emergency Message Dissemination

CVIA Controlled Vehicular Internet Access

HPP High Priority Period

LPP Low Priority Period

VCWC Vehicular Collision Warning Communication

CVIA-QoS Controlled Vehicular Internet Access protocol with QoS

FCAR Fuzzy Control Based AODV Routing

Chapter 1
Introduction

1.1 Overview

Traffic jam is a challenging issue in the 21th century. Nowadays, citizens and governments are faced with an increasing number of vehicles. This issue is much more critical in developing courtiers, such as India and China, where there is no balance between the improvement of road infrastructure and the growth of traffic. Moreover, statistics show that only relying on the construction and expansion of transport infrastructure cannot be the ultimate solution for this problem. Therefore, there is a demand to find ways to decrease, optimize and manage considerable costs of traffic jam for governments and countries, and to find ways for having safe, quick and convenient transportation [2].

There are different solutions for this problem, and one of the novel approaches is intelligent transportation system (ITS). ITS is a novel idea to manage and mitigate traffic issues. ITS is an umbrella term for a variety of novel technologies trying to evaluate, observe and analyse traffics, and integrate various technologies to obtain these aims: traffic efficiency, saving cost, energy efficiency, safe environment and reducing time [1]. The term "Intelligent Transport System" includes a wide range of systems such as portable systems, standalone systems installed on vehicles, systems that enable vehicle-vehicle and vehicle-infrastructure communication, and finally cooperative systems [2]. To sum up, ITS tries to establish a safe and convenient environment for drivers and a sustainable connection between vehicles.

A wireless ad hoc is a decentralized wireless network [3] which does not depend on pre-existing infrastructure like routers in a wired network or an access point in a managed wireless network. In other words, a wireless ad hoc network is an answer to novel wireless structures that are self healing and self organizing in which each node can participate in routing and all of devices have equal status in the network and they can have connection in specific range with other deceives. The term "ad-hoc networks" is usually associated with a form of operation of IEEE 802.11 wireless

networks [64].

There are various types of ad hoc networks and one of them is called VANET [4]. Vehicular Ad Hoc Network (VANET) is a branch of Mobile Ad hoc Network (MANET). MANET and VANET have some common characteristics such as low bandwidth and self organization and shared radio transmission. The main duty of VANET is the provision of vehicle-vehicle wireless communication and vehicle-infrastructure communication (e.g., between vehicles and road side equipments), and these connections can be established without central access. The communication between vehicles has some specifics such as high speed and mobility, and that is the key feature of vehicular ad-hoc networks that makes them unique in the context of MANETs [4].

By using vehicle to vehicle communications, drivers can be notified of important traffic data such as the condition of roads and accidents. Such information will improve drivers' decisions in hard conditions. Moreover, vehicular communications will help to monitor and manage traffic distribution and to improve vehicle fuel economy.

1.2 Research objectives

Routing algorithms are an important part of a vehicular ad hoc network where they forward information in order to connect vehicles. Artificial intelligence is a remarkable technique for improving routing algorithms. There are various artificial intelligence techniques which can be applied in a VANET system and one of them is fuzzy logic. Fuzzy logic is a kind of multi valued logic rather than fixed and binary logic and it can be applied as a tool which has good performance in classification of pattern and decision making systems.

This thesis aims to fulfil the following goals:

1. Studying the AODV algorithm in order to make familiar with its functions such as routing maintenance and routing discovery procedure.
2. Studying the concept of fuzzy logic and its functions.
3. Studying general routing algorithms for vehicular networks.
4. Studying the related work in the area of forwarding algorithms in VANET regarding to fuzzy logic and AODV.
5. Proposing a novel algorithm which improves the links stability based on studied literature.

6. Choosing appropriate procedures and tools to test the proposed algorithm.
7. Evalutating the proposed algorithm based on various network performance parameters such as end to end delay.
8. Discussing the possible Quality of Service which can be applied in relation to the proposed algorithm.
9. Discussing future work.

1.3 Research methodology

This work is based on my specialization project in which I studied different routing algorithms in VANET. Based on the studied literature, I propose a new algorithm which improves the normal AODV in metrics such as end to end delay.

A deep understanding of C++, Tcl/Otcl scripts and AWK was obtained to meet the specified requirements. Working with NS2 was the challenging part of this project and I spent considerable time to understand it deeply. Moreover, I learned about the concept of fuzzy logic and implemented the fuzzy logic codes in C++. Finally, considerable time was spent on debugging and tracing for fuzzy logic codes and NS2.

1.4 Outline

Chapter 1 of the present thesis contains four sections: Section 1 explains AODV as the basic algorithm studied in this research. Section 2 illustrates the concept of fuzzy logic. Section 3 describes general VANET routing algorithms, and their characteristics, applications, classifications and challenges. Finally, section 4 describes the related work regarding to AODV algorithm and fuzzy logic.

Based on the studied literatures, a novel algorithm is proposed in chapter 2 and it will be explained how this algorithm improves network performance by applying fuzzy logic. Chapter 3 of the this thesis is about the essential tools applied to testing the algorithm and obtaining the results.

Chapter 4 evaluates and interprets the obtained results. Different Quality of Service requirements which can be applied in VANET and the possible future work are discussed in chapter 5. Finally, the last chapter provides a conclusion to this work.

Note that, Harvard style is applied for references. All the figures and source codes which are cited from literatures are referenced. In other words, figures and source codes which do not have references are made by myself.

Chapter 2
State of the art

In this chapter, I introduce the necessary background information which helps us to understand the next chapters. The study of Ad-hoc On-Demand Distance Vector Routing (AODV) protocol and then an explanation of fuzzy logic concept, applied in this research, are presented. Finally, general forwarding algorithms in vehicular network and related work are presented.

2.1 Overview of AODV

AODV is classified as a member of Bellman-Ford distant vector protocol which is ready to work in mobile network. AODV is a reactive and an **on demand** distance vector routing protocol, implying that it searches for a route **only** when a node needs a route for sending packets to a destination. Proactive routing [5] algorithms such as OSPF will disseminate their routing tables periodically in order to keep lists of destinations and their routes. Therefore, this type of algorithm is fast in reacting to network modification. But it has considerable overhead, and is only appropriate for small topology networks. On the other hand, reactive algorithms will distribute routing request packets when it is required; as a result, the network overhead is very low, but the latency is high [5] compared to proactive algorithms. Moreover, AODV ensures loop-free routes by counting the sequence numbers [6] that is determining route freshness. This mechanism and the other important mechanisms, which help us to better understand AODV, are described in the coming sections.

2.1.1 Terminology

In this section, the essential terminologies are explained. (see [6], [7])

1. **Active route** A route is called active when it contains a routing table entry with the valid state. Only active routes can send data packets.

2. **Broadcasting** Broadcasting is applied to disseminate AODV messages throughout the ad hoc network and each node will send messages to all its neighbours.

3. **Destination** A destination refers to an IP address to which data should be forwarded.

4. **Forwarding node** in rfc 3561, a forwarding node is described as "A node that agrees to forward packets destined for another node, by retransmitting them to a next hop that is closer to the unicast destination along a path that has been set up using routing control messages" [6].

5. **Forward route** A forward route refers to a path that should be established to transmit information packets from a source node to a destination.

6. **Invalid route** A route is called invalid if it is expired. A route is expired by getting invalid state in routing table entry. An invalid route cannot be applied to transfer information, but it can store previous valid routes for a specific time. This can be used for repairing routes and coming RREQ messages.

7. **Originating node** An originating node is a node that starts an AODV route discovery message. This message can be resent by other nodes in the network.

8. **Reverse route** A reverse node is defined in the rfc 3561 as: "a route set up to forward a reply (RREP) packet back to the originator from the destination or from an intermediate node having a route to the destination" [6].

9. **Sequence number** Each time a source node sends a request message, the sequence number will be increased. It is used as a mechanism to distinguish the freshness of the node's information.

2.1.2 Packet types

There are four different types of packets that are involved in AODV [6], [7].

1. **Route Request (RREQ)**

   During the route discovery process, a source node will create and broadcast an RREQ packet. An RREQ contains the below information.

   a) Source Address
   b) Destination Sequence Number
   c) Destination Address
   d) Number of Hops to Destination
   e) Broadcast ID

f) Source Sequence Number

2. **Route Reply (RREP)** When a node replies to a request, it will create an RREP packet which contains these information:

   a) Destination Sequence Number

   b) Source Address

   c) Time when this entry is considered valid

   d) Number of Hops to Destination

   e) Destination Address

3. **HELLO** Each HELLO message is used for discovering (probing) a node's neighbours and it only contains two information:

   a) Address

   b) Sequence Number

4. **Route error (RRER)** In active routes, each node will look over the link status of next hops. If a broken link is determined, an RERR message is applied to inform other nodes that use this link. In the "Route discovery procedure" part, this mechanism will be explained in more details.

   a) Unreachable Destination IP Address

   b) Unreachable Destination Sequence

   c) DestCount Number

2.1.3 Counting to infinity

Count to infinity [7] is a critical problem for network algorithms, which will be formed when an error happens in the operation of the routing algorithm. As a result, the path to a specific destination forms a loop. By applying the sequence number, AODV avoids the "counting to infinity" problem in contrast to the classical distance vector algorithms (see Figure 2.1).

In AODV, each packet contains a sequence number that is increased just before every time it will be sent. By comparing the sequence number, another mobile node that is receiving this packet can distinguish the packet's freshness.

2.1.4 Routing discovery procedure

When an originating node requests a route to a destination which has not been selected before and there is no existing route to that destination, the originating node will broadcast an RREQ message to its neighbours. After receiving an RREQ

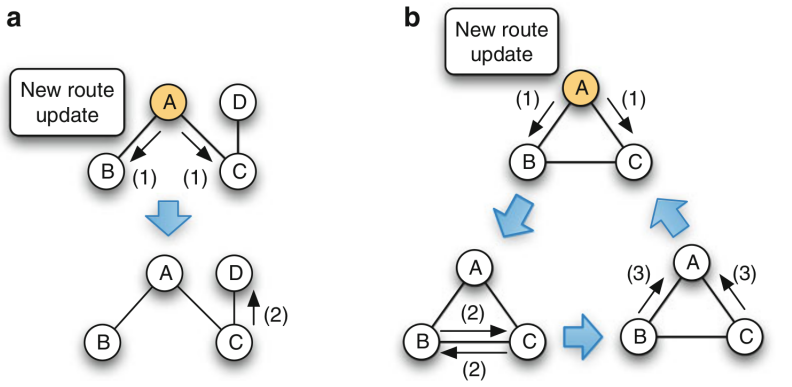

Figure 2.1: *"AODV as a distance vector protocol: (a) Tree topology – a process of route update will finish in two steps, (b) Loop topology – a process of route update goes on indefinitely" [7].*

message, if the RREQ has a greater sequence number or has the same sequence number with fewer hops compared to the destination, then the neighbour nodes will update their information. Moreover, the neighbour nodes will establish a reverese route entry for the originating node in their routing table. If the RREQ message does not have either a bigger sequence number or the same sequence number with the fewer hopes, then it will be discarded [7].

Afterwards, an RREP will be sent if one of these conditions will be satisfied: the neighbour is the destination or it has a fresher (unexpired) route to the destination [6]. Otherwise, an intermediate node will broadcast the RREQ message. As presented in Figure 2.2, node "S" broadcasts an RREQ message to its neighbours. Suppose the packet is fresh, if any neighbour has one of the conditions (it is either the destination node or it has a fresher route to the destination) it sends an RREP message, otherwise it creates a reverse route and rebroadcast the RREQ. This process is continuing until a node has the conditions, then it is time to piggyback the required routing information back to the originating node ("S") [7]. The RREP packet is carried back to the originating node and then an unicast connection between the source and the destination (it is named "D") is established (see Figure 2.2). Finally, data packets are transmitted to the destination until there is no disconnection in the route (no RRER message is denominated).

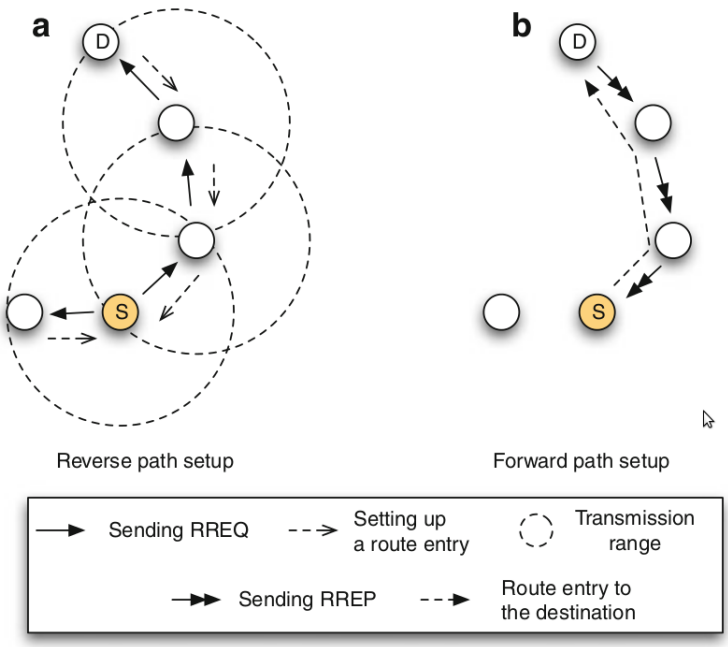

Figure 2.2: *Routing discovery procedure [7]*

2.1.5 Route maintenance

Whenever a link of an active route breaks, all routes that are using this broken link will be invalidated by the node upstream of the break. The node will broadcast an RERR message to all of its neighbours. Each RERR massage maintains the unreachable IP addresses of destinations. After receiving an RRER, each node monitors its routing table in order to find out if there are route(s) to these unreachable destinations. If yes, it will be invalidated and afterwards a new route error (RERR) will be disseminated [7]. At the end of this process, the originating node will get an RRER message and it will invalidate the unreachable route(s), and if it is needed a new RREQ message will be initiated.

2.2 Fuzzy logic

This part is written based on these papers: [8], [9].

2.2.1 Introduction

Fuzzy logic [10], [11] is a form of multi-valued logic rather than fixed and exact logic. In the traditionally binary sets, each variable can take a true or false value while fuzzy variables can take a true value that is between 0 and 1 [9]. In other word, instead of having absolute truth, it is possible to have partial truth that provides a very valuable flexibility for reasoning by taking into account inaccuracies and uncertainties [8].

One of the fuzzy logic's benefits is that the fuzzy rules are set in natural language; for example, if the weather is cold or if my speed is slow. These examples show that the words like cold or slow are more compatible with human-logic than the the numbers. For instance, this sentence "if the weather is -1.2 C°, then turn on the heater" is quite unfamiliar for people [8]. Finally it should be mentioned that fuzzy logic can be applied as an artificial intelligence based decision making system that has good performance in the pattern classification and the decision making systems [9].

2.2.2 Fuzzy sets

The basis of fuzzy logic is the fuzzy sets [11], [12] theory and the theory of fuzzy sets is actually a generalization of the classical theory; in other words, the classical set theory is subset of the theory of fuzzy sets (see Figure 2.3)

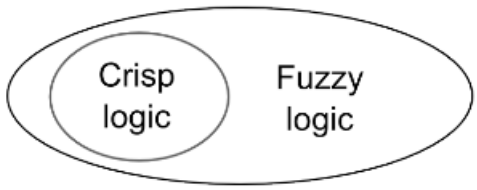

Figure 2.3: *Fuzzy sets theory via classical sets theory [9].*

Figure 2.4 demonstrates how the days of week can be considered as a weekend in fuzzy logic compared to binary logic. The weekend example attempts to explain the meaning of fuzzy sets by asking some basic questions. Is Saturday is a weekend? How about Sunday? Do you feel Friday as a weekend? How about Monday? The left plot presents the truth values for weekend-ness in the two value mode, and the right plot shows that you can consider Friday somehow as a weekend. As shown, in binary logic **only** Saturday and Sunday are counted as weekend [8].

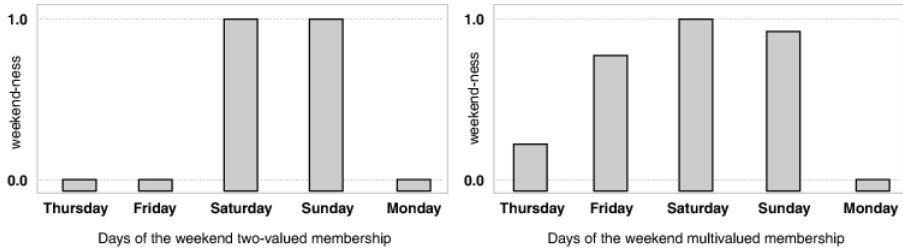

Figure 2.4: *Fuzzy weekend via two-value weekend [8].*

Membership function

A membership function (MF) is a "curve that defines how each point in the input space is mapped to a membership value (or degree of membership) between 0 and 1" [8]. One of the key features of fuzzy logic is that a value can be a member of many sets at the same time. If age can be old, young and very old, then it can have three values at the same time, but in various degree of memberships. For example, age can be 0.6 old and 0.3 young at the same time [8].

One of the famous example to explain fuzzy membership is tallness [8]. Given different people with various tallness, I want to classify them into two groups of tall and short. People taller more than 6 feet are considered as tall, then it is clearly an absurd classification, because a person is just 1 cm less than 6 feet will be considered as short. Figure 2.5 shows how the curve transits from non-tall to tall very smoothly (this curve is called as MF).

There are various types of membership functions. Most popular ones are: triangular, trape-zoidal, piecewise linear, Gaussian, or singleton. I applied the triangular form in this research.

2.2.3 Linguistic variables

Linguistic variables [9] refer to the non-numeric inputs and outputs variables of a fuzzy logic system which are words or sentences from the natural language. For example, age can have a value of young or old. Each linguistic variable can be decomposed into a set of linguistic terms. Figure 2.6 shows the linguistic variables for service function and it can be poor, good and excellent.

2.2.4 Fuzzy operators

The traditional two-value (bivalent) logic applies the boolean operators of AND, OR and NOT in order to perform complement, union and intersect operations. The

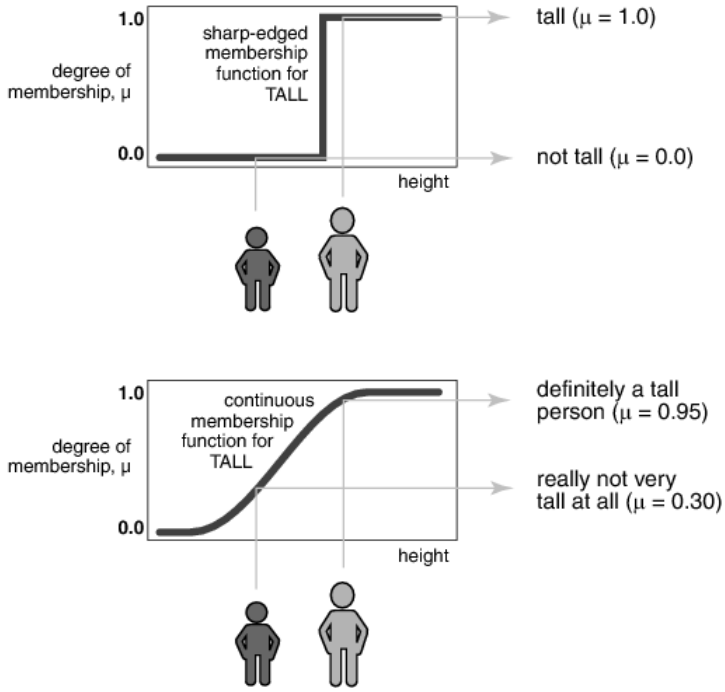

Figure 2.5: *Membership functions [8]*

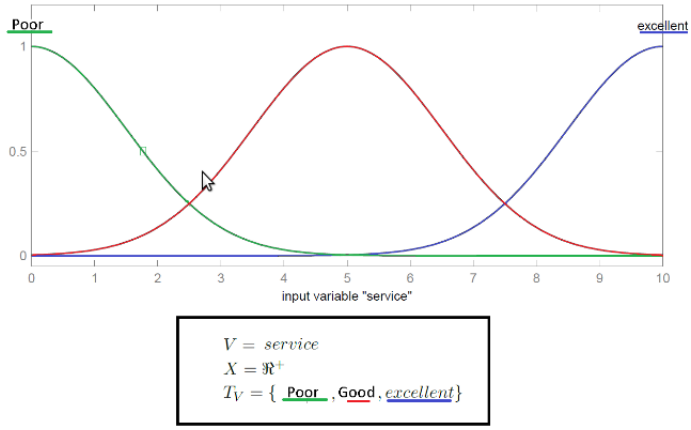

Figure 2.6: *Linguistic variables [9].*

below Table shows how the bivalent logic works.

X	Y	X AND Y	X OR Y	NOT X
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Table 2.1: *Truth table for binary logic [13].*

This truth table [13] is working appropriately for bivalent logic, but fuzzy logic needs operations that can cover all the possible fuzzy values (all the real numbers between 0 and 1); on the other hand, fuzzy logic is not limited to a (finite) set of input values and it needs the defined operations for all of these values. Table 2.2 explains the fuzzy table operations.

x AND y	min(x,y)
x OR y	max(x,y)
NOT x	1 - x

Table 2.2: *Fuzzy operators [13].*

Finally, Table 2.3 can visualize how the fuzzy logic operations can cover both of bivalent and fuzzy combinations; in other words, the bivalent operations are one special case of fuzzy operations [8].

X	Y	min(X,Y)	max(X,Y)	1-X
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0
0.2	0.5	0.2	0.5	0.8
0.7	0.2	0.2	0.7	0.3
0.6	0.6	0.6	0.6	0.4

Table 2.3: *Truth table for fuzzy logic [13].*

2.2.5 Reasoning in fuzzy logic

In the classical logic the rules are made based on these two forms:

1. **If p then q.**
2. **p true then q true.**

While, fuzzy reasoning or approximate reasoning is formed based on the rules which are written in natural language. For instance, if (the weather is cold outside), then (heater volume should be high) [9].

Implication

The fuzzy implication does not have just a single definition like other fuzzy operators, and there are considerable various implications. Two of them are more famous which are mentioned in Table 2.4.

Name	Truth value
Mamdani	min (fa (x), fb (x))
Larsen	fa (x) × fb (x)

Table 2.4: *Fuzzy implication [9].*

Figure 2.7 presents when the rule "If (the food quality is delicious), then (tip is high)" is applied and the Mamdani implication is used, this result will be obtained.

Figure 2.7: *Mamdani implication [9].*

The premise of a fuzzy rule can be made by combination of various fuzzy propositions while the operators of AND, OR and NOT participate in combining process.

Figure 2.8 explains where the quality of service is 7.83 out of 10, the food's quality is 7.32 out of 10 and the rule is "If (the service is excellent and the food is delicious), then (tip is high)", then this result will be obtained. Note that I used the Mamdani implication and the translation of OR by MAX.

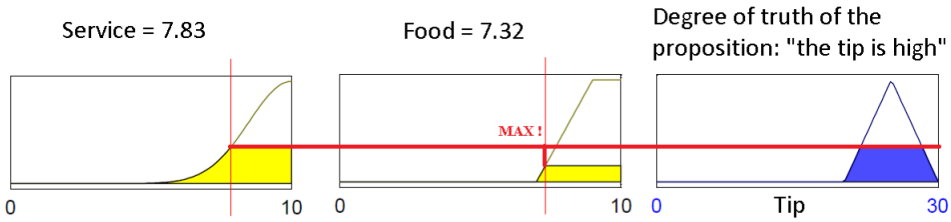

Figure 2.8: *Mamdani implication and translation of OR by MAX [9].*

2.2.6 Aggregate all outputs

In order to make a decision in a fuzzy logic system all of rules should participate. The process of combination of all rule's outputs into a single fuzzy set is called aggregation. Figure 2.9 shows how the rules are combined and finally a single result is aggregated.

2.2.7 The defuzzification

The final step is to get a non fuzzy value (crisp output). This step is called defuzzification which [8] assigns a real value or particular decision to the membership degrees of the fuzzy sets. There are various algorithms for defuzzification and one of the most popular ones is called center of gravity. Figure 2.10 presents the center of gravity algorithm.

So far, I have introduced the essential concepts that are needed to build a fuzzy control system. The fuzzy inference digram is presented in Figure 2.11 and 2.12 and as shown, the inputs values will be fuzzified, implicated, aggregated and defuzzified to obtain the output.

2.3 General forwarding algorithms in VANET and ITS

This section is written based on these papers: [14], [15].

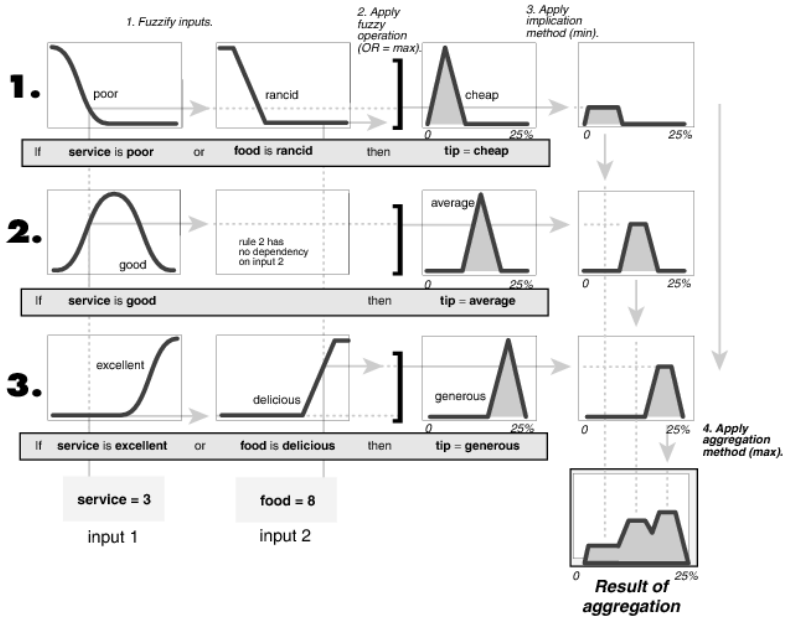

Figure 2.9: *Output aggregation [8].*

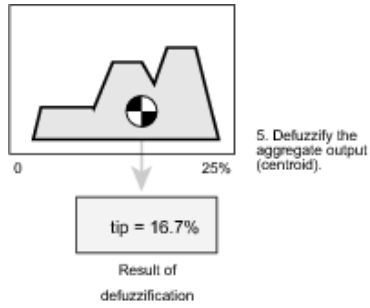

Figure 2.10: *Defuzzification [8].*

I briefly introduce wireless networks and especially ad hoc networks. Afterwards, a special kind of ad hoc network which connects two vehicles (VANET) is introduced. This section talks about some issues of VANETs such as architecture, features, challenges and classifications which should be considered for designing, implementing and evaluating new routing algorithms by researchers.

2.2.8 Conclusion

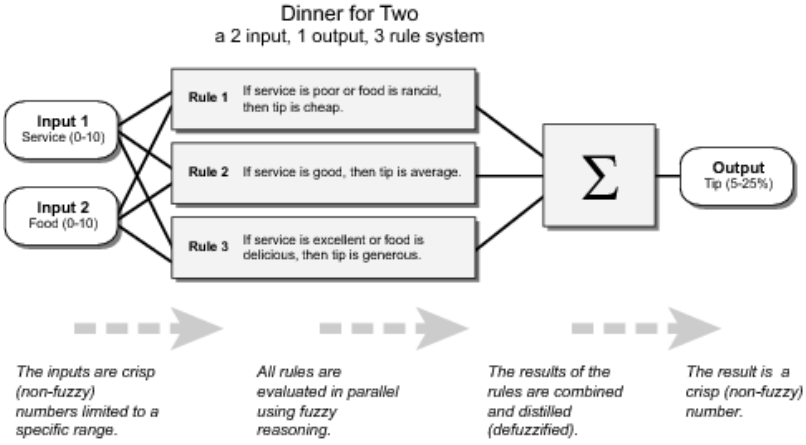

Figure 2.11: *Fuzzy logic system [8].*

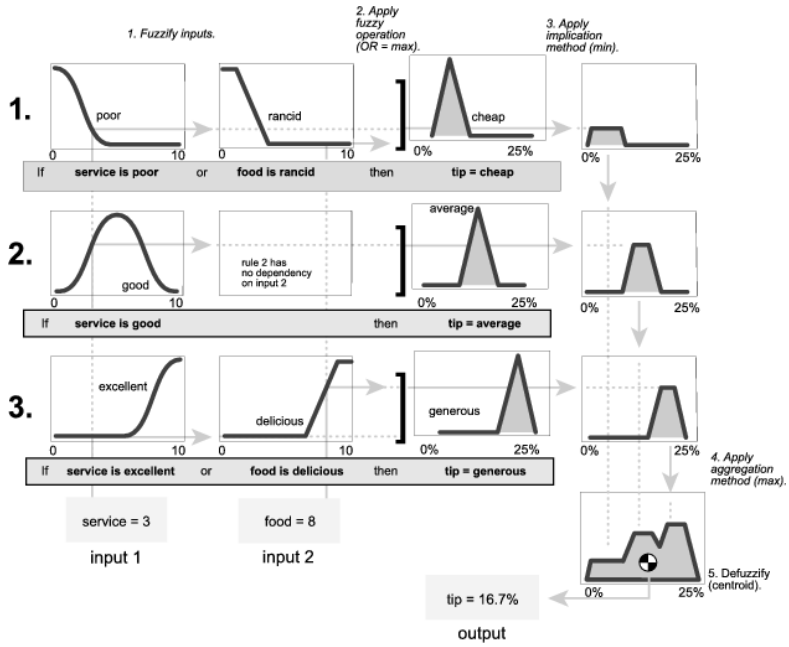

Figure 2.12: *Fuzzy inference [8].*

2.3.1 VANET architecture

Figure 2.13 presents the VANET architecture [14] which contains on-board sensors and roadside units (RSUs) which are installed along highways, roads and streets; these tools provide vehicle to vehicle connections (V2V) and vehicles to infrastructure (V2I) connections. As shown in Figure 2.13, V1, V2 and V3 have access to a fixed roadside infrastructure while V4, V5 and V6 have no connections with the fixed infrastructure. Due to limited access range of fixed base station, a considerable amount of information is transmitted via V2V connections, for example V4, V5 and V6 vehicles do not have any communications with the fixed infrastructure and they receive information via other vehicles.

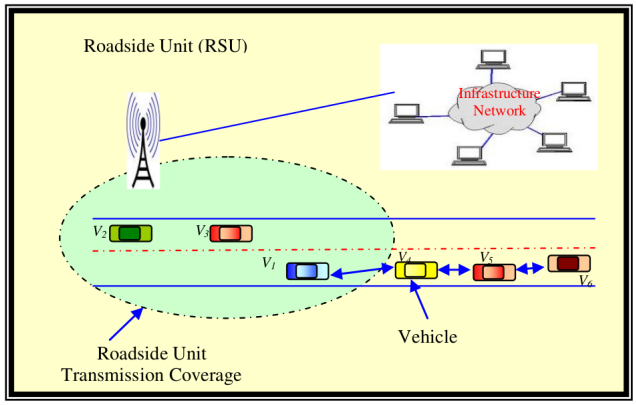

Figure 2.13: *VANET architecture [14].*

2.3.2 VANET applications

Use cases of VANET are divided into three main groups [14].

1. **Safety applications** One of the most important use cases of VANET is the safety applications which decrease the road accidents and save the lives of thousands of people. The shared information between V2V and V2I will be used to foresee the vehicle collisions and assists the drivers to have a convenient, fast and safe ride.

2. **Traffic monitoring and management applications** The main goal of this kind of application is increasing the performance of traffic assistance. The updated map information will help drivers to have better speed management and efficient navigation.

3. **Infotainment applications** The last VANET application offers entertainment and useful information for the drivers. These messages contain the information such as cinema, shopping centres, parking and fuel stations. These applications can be divided into 2 groups 1) co-operative local applications and 2) Global INTERNET Applications.

2.3.3 VANET characteristics

This section is written based on this paper: [14].

High dynamic topology

One of the main features of VANET is high dynamic topology. In VANET, there are a wide range of paths and vehicles which can be chosen. As vehicles move with considerable speed, the topology of the network can be very dynamic. Given two vehicles that move away from each other with the speed of 100 m/sec and the transmission range of 200m, then the link between them can be stable for: ((400 m)/(200 m * sec-1)) 2 seconds, which is very short.

Frequent disconnected Network

As mentioned in the previous part, the link between vehicles can change rapidly (for example in 2 seconds) and they need to establish a new connection. In an area with **low** vehicle density, such disconnections will happen frequently.

Mobility modelling and prediction

Predicting the movement patterns of vehicles is very challening in such a dynamic environment with high disconnection. The mobility pattern and the node prediction are really important for designing an effective system.

Communication environment

Each environment has its own features. the characteristics of a city's model differ form a highway's model. The city's model consists of various items, e.g., building, trees, different obstacles and important places (hospital and police stations), but the model of highway is very simpler than the model of city.

Adequate storage and energy

In Wireless Sensor Networks (WSN), energy management of nodes is a challenging issue, but this is not a problem in VANETs as the nodes are vehicle having enough storage for energy and processing units for computational processing.

Geographical type of communication

VANET networks apply the address of a geographical area as an additional type of communication in order to send the packets to the destinations. Generally, the type of data transmission which is applied in other networks is unicast or multicast, in which the end points are specified as ID or group ID.

Hard delay constraints

One of the VANET features which can been seen in **some** applications is hard delay constraints rather than high data rate demand. For instance, in an automatic highway system (due to the importance of safety), message information should be sent and received in a specific duration of time after a break happens. In such an application the **maximum** delay of transmission is a more important issue compared to the **average** delay.

2.3.4 VANET challenges

Security challenges

In any wireless communication, security is an important challenge. Security in VANET has got considerable attention in the past decades by researchers and industrialists. One of the most important issues in VANET is how to route information from sources to destinations. Designing an appropriate routing algorithm is a critical challenge for researchers. VANET systems can be attacked by several various kinds of attacks (such as authentication/identification, black hole, malware, DoS, broadcast, tampering, spamming, masquerading, replay, GPS spoofing and tunnelling). To sum up, the security issues should always be considered as an important part of designing and maintaining a VANET system [16].

Frequent link disconnections

I described the frequent link disconnection [14] features in the section "VANET characteristics". Keep in mind that this character counts as a **critical** challenge of VANET systems which should be studied accurately by researchers.

Socio-economic challenges

One of the most vital issues to expand a new technology is to encourage users to purchase it. In order to have efficient communication between vehicles in VANETS, the number of customers who have equipped their vehicles should be considerable. The **added value** for one user is directly related to the number of users that apply this technology. There are various ways to encourage (or in some cases enforce) customers to buy the essential equipments. For instance, the law can enforce drivers

to buy the essential VANET equipments or promote advertisements showing the benefits of using VANET such as entertainment or better navigation [71].

Highly Dynamic Spatio–Temporal Traffic Conditions

Vehicles can move from one environment to another. For instance, a vehicle can pass from a city environment with a lot of vehicles, obstacles, trees and buildings to a highway environment with a low density of vehicles. The environment features directly affect the performance of VANET systems. As illustrated in the above example, the speed in a highway environment is more than the speed in a city environment and this speed difference in two scenarios can be seen in the drop rate and link's stability of VANET's algorithm. To sum up, the designer of VANET system should consider the details of environments in order to design a networks which can satisfy the user requirements [14].

Information dissemination

In this part, I talk about the challenge of disseminating traffic data. "The traffic data is destined for the public interest, and not only for an individual" [14]; as a result, the data in VANET needs to be broadcasted while other networks just unicast the traffic data. This form of information dissemination can decrease the complexity of routing discovery process, address resolution and network topology management. Finally, various techniques of disseminating traffic data will be discussed in the next section. In other hand, information dissemination just formed on broadcasting can not satisfy the specific technical VANET requirement. Considering several different layers in the VANET stack is one of these challenges.

2.3.5 Data dissemination techniques in VANET

Before talking about the routing algorithm classification for VANET, it is important to explain various data dissemination strategies in VANET because a considerable number of problems can be resolved by efficiently disseminating data. Data dissemination is defined as the process of spreading information over distributed wireless network. The data dissemination relies on the various parameters such as the speed of vehicles, accelerations, pauses and the network size. Based on the current literature, information dissemination techniques can be classified in 3 groups: [13], [14].

Opportunistic data dissemination [14].

Data will be retrieved from vehicles or infrastructure when the objective vehicle encounters them.

Vehicle-assisted data dissemination

In this type of data dissemination strategy, the information is carried along with all of vehicles and the information will be delivered to other vehicles or infrastructure when they meet.

Cooperative data dissemination:

In this technique, part of information can be downloaded by the vehicles and this information will be shared later to get the data.

2.3.6 VANET routing algorithm classification

There are various VANET routing algorithm classification and one of this classification is shown in Figure 2.14. In this classification, the routing algorithm is divided into two groups: **topology based** routing algorithms and **geographic routing**.

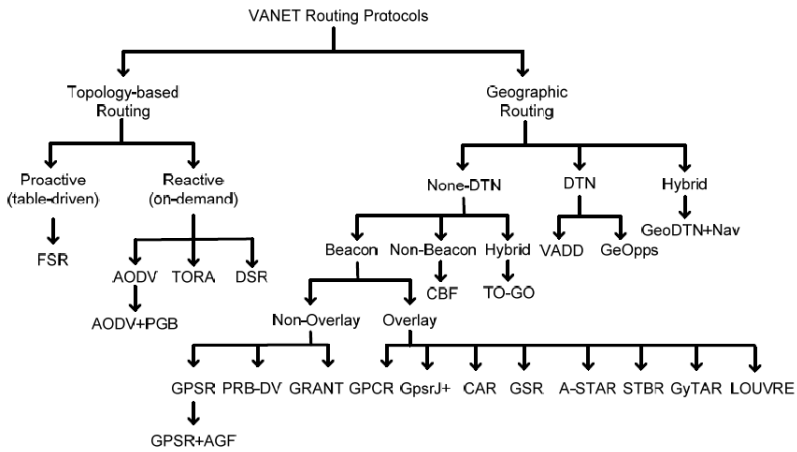

Figure 2.14: *VANET routing algorithm classification number 1 [17].*

In Figure 2.15, another routing algorithm of a VANET network is presented where the routing protocols can be divided as position based, cluster based, topology based, geocast based, and broadcast based.

2.4 AODV algorithm and fuzzy logic approach (related work)

In this section, I study the work that have been done for improving the routing algorithms in VANET by modifying AODV, Ad hoc On-Demand Multi-path Distance Vector (AOMDV) or by applying fuzzy logic technique. Considerable effort has been

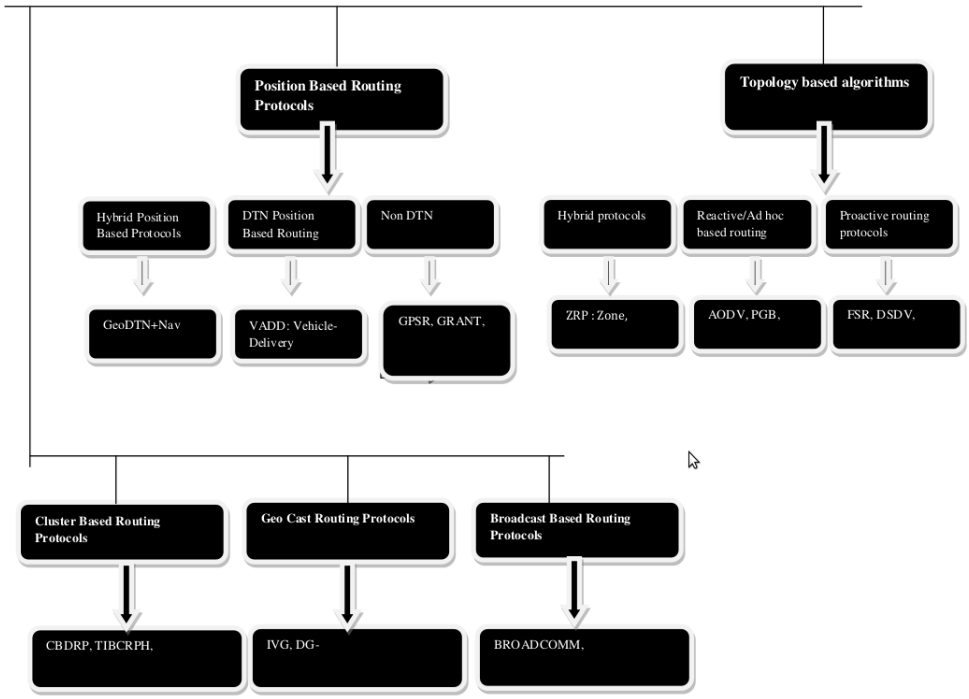

Figure 2.15: *VANET routing algorithm classification number 2 [18].*

done to improve forwarding algorithms by considering various QoS parameters such as end to end delay, throughput and overhead. Considering more mobility parameters and using different artificial intelligence techniques are my interests in this research. In the first part of this section I explain AODV/AOMDV improvements in relation to ITS/VANETs and in the second part I describe how fuzzy logic can be applied as a useful research approach.

2.4.1 AODV/AOMDV improvement

In this section, I first briefly refer to some work that has already been done in this scope. Afterwards, two important approaches are chosen to be studied in details. I propose my algorithm based on these two work.

There have been considerable work to improve AODV/AOMDV. The authors in [19] introduce the secure version of AODV (SAODV) which protects the routing messages of AODV. The digital signature and hash chains are applied to authenticate

non-mutable fields and the hop-count field, respectively in RREQ and RREP messages. Moreover, in [20] and [21] another protocols are proposed that are eliminating broadcasting in networks (for instance, Neighbor Elimination Scheme (NES)).

Mobility predication is another interesting area for researchers (see [22] and [23]). AOMDV is a multi-path extension of AODV protocol. In [24], a new routing protocol (S-AOMDV) is proposed that consists of hop and speed metrics in order to make decision. The simulation results of S-AOMDV confirms generally that it has better performance compared to AODV, and specifically it reduces the end to end delay by 11.92 % when there is a high load (8 packet/s).

In [25], Abedi proposed an enhanced version of AODV for VANET that two different mobility parameters have been used: position and direction. In this method, a node's direction is considered as the most important parameter in order to find the next intermediate hop; moreover, the node's position which has less importance compared to node's direction is selected as the second parameter. He assumed that these parameters will be obtained by GPS and the vehicles are moving based on Manhattan mobility pattern (it is described with details in Chapter 5).

When a node sends information to the destination node, the intermediate nodes will be selected based on the direction of the source and destination. Because of emplying Manhattan, Abedi assumed that 2 conditions can be performed in this protocol: first, the source node and destination are in same direction. Second, they move in opposite of each other. In other words, a node can be selected if it moves in the same direction with and/or destination and if its position is between source and destination.

His algorithm is explained using the pseudo code below:

Listing 2.1: *Abedi's algorithm number 1*

```
Bool
 Next_Hop (node, source, destination)
// Step1:
{
    Ds = Get_Direction (source);
    Dd = Get_Direction(destination);
    Dn = Get_Direction (node);
    If ((Dn == Ds) || (Dn == Dd))
    //Step 2:
     {
       Ps = Get_Position(source);
       Pd = Get_Position(destination);
       Pn = Get_Position(node);
```

```
      If ((Ps<=Pn<=Pd) || (Pd<=Pn<=Ps))
                  Return TRUE;
      Else
                 Return FALSE;
      }
    Else Return FALSE;
}
```

This protocol gets 3 inputs which are listed as: the candidate node for next hop, destination node and source node in order to determine the next hop.

But there is one problem, it is possible to not find any intermediate node with this criteria; therefore, Abedi changed his algorithm by adding a lower bound on the number of found routes and then dividing the protocol into two stages. (see Abedi's algorithm n.2)

Listing 2.2: *Abedi's algorithm number 2*

```
Attempt=0;
// Step 1:
If (((Dn==Ds) || (Dn==Dd))&& ((Pd<=Pn<=Ps) || (Ps<=Pn<=Pd))
    {
        Send_RREQ_packet(node);
        NR++;
    }
Else
Attempt++;
If (attempt<2)
   {
        Wait (w_t);
       // Go to Step 1;
   }
Else if (NR<Min_route_thershold)
    // Step 2:
If ((Dn==Ds) || (Dn == Dd))
    {
       Send_RREQ_packet (node);
       NR++;
    }
```

In Step 1, searching for the nodes which have both conditions is done. If the condition for lower bound routes is fulfilled, then the algorithm will be finished without doing anything. If the results did not satisfy the lower bound, then the

protocol will go to step 2 where the position condition will be removed and the nodes which have the same direction with the source and/or distension will be selected. At the end, Abedi evaluated his algorithm based on metrics such route length and broken links.

Hafez in [26] proposed a new algorithm called SD-AOMDV. The proposed algorithm improves the AOMDV based on VANET requirements and features which has been explained before. This algorithm considers these mobility parameters to choose a better intermediate node: speed and direction.

During the process of data transmission, the protocol will get the direction and speed of source and destination nodes, and based on these information an intermediate node will be chosen. In contrast to Abedi's work, Hafez considered speed as a new parameter to select a better next hop. To sum up, a node will be chosen based on two conditions: firstly, it has the same direction as source and/or destination, and secondly, it has the speed which is the closest to the average speed of source and destination.

One of the innovations of Hafez's work is finding the direction which is calculated based on vehicle's coordinations. Hafez considered 4 groups and assumed if the changes in coordination is positive for both X and Y (0°-90°), then it will be placed in group 1. If the change in the X axis is negative and changes in the Y axis is positive (90°-180°) it is set in group 2 and if both of them has negative changes it is placed in group 3 (180°-270°), otherwise it is in group 4 (270°-360°) (see Figure 2.16).

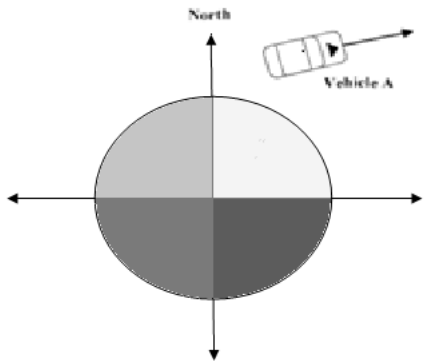

Figure 2.16: *Hafez's work, finding the direction [26]*

2.4.2 Fuzzy logic approach

In addition to normal algorithms, artificial intelligence (AI) has been used to optimize routing algorithms. Fuzzy logic has been applied widely in routing algorithms for wireless ad hoc networks. The author in [27] proposed an ant colony and fuzzy logic based algorithm to select the best paths. Moreover, in [28] a multi-path routing algorithm has been proposed based on fuzzy logic. The optimal path will be selected based on four inputs: energy consumption rate, queue occupancy rate, link stability and the number of intermediate node.

In [29] (note that I proposed my algorithm based on this work) researchers try to find the best route based on multi metric fuzzy logic. This algorithm is named Fuzzy Control Based AODV Routing Protocol (FCAR). Fuzzy logic system gets the life time of a route and the moving direction of vehicles, then it will choose the most stable routes.

The author in [29] explains that in reality road features limit the moving track of vehicles. Moreover, the vehicles that are moving in the same direction, can form more stable connections. This algorithm will calculate **the moving direction** between itself and last hop whenever it gets a route request message and if it is less than 20°, then the protocol counts two vehicles as the same direction. Finally, **the route life time** is defined as the shortest duration of time that two vehicles can communicate with each other (see [29] in order to see how mathematically it is calculated).

After designing the protocol based on fuzzy logic, the author used SUMO (Simulation of Urban MObility) which is a mobility generator to create nodes movements. At the end, he evaluated his algorithm based on these metrics: packet drop rate, average end-to-end delay, route discovery frequency and routing overhead.

Chapter 3
Algorithm description

3.1 Four-inputs Fuzzy-logic-based AODV Routing (FFAR) protocol

3.1.1 Introduction

Figure 3.1: *VANET network, real scenario [14]*

I choose AODV [32], [33], [34] as the routing algorithm and fuzzy logic as the artificial intelligence technique in order to improve forwarding messages in vehicular ad hoc system (see Figure 3.1 which shows a real VANET network's view) Moreover, as discussed earlier, the stability of selected routes is an important issue to forward

information appropriately and the main focus of my algorithm is to improve the forwarding message mechanism by having more stable routes.

3.1.2 Proposal questions

Why did I choose AODV? I chose AODV for the following reasons: AODV is one of the most famous protocols which got considerable attention by researchers and it has been implemented already by wide range of simulators, and the source codes and the documentation are available. Moreover, the simulation results have proved that AODV is superior (see [30], [31]) than the other existing algorithms (DSDV,DSR and TORA) based on these metrics: routing overhead, packet delivery ratio and path optimality.

Why did I choose Fuzzy logic? Fuzzy logic [35] can be used as a good decision making tool which is easy to understand. The underlying theory is easy and propositions in fuzzy logic [8], [9] can be formulated using the natural language which is an efficient way of communication. The other important aspect of fuzzy logic is its flexibility and the tolerance for imprecise data. Finally, fuzzy logic can model non-linear functions of arbitrary complexity.

3.1.3 FFAR procedure

I call this algorithm Four-inputs Fuzzy-Logic-based AODV Routing (FFAR), because it gets four parameters to evaluate the stability of a route: same directional vehicles, position of the nodes, relative velocity and the direction related to source and destitution. As mentioned before, the authors in [29] use a multi-metric fuzzy logic system to find the most stable routes based on the route life time and percentage of same-directional vehicles. **Based on this idea**, I expand and develop this procedure in order to find more accurate results. **My goal is to present a novel alternative algorithm that can improve AODV based on new fuzzy inputs.** To sum up, my algorithm is more accurate compared to FCAR because it considers more input parameters.

My fuzzy system considers more different parameters and it is not **blind** about the position and direction of source and destination. It means my algorithm selects an intermediate node based on **both** the situation of surrounding nodes and the situation of the source and destination nodes. As an example, Figure 3.2 shows that although vehicle 1 and 2 have the same direction, but if vehicle 1 wants to send data to vehicle 4, vehicle 3 is a better choice compared to vehicle 2 (because of its position which is between the source and the destination; therefore, it has better position compared to vehicle 2). By having the various parameters, I have this opportunity to consider more details and find out **the most stable** routes. In FFAR, when a

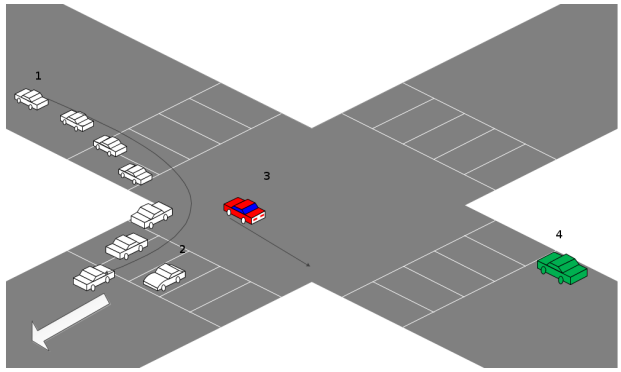

Figure 3.2: *Example scenario*

vehicle needs to set up a connection to a destination, first it will broadcast a route request (RREQ) message. I modified the RREQ message so that it contains the node's speed, position and direction. When an intermediate vehicle receives the RREQ message, it will call the stability function. The stability function will get four inputs (as it will be explained later) and will give an output (see Figure 3.3 and pseudo code of FFAR).

Listing 3.1: *FFAR algorithm*

```
1:(step 1) receive RREQ
2: calculate direction of node
3: calculate velocity of node
4: calculate position of node
5: calculate angle of node
6: if same RREQ then
7:        comparing with data of route table
8:        if fewer hops or bigger stability then
9:                refresh inverse route
10:                    if current vehicle is a destination then
11:                          send RREP
12:                    else if has a fresher route to a destination then
13:                          send RREP
14:                    else
15:                          broadcast RREQ (go to step 1)
16:                    end if
17:      else
18:                discard RREQ
19:      end if
20: else
21:        comparing with data of route table
```

```
22:        if fewer hops or bigger stability or bigger sequence number then
23:               refresh inverse route
24:                    if current vehicle is a destination then
25:                           send RREP
26:                    else if has a fresher route to a destination then
27:                           send RREP
28:                    else
29:                           broadcast RREQ (go to step 1)
30:                    end if
31:        else
32:                    if current vehicle is a destination then
33:                           send RREP
34:                    else if has a fresher route to a destination then
35:                           send RREP
36:                    else
37:                           broadcast RREQ (go to step 1)
38:                    end if
39:        end if
40:end if
 /*
```

If a node receives an RREQ message with the same sequence number, then it compares the hop number and the route stability output. If it has smaller hop number or bigger stability, it will update the reverse path to the source. Afterwards, it checks if it is the destination. if "Yes", it sends an RREP message, otherwise it checks if it has a fresher route to the destination. If "Yes", it sends an RREP message, otherwise it will broadcast an RREQ message. If it receives an RREQ message with the same sequence number and it does not have bigger route stability or less hop, then it discards the RREQ message.

If a node receives an RREQ message with a different sequence number, then if it has bigger sequence number or bigger route stability output or less hop count, then it will update the reverse route to the source. In continue, it checks to see if it is the destination or not. if "Yes", it sends an RREP message, otherwise it checks to see that if it has a fresher route to the destination or not. If "Yes", it sends an RREP message otherwise, it will broadcast the RREQ message.

Finally, if a node receives an RREQ message with a smaller sequence number and it does not have bigger route stability output or a smaller hop count, then the reverse path will not be updated. After that, it will check to see if it is the destination or not. if "Yes", it sends an RREP message otherwise, it checks to see that if it has a fresher route to the destination or not. If "Yes", it sends an RREP message otherwise, it will broadcast the RREQ message.

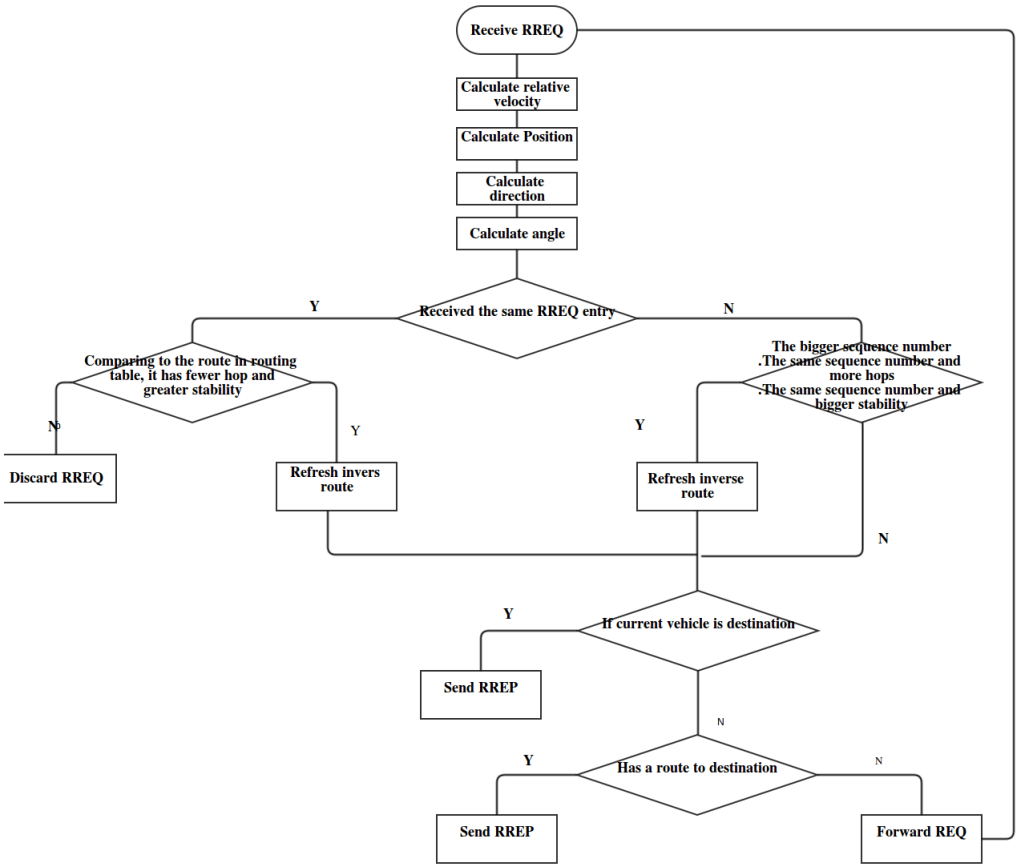

Figure 3.3: *FFAR algorithm flowchart [29].*

Figure 3.3 explains if a node received a RREQ message, it asks the fuzzy control system to determine stability of the route. Based on the output, I can decide to choose the most stable route.

In the regular routing algorithm, routing table will be updated **only** when there is a newer sequence number or smaller hops, but as explained before, VANET has its own characteristics, such as high speed and frequently disconnection, and the decision based on these parameters cannot fulfil required expectation; therefore, I need newer parameters which will be described in the next section.

3.1.4 Selecting route metrics

Same direction of two intermediate nodes

As shown in Figure 3.4.1, when an intermediate node receives an RREQ message from its previous node, it will create a reverse path. If the two intermediate nodes are in the same direction, the probability of having a stable route will be increased. As illustrated in Figure 3.4.1, node 2 has the same direction to the red node (receiver intermediate node) and it can increase stability of the route much more, compared to node 1 and node 3.

Consider the moving direction of node 1 is (dx1, dy1) and the moving direction of node 2 is (dx 2, dy2), where dx and dy represent the direction vector component on X and Y axis, respectively. The angle between two nodes can be calculated as: [29]

$$Angle = \arccos \frac{dx1 * dx2 + dy1 * dy2}{\sqrt[2]{dx1^2 + dy1^2} * \sqrt[2]{dx2^2 + dy2^2}} \tag{3.1}$$

When a vehicle gets an RREQ message, it will calculate the included angle of moving direction between itself and the last hop. Table 3.1 shows if the angle is less than 20°, then I call it good.

Bad	Good
$10° <= \|Angle\| < 180°$	$\|Angle\| < 20°$

Table 3.1: *Same directional of two intermediate nodes*

Intermediate node's position

One of the most important parameters to choose and weight a route is the position of intermediate nodes in the route. I call a position "good" if the distance (it is named "D" in Table 3.2) of the intermediate node to the line passing through the source and the destination node is less than 100cm. If the distance is between 1m and 3m, it is normal, and if it is between 2m and 4m, I call it bad (see Table 3.2). Figure 3.4.2 presents how node 2 has a good position and choosing node 2 will increase the stability of the route compared to nodes 1, 3 or 4.

Bad	Normal	Good
$2m < D < 4m$	$1m < D < 3m$	$D < 100cm$

Table 3.2: *Intermediate node's position*

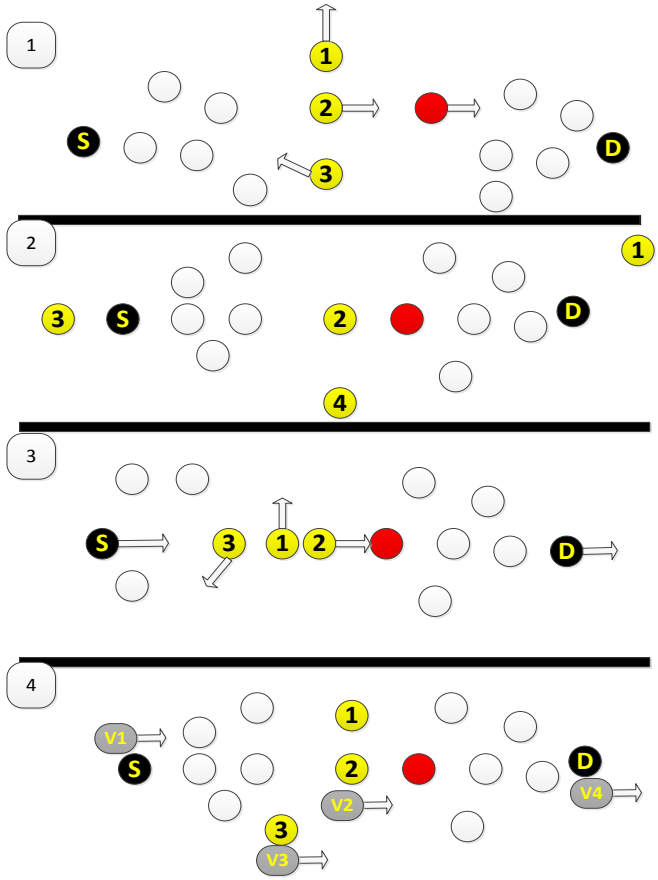

Figure 3.4: *Selecting route metrics: 1-Same direction of two intermediate nodes 2- intermediate node's position 3-same directional node 4- relative velocity*

Same direction to source and destination

I considered the same direction compared to the source and destination as the other important parameter, see Figure 3.4.3. In such a scenario node 2 will be selected compared to node 1 and node 3. Table 3 shows if the sum angle between intermediate node's vector and source interminable node destination is less than 20°, I call it good, otherwise bad.

Bad	Good
$10° <= \|Angle\| < 180°$	$\|Angle\| < 20°$

Table 3.3: *Same direction to source and destination*

Relative velocity

Velocity has considerable effect on the route life time; therefore, I used relative velocity as an important input for my fuzzy logic system. If Vs and Vd show the speed of the source and destination nodes, respectively, and Vi shows the speed of intermediate node, then I call a node good if the sum of the difference compared to the source and destination is less than 30 m/s (see Table 3.4). Figure 3.4.4 shows the relative velocity parameter.

Bad	Good
$20m/s <= \|Vd - Vi\| +$ $\|Vs - Vi\| <= 90m/s$	$0m/s < \|Vd - Vi\| +$ $\|Vs - Vi\| < 30m/s$

Table 3.4: *Relative velocity*

3.1.5 Modification of control message and routing table

RREQ's modification

The below Table presents the added data to RREQ: [29]

Added data to RREQ
Moving direction vector component on X axis
Moving direction vector component on Y axis
X-coordinate of location
Y-coordinate of location
speed vector component on X axis
speed vector component on Y axis

Table 3.5: *Modify of RREQ*

The added information to "send request function".

Listing 3.2: *Modified send request function*

```
 void
AODV::sendRequest(nsaddr_t dst) {
 iNode = (MobileNode *) (Node::get_node_by_address (index) );
 rq->xpos = iNode->X(); // x positon
 rq->ypos = iNode->Y(); // y positon
 rq->zpos = iNode->Z(); // z position
 rq->dx=iNode->dX();  // dx
 rq->dy=iNode->dY();  // dy
 rq->v=iNodeDst->speed();//speed
```

The added information to "route request packet".

Listing 3.3: *Modified route request packet*

```
    struct hdr_aodv_request {
        u_int8_t      rq_type;      // Packet Type
        u_int8_t      reserved[2];
        u_int8_t      rq_hop_count; // Hop Count
        u_int32_t     rq_bcast_id;  // Broadcast ID
        nsaddr_t      rq_dst;       // Destination IP Address
        u_int32_t     rq_dst_seqno; // Destination Sequence Number
        nsaddr_t      rq_src;       // Source IP Address
        u_int32_t     rq_src_seqno; // Source Sequence Number
        double        rq_timestamp; // when REQUEST sent;
        double xpos ;               //x position
        double ypos;                //y position
        double zpos ;               //z position
        double dx;                  // dx
        double dy;                  // dy
        double V;                   //   speed
```

RREP's modification

After calling stability function (fuzzy logic system), the output will be set in RREP message and it will be used to find the most stable routes.

Listing 3.4: *RREP's modification*

```
struct hdr_aodv_reply {
        u_int8_t      rp_type;      // Packet Type
        u_int8_t      reserved[2];
        u_int8_t      rp_hop_count;        // Hop Count
        nsaddr_t      rp_dst;              // Destination IP Address
        u_int32_t     rp_dst_seqno;        // Destination Sequence Number
```

```
        nsaddr_t        rp_src;             // Source IP Address
        double          rp_lifetime;        // Lifetime

        double          rp_timestamp;       // when corresponding REQ sent;
                                            // used to compute route
                                                discovery latency
        int stability                        // stability parameter;
```

Routing table's modification

The routing table will be updated like an RREP message and **only** the stability parameter in the routing table will be updated.

Listing 3.5: *Routing table's modification*

```
class aodv_rt_entry {
        friend class aodv_rtable;
        friend class AODV;
        friend class LocalRepairTimer;
 public:
        aodv_rt_entry();
        ~aodv_rt_entry();

        void            nb_insert(nsaddr_t id);
        AODV_Neighbor* nb_lookup(nsaddr_t id);

        void            pc_insert(nsaddr_t id);
        AODV_Precursor* pc_lookup(nsaddr_t id);
        void            pc_delete(nsaddr_t id);
        void            pc_delete(void);
        bool            pc_empty(void);

        double          rt_req_timeout;       //
        u_int8_t        rt_req_cnt;           //
            int        stability;   // stability paramater
```

3.1.6 Stability function

This section is written based on these papers: [8], [9].

The concept of fuzzy logic is explained in the Chapter 2. In this section, I apply fuzzy logic to find the most stable routes (*in some parts, I forced to rewrite or address to the defined previous concepts and it is because of splitting the fuzzy logic section in two parts*)

A fuzzy logic system is shown in Figure 2.12. A fuzzy logic system (FLS) can be defined as a non-linear mapping of an input data set to a scalar output data. Each FLS has four key parts: [8], [9].

fuzzifier, rules, inference engine, and defuzzifier.

The first step is called **fuzzification** [8] and the gathered crisp data are converted to a fuzzy set by using a fuzzy linguistic variable, fuzzy linguistic terms and membership functions. In order to map non-fuzzy input values to fuzzy linguistic [9] terms and contrariwise, **membership function** will be applied. There are various types of membership functions such as triangular, Gaussian, trape-zoidal, piecewise linear or singleton. I used the **triangular** form in this research. Figures 3.5, 3.6, 3.7, 3.8 and 3.9 show the angle, direction, velocity, position and stability membership function, respectively.

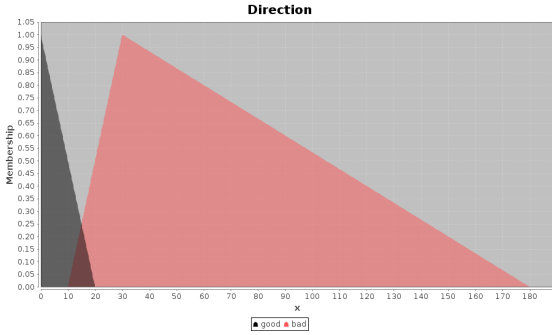

Figure 3.5: *Direction membership function*

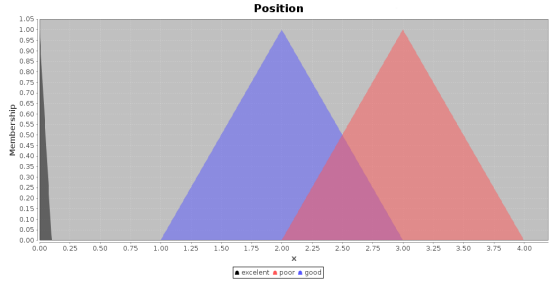

Figure 3.6: *Position membership function*

The quality of fuzzy logic approximation depends on the quality of fuzzy rules. I designed the fuzzy logic system to calculate the stability of each link between source

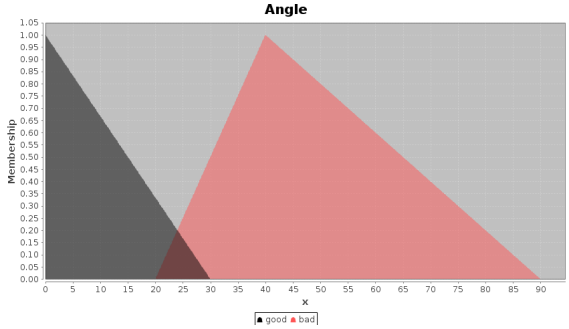

Figure 3.7: *Angle membership function*

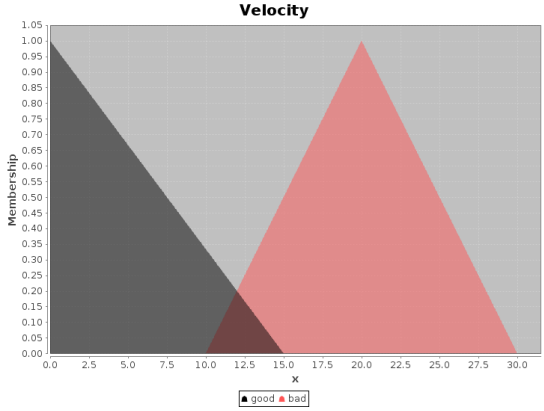

Figure 3.8: *Velocity membership function*

and destination. This fuzzy system has four inputs that are named with numbers 1, 2, 3 and 4 in Table 3.6.

1. Intermediate node's position
2. Same direction of two intermediate nodes
3. Relative velocity
4. Same direction to source and destination

The inputs are **fuzzified**, **implicated**, **aggregated** and **defuzzified** [8], [9] to get the link stability as the output. The linguistic variables which are associated with the output variables are **too bad**, **very bad**, **bad**, **normal, good, very good,**

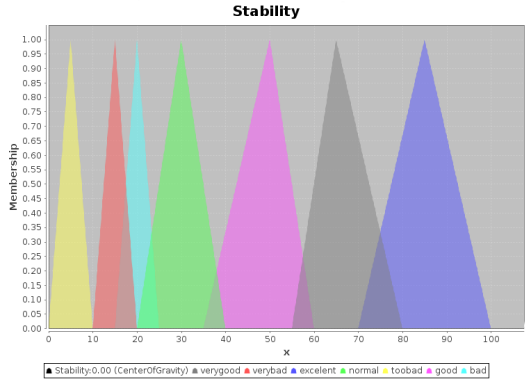

Figure 3.9: *Stability membership function*

excellent. Input variables are explained earlier. The important part of any fuzzy logic system is how to design **rules**. A rule is actually a simple "IF-THEN" with a condition and a conclusion. My four inputs assigned as antecedent of the conditional statement and the result of the conditional is the output. I just used the "**AND**" operation to build the rules. For example,
RULE 1 : IF Position IS excellent AND Direction IS good AND Velocity IS good AND Angle Is good THEN Stability IS excellent. Table 3.6 shows the fuzzy rules which is applied in this report.

Fuzzy logic

Link situation	1	2	3	4
Excellent	G	G	G	G
Excellent	G	G	G	B
Excellent	G	G	B	G
Excellent	G	G	B	B
Good	G	B	G	G
Good	G	B	G	B
Good	G	B	B	G
Good	G	B	b	B
Very good	N	G	G	G
Very good	N	G	G	B
Good	N	G	B	G
Normal	N	G	B	B
Normal	N	B	G	G
Normal	N	B	G	B
Bad	N	B	B	G
Bad	N	B	b	B
Very bad	B	G	G	G
Very bad	B	G	G	B
Very bad	B	G	B	G
Very bad	B	G	B	B
Very bad	B	B	G	G
Too bad	B	B	G	B
Too bad	B	B	B	G
Too bad	B	B	B	B

Table 3.6: *Fuzzy logic rules' table*

In order to evaluate the fuzzy rules and the combination of results of the individual rules, fuzzy set operation [8], [9] will be applied. After evaluation of the rules, the results will be combined to get the final answer (the output is actually the stability parameter and will passed to RREP function). This process is named inference. For combining the results, different methods exist, I used "**maximum**" algorithm in this research.

The final step is to get a non-fuzzy value (crisp output) which is named **defuzzification**. Defuzzification [8] is the process of interpreting the membership degrees of a fuzzy set into a specific decision or real value. There are various algorithms for defuzzification and I used **center of gravity** in this research. Appendix B explains the fuzzy logic library I have designed in this thesis, and appendix C illustrates how to calculate 4 fuzzy inputs.

Chapter 4
Simulation tools

4.1 Network simulator 2 (NS2)

A network simulator [36], [37] is a software or hardware which can foresee the performance, behaviour and actions of a network in various situations while there is no actual network [36]. Generally, users can configure the simulator based on their requirement to reach their goals. There are various network simulators which support the most popular protocols like TCP and AODV and various types of networks such as wireless sensor network and Wi-Max.

In the field of computer networks, **network simulation** [36] refers to the technique by which the behaviour of a network is modelled by measuring the interaction between various network entities such as routers, switches and data links or obtaining and playing back the events from a production network. The behaviour of a network and different applications under different conditions can be observed and tested by changing network parameters. For example, to see the behaviour of a network in high congestion area I decrease the arrival time or increase the movement speed of mobile nodes. Finally, in the validation process, the results obtained by the simulation will be compared to the results from mathematical models.

There are many different reasons to apply network simulators compared to set-up an entire testing enviroment consisting of networked computers, switches, routers and information links [36]. Two of the main reasons are: simulators are significantly **cheap and fast**. To study the scenarios which takes considerable time and resources, the simulators will allow researchers to have a under control and reproducible setup. In general, network simulators give the user the opportunity to build a network topology by adding nodes and links between them. One of this simulators is called network simulator 2 (NS2).

NS2 [38] is an object-oriented and discrete event simulator which is developed and designed at Berkeley university. The core of NS2 is written in C++ with an

Object Tcl (OTcl) interpreter as a command and configuration interface. The reason for this technology stack is that C++ is fast for running programs, but it is slow for modifying the configurations. On the other hand, Otcl can be modified quickly while it is slow to run. This approach is called **split-language programming** that allows to easily run and modify large scenarios. On other hands NS2 requires a deep understanding of both languages and compiling, running and debugging at the same time [7].

NS2 has a large number of built-in C++ classes that is used to set up a simulation via Tcl script. To make your own protocol you need to create your own C++ classes. As shown in Figure 4.1, NS2 gives a text based output which can be interpreted graphically and interactively by a network animator (NAM). NAM environment is presented in Figure 4.2 [7].

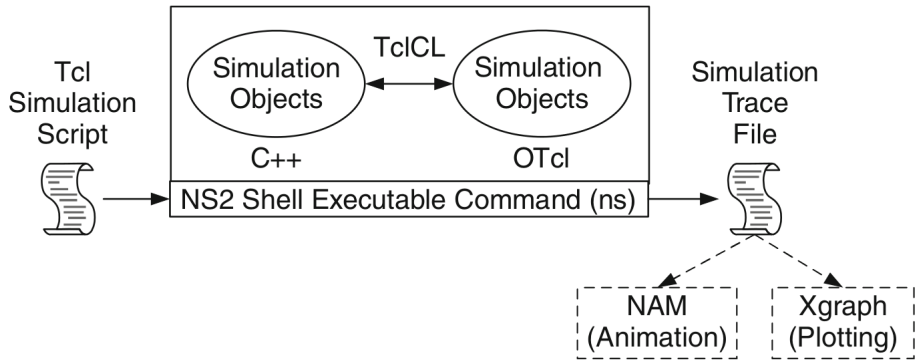

Figure 4.1: *NS2 architecture [7].*

NAM is an effective visualization tool to animate the network simulation traces and it gives this opportunity to users to observe packet movements, packet loss and the topology of network (nodes and link between them) in a real world [7].

4.1.1 Why did I choose NS2?

In conclusion, I chose NS2 as the simulator for this project because it is an **open source software** and its code can be modified. Moreover, it has **considerable libraries** that cover various areas such as different routing algorithms, different layers and mobility. There are many various versions of NS and I applied the last version which is ns-2.35 [7].

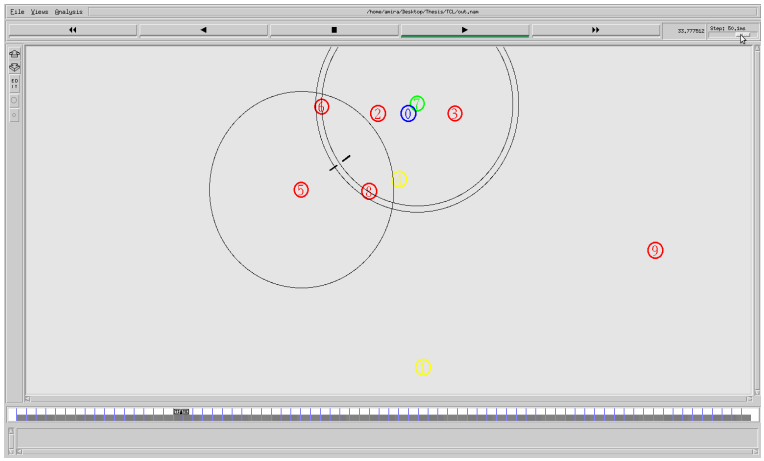

Figure 4.2: *NS2 network animator (NAM)*

4.1.2 AWK

AWK [42] is an interpreted programming language that I used in order to extract the required data from NS2 trace file (.tr). Appendix A presents the AWK files which are used in this thesis. To run the AWK script in Linux, this command should be run: **"awk –f filename.awk filename.tr"**

4.2 Mobility model

Mobility models [43], [47] show the motion of mobile nodes and how their position and velocity modify. There are different mobility models such as: random walk model, and random Gauss-Markov model.

4.2.1 Mobility generator

In order to generate random movement, I used BonnMotion. "BonnMotion is a Java software which creates and analyzes mobility scenarios and is most commonly used as a tool for the investigation of mobile ad hoc network characteristics. The scenarios can also be exported for several network simulators, such as ns-2, ns-3, GloMoSim/QualNet, COOJA, MiXiM, and ONE" [44]. BonnMotion will be run by this command:
bm -f scenario1 Manhattan -n 100 -x 900 -y 3600 which the -n refers to number of nodes and -x and -y refer to width and hight respectively. Finally, I used the jFuzzyLogic library [45], [46] in Java in order to create a Fuzzy logic system (see Appendix E which shows the mobility scenario made by Bonnmotion).

Chapter 5
Evaluation

After defining the problem, introducing the essential background and proposing the new algorithm, in this section I run various simulation scenarios based on the tools that have been described earlier. This chapter presents how the simulation characteristics are set and what results are obtained.

The simulations were conducted on a computer with these specifications: an Intel core i5 processor at 2.50 GHz, 8 GB of RAM running Linux Ubuntu 13.10.

5.1 Simulation set-up

Features of the simulation scenario, movement model, communication model and simulation parameters are described in this section.

5.1.1 Scenario

My studied scenario consists of 10 mobile nodes, and at each time 5 mobile nodes are added to the next scenario (the nodes are increasing between 10-50) and my topology is a square with 2000 * 2000 m$\hat{2}$. 1000 seconds is considered as the simulation time. For each scenario, the number of source and destination nodes are random. In addition, sources and destinations are distributed stochastically among all of mobile nodes (see appendix D). It should be mentioned that I assume that all of links during simulation are bi-directional. Figure 5.1 shows how the mobile nodes are distributed in a scenario with 25 vehicles. Moreover, error bars are applied in this report and I repeated the simulation for 10 times for each value by applying **random mobility model** for each one.

Error bars are applied to show the variability of data graphically. In other words, error bars shows the accuracy of data or the amount of deviation value from the average or expected value.

Usually error bars offer the amount of deviation from the average data or expected data by presenting standard deviation. A low standard deviation means the data are closed to the average value and high standard presents that the data are far from the expected value.

A confidence interval refers to the precise range of measurement. 95% confidence interval is applied in this research and it means that there is just 5% chance of being wrong.

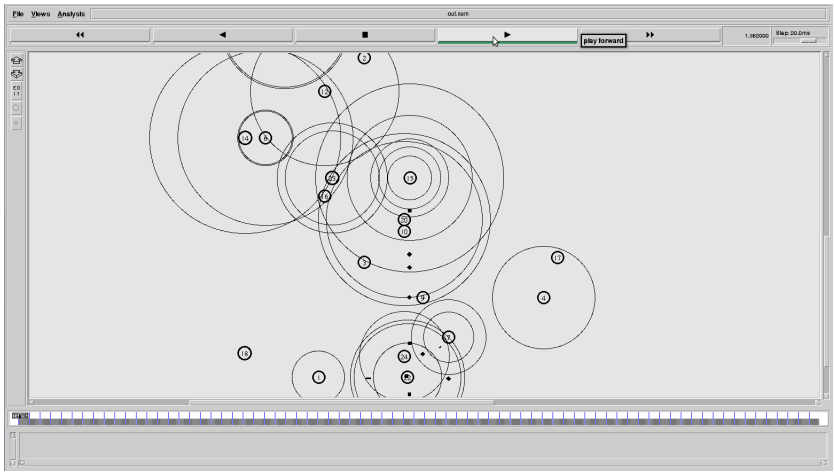

Figure 5.1: *Scenario visualization in NAM*

5.1.2 Movement model

Manhattan is one of the most important mobility models of VANET. I chose Manhattan for simulating different scenarios in this thesis. In this model, there are many horizontal and vertical streets present in the simulation field and vehicles move in lanes of the streets. The street can be one directional or bi-directional; moreover, the vehicles can go to left, right, up and down. The streets can cross each other and given one direction has positive value 1, the negative value is assigned to the opposite one. The Manhattan model is shown in Figure 5.2 [25]. In my mobility model, a random value for speed which is constrained between the optional minimum and maximum values is allocated to each mobile node. **-M** and **-h** commands show the minimum and maximum speed, respectively in BonnMotion. To see how the nodes are randomly distributed in the map and how their speeds are different from each other see appendix E.

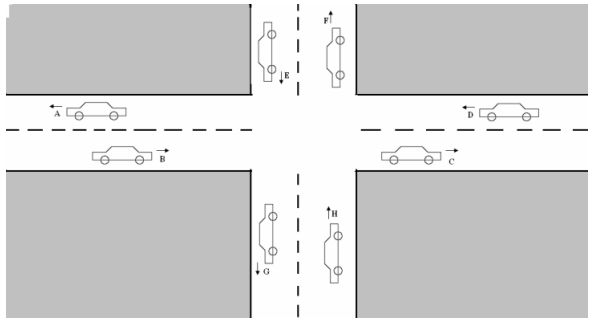

Figure 5.2: *Manhattan model [25]*

5.1.3 Communication model

In this research, I target the effects of mobility parameters in VANET networks based on various parameters; therefore, the **size** of a packet and the packet's **inter arrival time** are selected as constant variables (deterministic functions).

Listing 5.1: *Time arrival and packet size*

```
set udp1 [$ns_ create-connection UDP $node_(5) LossMonitor $node_(0) 0]
 $udp1 set fid_ 1
 set cbr1 [$udp1 attach-app Traffic/CBR]
 $cbr1 set packetSize_ 1000
 $cbr1 set interval_ .07
 $ns_ at 0.0 "$cbr1 start"
 $ns_ at 1000.0 "$cbr1 stop"
```

As shown in the above code, the inter arrival time for the source mobile nodes is constant and it starts at 0 seconds and will finish at 1000 seconds. Each source mobile node generates its packets every 0.7 seconds in this research; in other words, 10 packets per 7 seconds and each packet size is 1000 bytes. Therefore, the amount of generated data per second is (10) / (7) * (1000) * (8) = 1142.85 bits.

Note that TCP is not applied in this report because I did not want to investigate the effect of TCP on the flow control and re-transmission.

5.1.4 Parameters

The commonly applied parameters for simulation in this thesis are shown in Table 5.1.

Parameter	Value
Application layer	UDP
Traffic type	CBR
Routing protocol	AODV / FFAR
MAC band width	54 Mbit/s
MAC	802.11
Frequency	2.4 GHz
Transmitting power	17 dbm
Antenna type	Omni-directional
Size of simulation scene	2000 * 2000 (m * m)
Simulation time	1000 s
Packet rate	0.7
Transmission range	250 m

Table 5.1: *simulation parameter*

5.2 Performance metrics

The definition of network performance metrics are written based on these papers: [26], [29], [40], [41], [48], [49].

To evaluate the simulation results five performance parameters are taken into account.

Packet delivery ratio

The number of received packets compared to the number of generated packets is defined as the packet delivery ratio. This parameter determines how the protocol's performance in term of delivery of packets, while speed, acceleration or payloads are variables. This property can be defined as:

$$\text{Packet delivery ratio } \frac{\textbf{received packets}}{\textbf{generated packets}}$$

Note that packet drop ratio can be calculated as: 1 - (packet delivery ratio).

Route discovery frequency

A source node will start the route discovery process by broadcasting a probe packet (in AODV, it is called route request message or RREQ message) to all of its neighbours. In Chapter 2, it has been explained how a route can be established in

AODV. After establishing the route, the packets are transmitted to the destination. Therefore, the intermediate nodes do not need to maintain the routing tables any more. This will decrease network overhead. As mentioned before, VANET networks have unstable connections because of high mobility which causes considerable overhead for networks to establish new routes. As a result, the frequency of route discovery procedure **should** be reduced as much as possible. Keep in mind, generally, the routing discovery approach takes considerable network bandwidth. Formally, route discovery frequency can be defined as:

$$\text{Route discovery frequency: } \frac{\textbf{route discovery time initiated by route}}{\textbf{simulation time}}$$

End to end delay

It refers to the entire time that is spent for sending a packet across a network from a source to a destination node. This time includes transmission delay, processing delay and propagation delay.

Network overhead

Overhead should be considered as a significant issue in design and implemention of a network. Briefly, the network protocol overhead can be described as follows: network protocols such as TCP need sending control and signalling data in order to manage the data transfer over the network; as a result, the maximum bandwidth of the network cannot be used to send **actual** data. Moreover, it causes more transmitting time and requires more computational processing.

Network throughput

Throughput or network throughput in computer networks is defined as the average rate of successful messages delivery over a communication channel, which will be measured by bits per second (bit/s or bps) and data packet per time slot or data packets per second. The queue theory is widely applied to analyse network throughput mathematically where the data can be sent through a physical link or a network node.

5.3 Simulation results

After introducing the network parameters, this section presents and discusses the results of simulation scenarios.

Routing discovery frequency

Figure 5.3 and 5.4 show route discovery frequency when the network size is varying from 10 to 50. As shown, the route discovery frequency is getting higher when the number of nodes increases because when there are more nodes, the probability of link disconnection is getting higher. Therefore, sources initiate new route requests messages which increase the route discovery frequency. Figures 5.5 and 5.6 present the comparison of FFAR and normal AODV in one figure. Based on my expectation, Figure 5.5 indicates when the number of nodes increases, the route discovery frequency of FFAR is getting lower than AODV. The routes that are set up in FFAR are more stable in contrast to normal AODV, because FFAR searches for more stable routes based on the new mobility metrics. In conclusion, AODV needs to initiate more route request messages in order to find new routes. This causes higher route discovery frequency compared to FFAR.

In Figure 5.6, each data point is the average value of running the simulation scenarios for 10 times with different seeds.

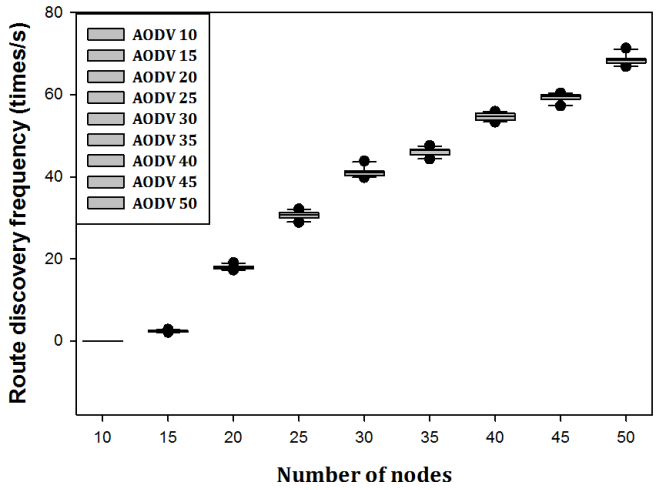

Figure 5.3: *Route discovery frequency vs. Network size for AODV by applying error bars.*

Drop rate

As shown in Figure 5.7 and 5.8, packet drop ratio is increasing when the size of the network is increased from 10 to 50. As Figure 5.7 and 5.8 present, packet drop ratio is relatively high (above 70%) for all of the scenarios. There is no **specific** reason

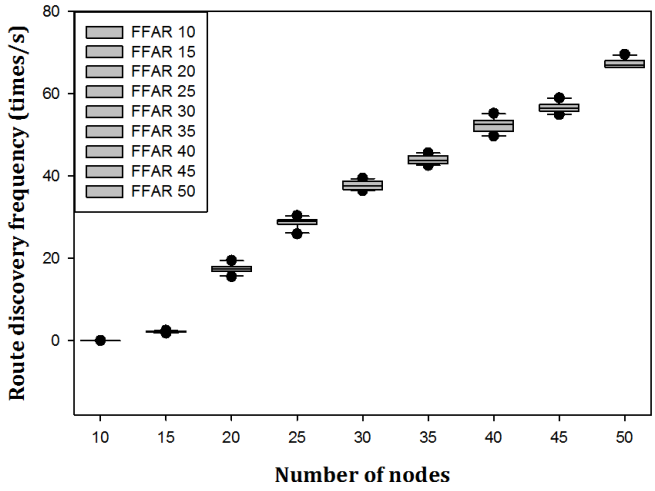

Figure 5.4: *Route discovery frequency vs. the network size for FFAR by applying error bars.*

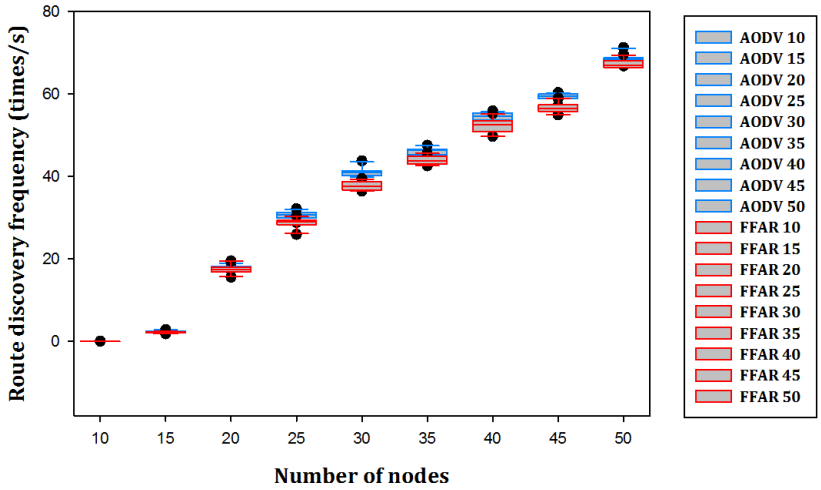

Figure 5.5: *Comparing the route discovery frequency of AODV to FFAR when the network size is increased (by using error bars).*

for this event and it was completely stochastic. For instance, if the distance between

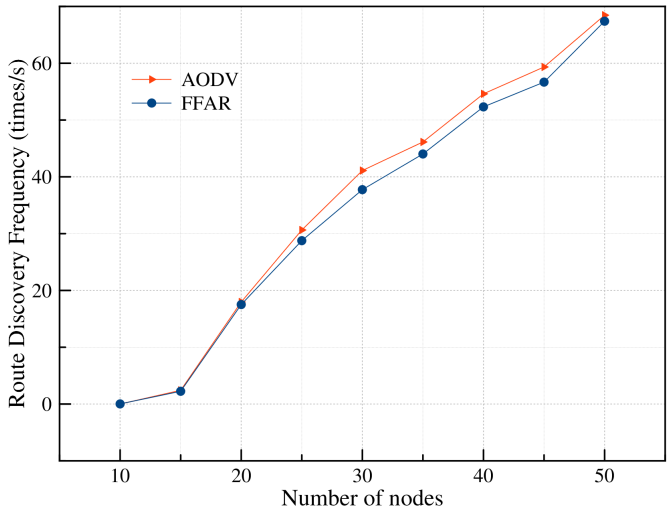

Figure 5.6: *Comparing the route discovery frequency of AODV to FFAR when the network size is increased (by using average value).*

source and destination nodes was short, then less drop rate would be observed. It should be mentioned that this high drop rate directly is related to the designed mobility model with a several nodes and connections. As shown in Figure 5.7 and 5.8, drop rate increases when the number of nodes are increasing, because by having more nodes and a bigger network topology, there will be longer routes (in number of hops). Consequently, it takes more time for sources to send an RREQ and receive an RERR; as a result, it causes more dropped packets and higher link's disconnections.

Figure 5.9 explains the comparison between two curves in one figure. It is clear when the number of nodes is increasing, the drop rate in FFAR is less than normal AODV. AODV chooses a route **just** based on newer sequence number and less hops (it does not care about route's firmness); therefore, there will be some unstable intermediate nodes among the selected nodes. The route that contains these nodes breaks easily which causes more drop rate in contrast to FFAR which is seeking for more stable routes. Note that although there are slight difference between two algorithms, but when there are millions of packets, it is a considerable improvement. Finally, I have used error bar in Figure 5.9 and in Figure 5.10 the data points are the average value of simulations results.

As shown in Figure 5.10 there is an unexpected event where drop rate for the network for 40 nodes is less than the network with 35 nodes (in FFAR). This is because

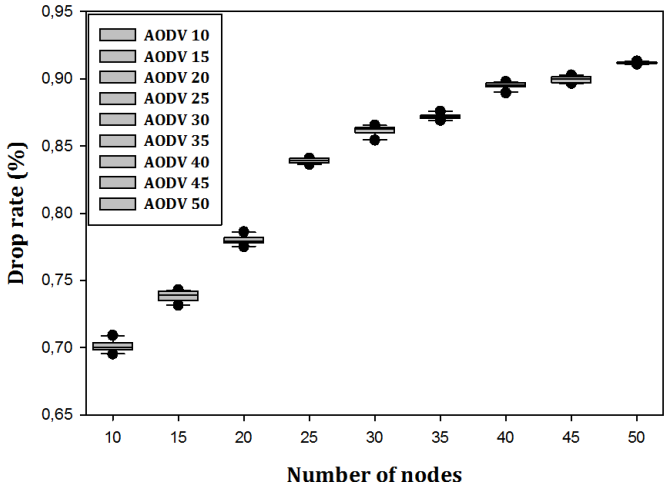

Figure 5.7: *Drop rate vs. the network size for AODV by applying error bars.*

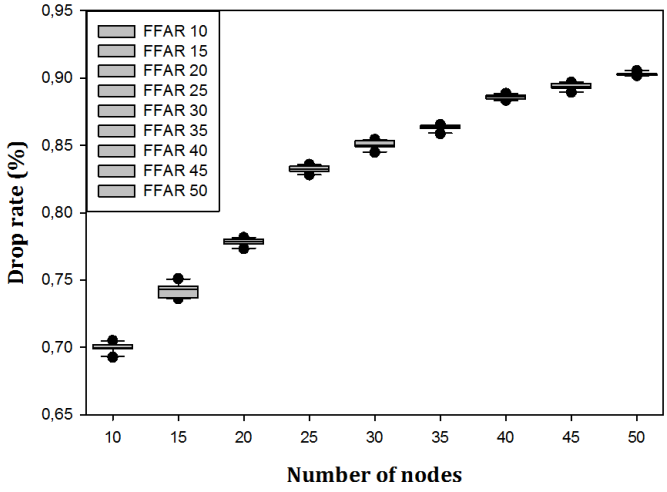

Figure 5.8: *Drop rate vs. the network size for FFAR by applying error bars.*

the nodes and sources are distributed randomly and there is a small probability that the drop rate is not increased compared to the previous scenario which has less nodes. I refer the reader to this paper [29] which is showing the results are relying on the

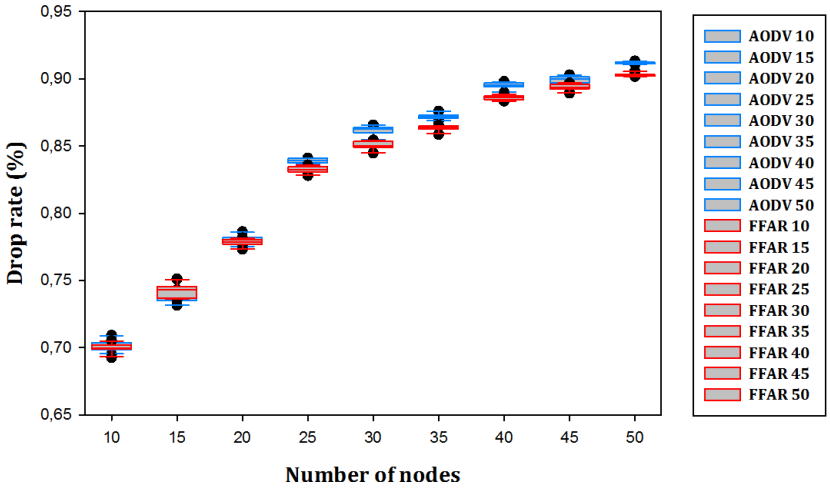

Figure 5.9: *Comparing the drop rate of AODV to FFAR when the network size is increased (by using error bars).*

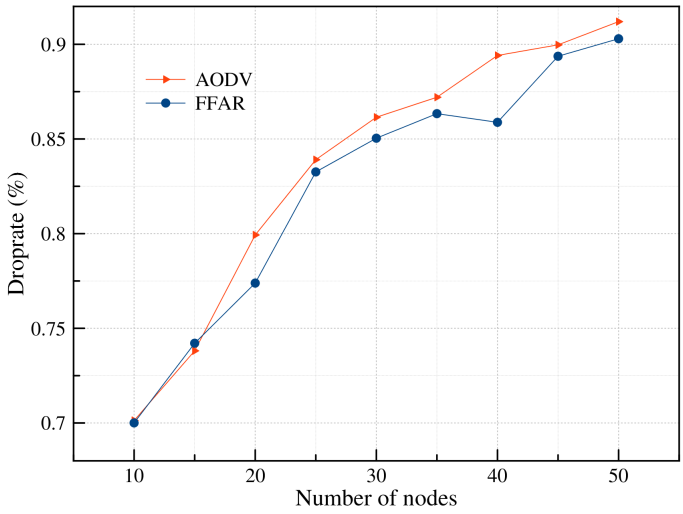

Figure 5.10: *Comparing the drop rate of AODV to FFAR when the network size is increased (by using average value).*

scenarios situations dramatically. At the end, it should be mentioned that the **main** point of Figure 5.9 and 5.10 is showing that FFAR is superior than AODV in terms of drop rate (even when the node number is 40).

Network overhead

Control data messages are used to establish new routes in forwarding routing algorithms. Studying the amount of control information is an interesting issue for researchers. As explained earlier, this control messages cause overhead and consequently, overhead uses the bandwidth. Therefore, the **actual** data which should be sent cannot use the maximum bandwidth. Moreover, the overhead is very **expensive** in terms of transmission time, memory usage and commuting processing. Figure 5.11 and 5.12 show the network overhead when the size of a network is increased from 10 to 50. As presented, the network overhead is increased by increasing the network size. By having more nodes and a bigger network topology, there will be **longer** routes and searching for longer routes (in number of hops) requires more **control messages**. In addition, the probability of disconnection in longer routes is more than shorter ones; therefore, the sources need to initiate new route request messages again.

The comparison of two protocols is shown in Figure 5.13 and 5.14 and as presented, the network overhead in FFAR is less than normal AODV. FFAR adds some **more** information to RREQ and RREP (control messages) compared to normal AODV. At first glance, it seems that it should have higher routing overhead, but as explained earlier, FFAR has this ability to find firmer routes and, therefore, there is less demand to search for new routes. In the other word, it will decrease the **route discovery frequency**.

By considering these two issues (the lower route discovery frequency in FFAR and having more control messages compared to AODV), it can be concluded that the overhead in FFAR is less than AODV and especially when the network is getting big and bigger, there will be a significant difference between the two protocols in term of network overhead. Note that I have used error bar in Figure 5.13 and Figure 5.14 shows the data points are the average value of 10 simulation scenarios.

As explained in the drop rate section, the results of mobility scenarios are scenario based. Sometimes it causes that they will not be based on my expectation (see Figure 5.14 when the node number is 25). I should emphasize that the main aim of the evaluation section is to show how my algorithm is better than AODV in all situations.

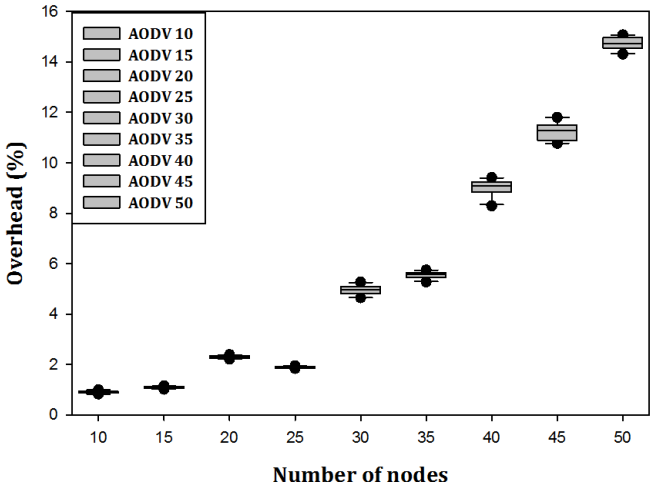

Figure 5.11: *Overhead vs. Network size for AODV by applying error bars.*

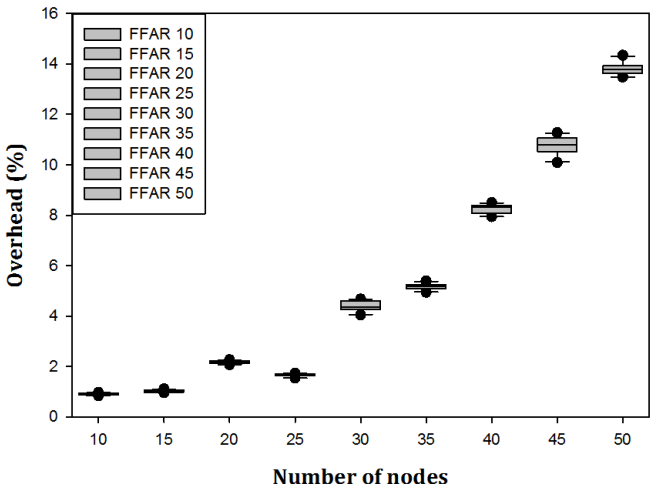

Figure 5.12: *Overhead vs. the network size for FFAR by applying error bars.*

End to end delay

As described in Chapter 2, the routes are very unstable in vehicular ad hoc network systems due to the high mobility of nodes. Moreover, I have learned that AODV

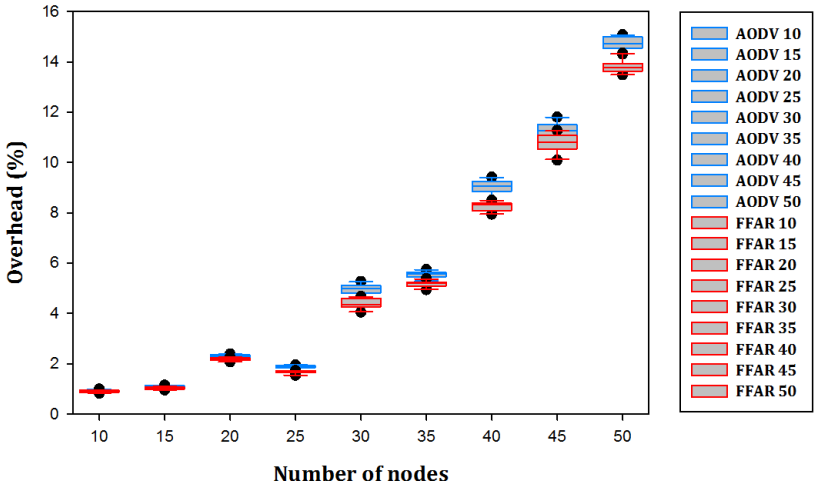

Figure 5.13: *Comparing the overhead of AODV to FFAR when the network size is increased (by using error bars).*

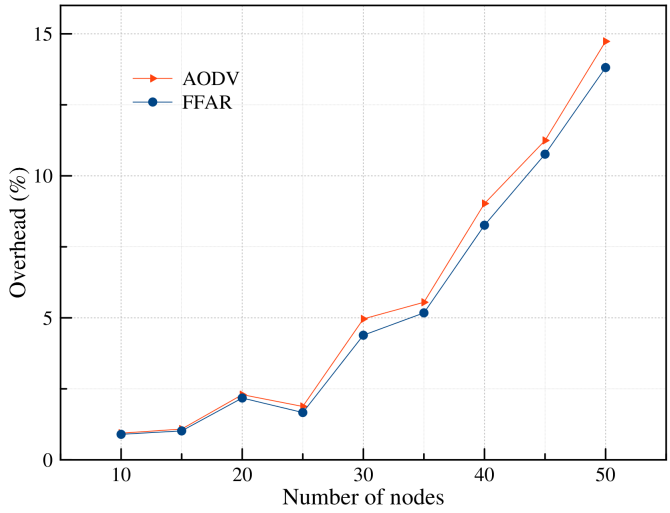

Figure 5.14: *Comparing overhead of AODV to FFAR when network size is increased (by using average value).*

has an error recognition mechanism which generates RRER messages. The sources will initiate new route request messages (RREQ messages) after reviving an error message. Figures 5.15, 5.16 show the end to end delay curves for FFAR and AODV protocols.

Note that in this section I am **just** interested in the **comparison** of AODV and FFAR. Moreover, there is no attempt to **minimize** the end to end delay and I only want to show that my algorithm is superior compared to the normal AODV with respect to the mobility parameters. In terms of end to end delay, the routes that are set up in FFAR are more stable than AODV, and AODV needs to spend more **time** to **repair** the broken links and also to **search** for the new routes. In summary, the end to end delay is higher than FFAR.

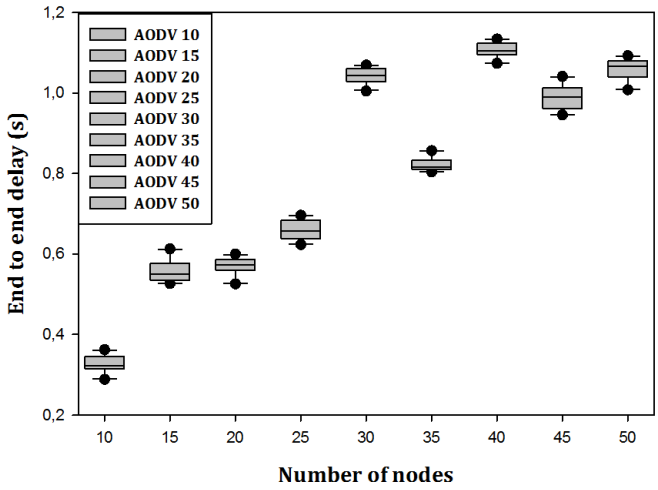

Figure 5.15: *End to end delay vs. the network size for AODV by applying error bars.*

The error bars is used in Figure 5.17 and each data point in Figure 5.18 is the average value of 10 simulation repetitions. Finally, it should be mentioned that the end to end delay is applied only for the successful data packets.

Throughput

Figure 5.19, 5.20 present the average of successful data delivery over network (network throughput) in FFAR and AODV, respectively. As I repeated earlier, by choosing more stable routes in FFAR the drop rate will be decreased and it causes higher throughput compared to normal AODV (see the network throughput definition part).

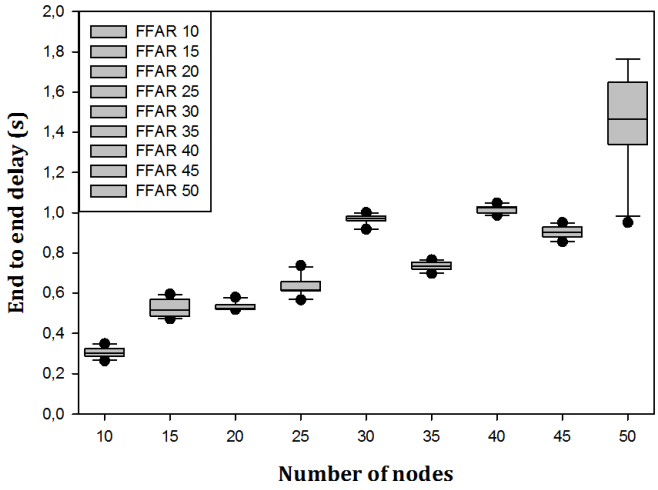

Figure 5.16: *End to end delay vs. the network size for FFAR by applying error bars.*

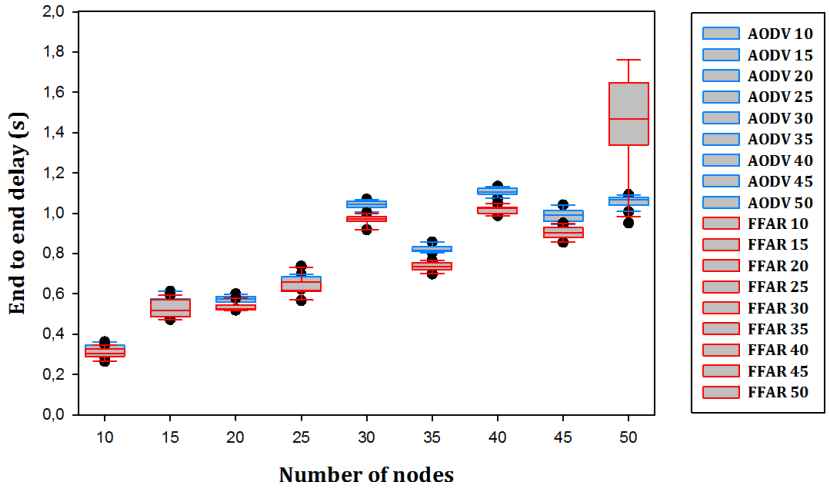

Figure 5.17: *Comparing the end to end delay of AODV to FFAR when the network size is increased (by using error bars).*

Note that I am only interested in the **comparison of algorithms** in terms of network throughput and the results are very related to the scenarios. Let us make

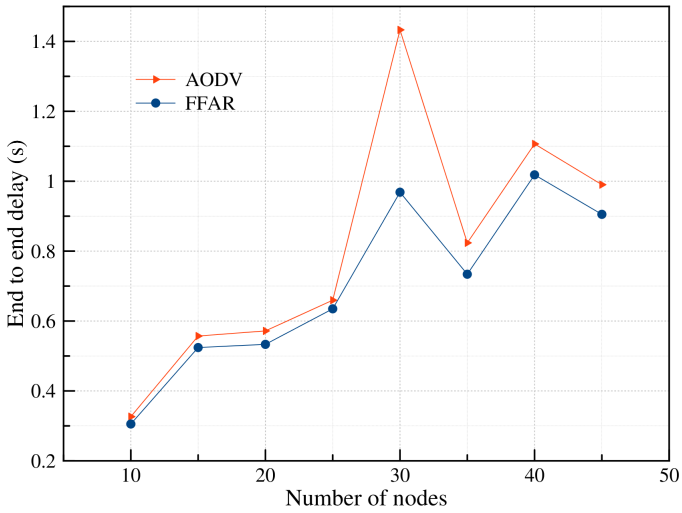

Figure 5.18: *Comparing the end to end delay of AODV to FFAR when the network size is increased (by using average value).*

it clear by an example: if source and destination nodes are placed in one side of generated scenarios, then I have short routes and therefore it is much easier to initiate, search and maintain these routes compared to scenarios that all sources are distrusted all over the map. I will see less drop packets, broken routes and less overhead; therefore, it will have higher throughput.

In other words, they do not follow a specific pattern, but the **main point** is that in all of the scenarios, FFAR is **superior** in terms of network throughput compared to normal AODV. (see [29])

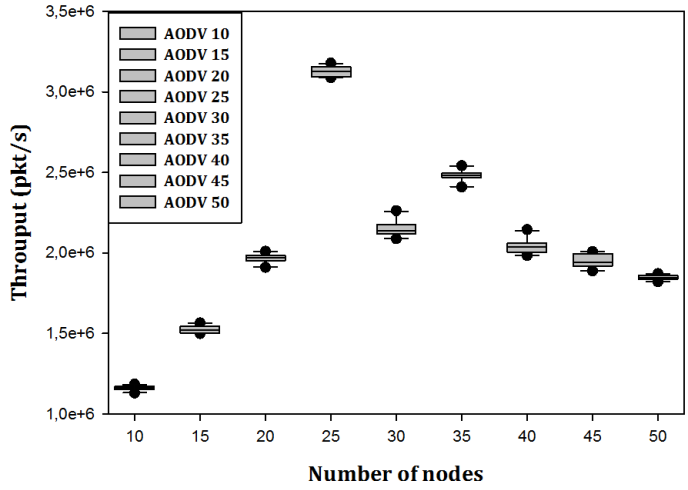

Figure 5.19: *Throughput vs. the network size for AODV by applying error bars.*

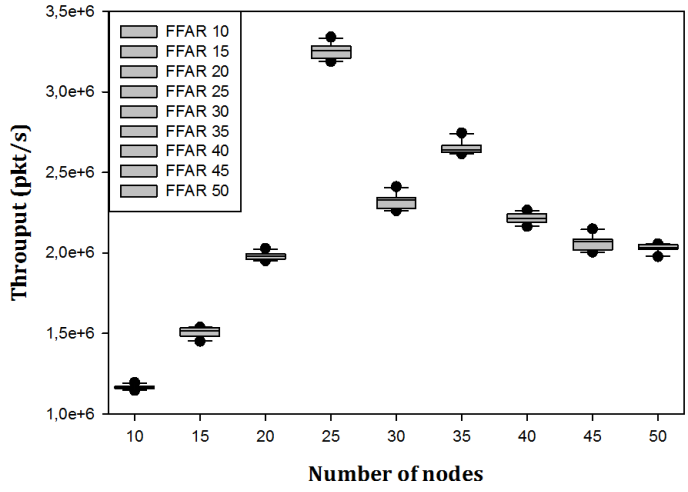

Figure 5.20: *Throughput vs. the network size for FFAR by applying error bars.*

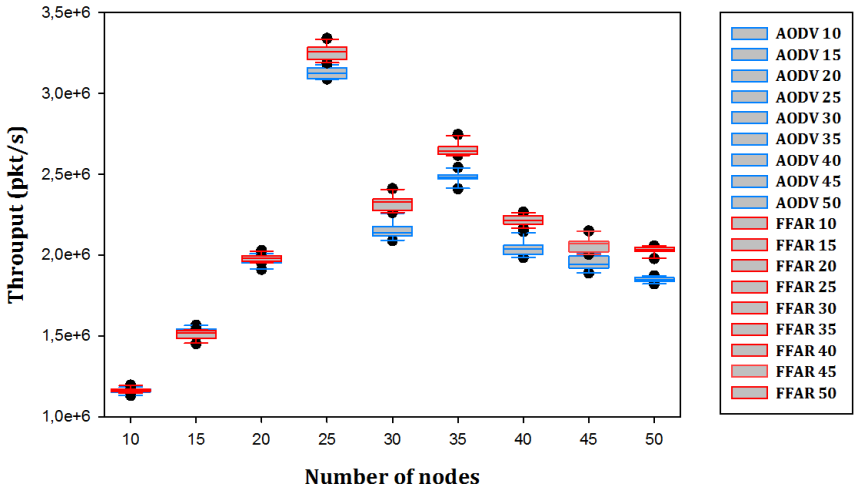

Figure 5.21: *Comparing the throughput of AODV to FFAR when network size is increased (by using error bar).*

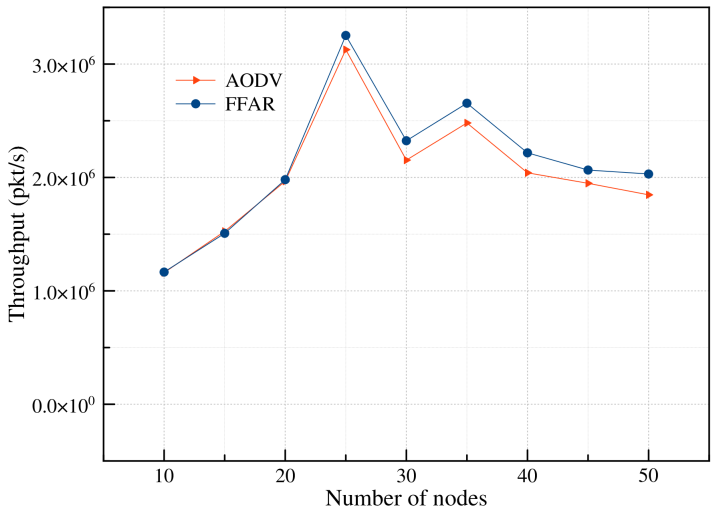

Figure 5.22: *Comparing the throughput of AODV to FFAR when network size is increased (by using average value).*

Chapter 6
Discussion and future work

The following topics are discussed in this chapter:

1. Different QoS requirements that can be applied in vehicle networks
2. Why FFAR has not been tested when speed is varying?
3. Why acceleration has not been applied in my algorithm?
4. Future work

6.1 Different QoS

6.1.1 Quality of Service definition

Generally, the term QoS [55], [56] refers to a sort of guarantee or assurance about the level and grade of service which is given to an application. In other words, QoS provides various priority to different applications. QoS parameters are relying on the application's requirements; for instances, an application which is sensitive to delivery ratio might requires QoS packet delivery ratio assurance. There are many different QoS parameters such as jitter, required bit rate and packet dropping.

6.1.2 Issues and challenges

This section is written based on these papers: [55], [56].

As described earlier, a VANET network, which is a kind of ad hoc network, contains mobile nodes connected through **wireless** links, which causes more errors in contrast to wired links. In order to transmit data packets in VANET networks, mobile (vehicle) nodes should cooperate together for transmission through a shared channel. This property causes some delay and it is hard to guarantee the end to end delay in such a network because the nodes should wait until the channel will be free

to send data packets. In addition, it is possible to have multiple hops where each of them might contend for the channel between a source and a destination. Therefore, due to channel contention, the end to end delay assurance is a difficult task.

The authors in [58] illustrate the characteristics of a network which influences QoS provisioning. **High dynamic topology** of VAENT networks is one of these properties which causes high loss of packets and link disconnections. Another weak point of VANET networks is the lack of **central controller** which is coordinating the activity of mobile nodes (cellular networks and wireless LAN networks has this facility) [57], [58]. By nature, the radio channel is a broadcast medium and wireless applies the radio waves to propagate information.

Interference, attenuation and multi-path propagation are the main weak points of dissemination through the radio waves [58]. The other critical challenge in wireless ad hoc networks is the **hidden terminal** issue which happens when a node can be visible from a wireless access point (AP) while it cannot be seen from other nodes communicating with that AP [57], [58]. **Resource limitations** like bandwidth, storage size, and processing capability is another challenge of VANET networks. Finally, because wireless networks have a broadcasting nature, they are considerably vulnerable to attacks such as eavesdropping, spoofing and DoS. As a result, the **security** issues should get more attention.

6.1.3 QoS comparison criteria

After illustrating the definition of QoS and various issues and challenges, in this section, different requirements that can be applied to compare several QoS are described. The author in [59] explains that the QoS in VANET network application can be satisfied by resource reservation from the network's viewpoint and by adaptation from the application's viewpoint. There are different QoS comparison requirements which are noted here [59].

Requirement 1: applying various radio channels for different kinds of applications.

For each application type, a different radio channel can be applied. For instance, a specific radio channel can be allocated to vehicle Traffic Safety Applications (TSAs).

Requirement 2: Media Access Controller (MAC) layer can support QoS differentiation.

In order to apply a different QoS, the wireless MAC layer should behave differently with various data packets.

Requirement 3: Support of an increased end-to-end throughput.

Fairness is the ability of a network to provide the same level of service to all its users. In order to satisfy fairness guarantees, the end to end throughput should be decreased in VANET applications. This capability should be designed for QoS solutions.

Requirement 4: Achieving low latency [59] in delivering emergency warnings

Emergency messages should be sent immediately. Therefore, it is important to decrease the latency of emergency warning messages in VANET networks and this issue should be considered in designing of QoS solutions.

6.1.4 QoS solutions

In what follows, various algorithms and architectures which are proposed in relation to QoS assurance in VANET networks are presented.

1) A Distributed MAC scheme for Emergency Message Dissemination in Vehicular Ad hoc Networks (DMEMD)

The author in [63] introduces a novel algorithm which supports strict priority for individual packets based on MAC scheme. This algorithm mainly focuses on emergency data propagation in vehicular networks which suffer from high packet drop rate. Two important duties of this algorithm are as follows: first, it should find out strict packet-level priority scheduling, and second, for emergency packets it should support various levels of strict priority .

In this protocol, the MAC scheme operates with three various radio channels which are listed as:

1. a channel for a non-emergency messages
2. a channel for emergency messages
3. a chancel for pluses (it is named as priopulses)

Figure 6.1 illustrates when an emergency message is sent via an emergency channel, a priopulses is sent too, but it will be sent through a control channel. As shown in Figure 6.1, node A is the node which generates an emergency message while node B is A's neighbour and node C is a hidden terminal which is placed between node A and B and blocks their connection. The author explains that the priopulses guarantees

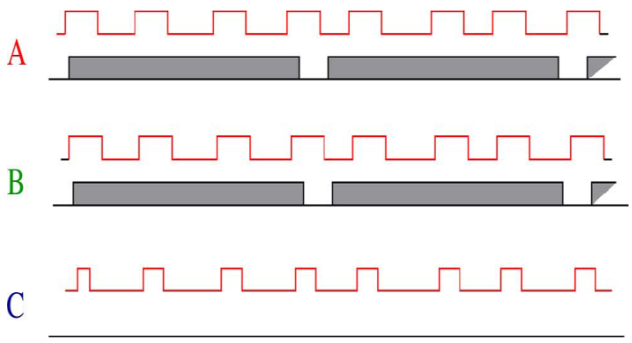

Figure 6.1: *Emergency packets and their accompanying priopulses [63].*

that, based on the emergency level, each emergency packet gets the priority that it deserves. The full algorithm is described in [63].

2) A cross layer multihop delivery protocol with fairness guarantees for vehicular networks (CVIA)

In [62], the author proposed a new algorithm which is named the controlled vehicular internet access (CVIA) protocol. This algorithm targets to increase the end to end throughput when it guarantees the bandwidth usage fairness between road segments. "To achieve this goal, the CVIA protocol eliminates contention in relaying packets over long distances. CVIA creates single-hop vehicle clusters and mitigates the hidden node problem by dividing the road into segments and controlling the active time of each segment" [62].

The fairness in the context of this paper is defined as the equal throughput for all segments and in order to satisfy this requirement, the content of packet trains should be checked while leaving a segment. In order to assure fairness for packet trains, CVIA makes sure the packets in each segments have the same number (see Figure 6.2)

3) Internet Access Protocol Providing QoS in Vehicular Networks with infrastructure Support (CVIA-QoS)

The author in [61], proposed a new algorithm which is named CVIA-QoS and it is defined as "A cross-layer solution for vehicular multi-hop networks spanning MAC and routing functions with infrastructure support" [61]. Actually this algorithm expands the previous illustrated CVIA algorithm and compared to the CVIA which is providing the best-effort traffic, CVIA-QoS assures a fixed delay bound for real-time applications such as voice and video streaming [59], [61].

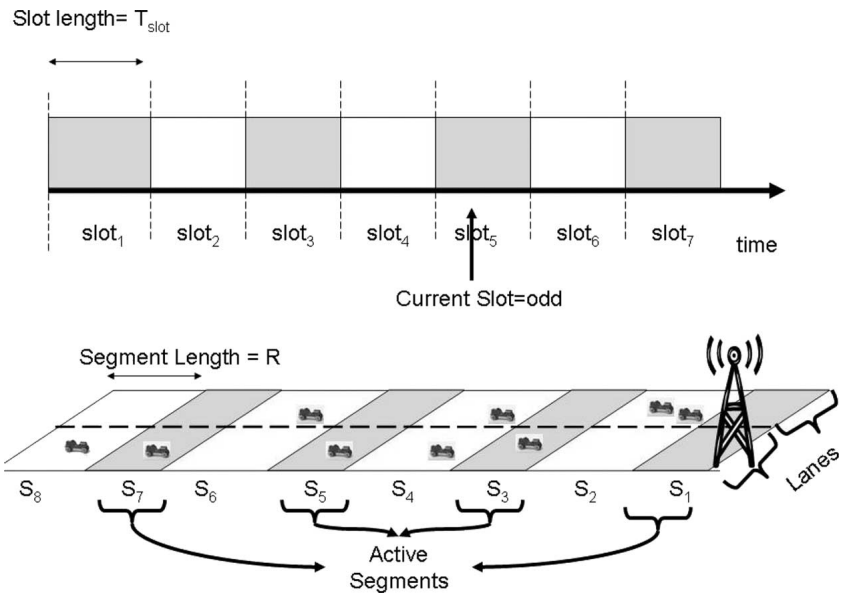

Figure 6.2: *Slots and segments in the CVIA protocol [62].*

Based on the proposed algorithm, first, an available bandwidth will be allocated to the soft real time traffic, then the rest of bandwidth will be allocated to the best effort traffic. Figure 6.3 explains that each time slot contains two parts: a high priority period (HPP) and a low priority period (LPP). In Figure 6.3, various phases that are applied in the CVIA-QoS protocol are presented and each session which needs service, should send registration packets. Through these phases the selected packeted will be transmitted in the gateway. The LPP already has been defined in CVIA while HPP is a novel phase that is described here [59], [61].

The author concludes that the capacity of the proposed algorithm regarding to the "best effort traffic throughput" is smaller than CVIA, because CVIA-QoS spends some of its throughput to guarantee the fixed delay bound. Moreover, the results indicate that the soft real time throughput is not decreased, due to admission control and polling [59], [61].

4) A Vehicle-to-Vehicle Communication Protocol for Cooperative Collision Warning (VCWC)

The novel algorithm which is presented in [60] can help solve considerable challenges in V2V communications. It is named Vehicular Collision Warning Communication (VCWC) [60] and the main purpose of this algorithm is providing low latency

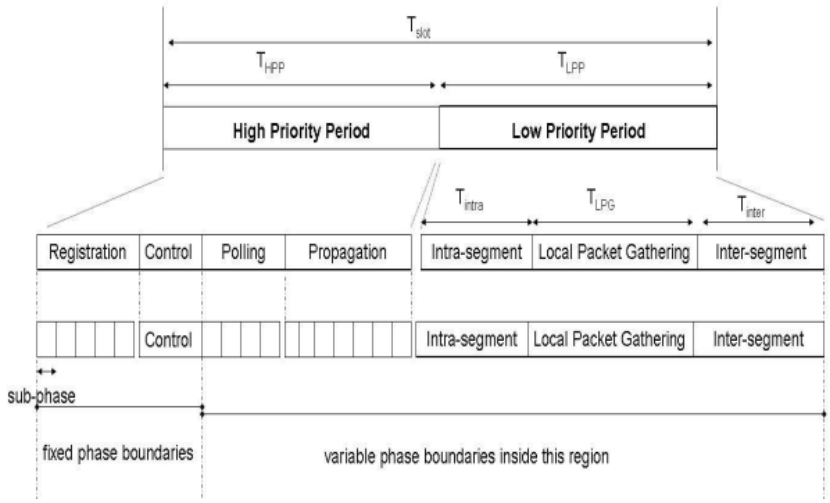

Figure 6.3: *Phases in the CVIA-QoS protocol [61].*

for emergency warning applications in VANET networks [60]. For instance, in a congestion area when an accident occurs, VCMC warns the vehicles behind the collision with a low delay. The author assumes that the vehicle has this ability to gain its own geographical location. Moreover, each vehicle has a wireless transceiver in range of 300 meters. In what follows, the author illustrates one of the challenges regarding emergency messages delivery.

It is necessary to warn the vehicles behind a road collision immediately. Due to the nature of wireless communication, this fast data delivery is very unreliable. The author solves this issue by proposing a new rate decreasing protocol. (for more information see [60]). In conclusion, the simulation results proves that this algorithm can provide essential QoS emergency warning delivery requirements.

6.2 Varying speed for simulation

It has been planned to observe and evaluate simulations results when speed is varying. But due to the algorithm's structure and properties, it has been decided to not interpret these results. FFAR considers the relative speed as the input for its fuzzy logic controller and based on the FFAR design, it chooses a node which has a "good" relative speed in relation to source and destination. The term relative speed is defined as the diference in speed between an intermediate node and destination and source (see Chapter 3).

On the other hand, BonnMotion creates a mobility model in which the mobile nodes move with random speed. By increasing the speed parameter, all nodes will have higher speed. Therefore, it is seen that nodes are moving faster, but the relative deference is still the same approximately. By increasing the speed parameter, it was observed that the simulation results does not follow an expected behaviour.

6.3 Acceleration

In the basic proposed algorithm, acceleration was considered as a novel parameter for fuzzy logic system, but during the implementation process I was faced with the problem that the acceleration has been defined "zero" for mobile node in NS2. In this model, it has been assumed that the simulation time is divided into very short time slots where in each slot, a mobile node moves with constant speed. In other words, simulation curves move similar to a stair in which each slot has a constant speed and a mobile node can jump from one speed to another speed. To sum up, a mobile node's speed cannot smoothly increase; therefore, **the actual** acceleration counts as zero in this mobility model. In the future, I plan to implement this idea, acceleration for mobile nodes, in NS3 (see [72], [73], [74] for more information about NS3).

6.4 Future work

Based on this work, in future I plan to improve FFAR by considering more mobility parameters and more environments (urban, rural). Moreover, other AI techniques can be applied to find the optimized results. For example, One of the interesting issues which can be studied is artificial neural network.

Neural network

An artificial neural network can be defined as a computational model which is inspired by biological information processing in nervous systems such as the human brain, which has pattern recognition and machine learning abilities. An artificial neural network [51], [53] (ANN) is composed of considerable interconnected neurones (processing elements) that cooperate to find out the optimised results.

The ability of learning is one of the most interesting parts of a neural network. ANN can modify its internal structure based on the obtained information. The process of learning stands on adjusting of weights. As shown in Figure 6.4, each line presents the connection between two neurons and will be applied to transfer information. Each line has a weight that will control the signal between two neurons. There is no need to adjust the weights if the network creates a "good" output. The network will adapt the weights if the output will be "poor".

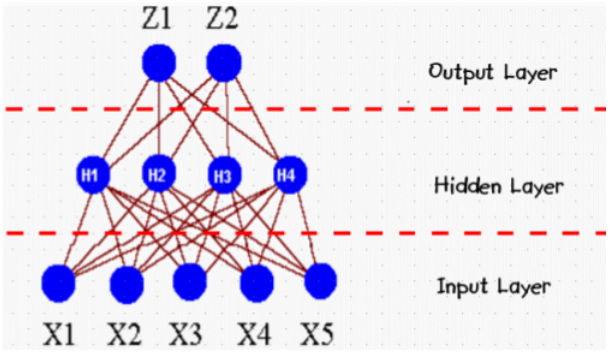

Figure 6.4: *Neural network [9].*

The neural networks is a significant tool due to these characteristics: adaptive learning, fault tolerance, real time operation and self organization. Neural networks and fuzzy systems have common features. One of these features is solving a problem in lack of any mathematical model. As illustrated in Figure 6.5, a neuro-fuzzy [52], [54], [9] system has the following blocks: fuzzification, multiplication, summation, and division. When neural networks and fuzzy logic systems are combined, it gains both advantages in one framework and it creates **smoother control surface**, although it will be more difficult to implement such a system.

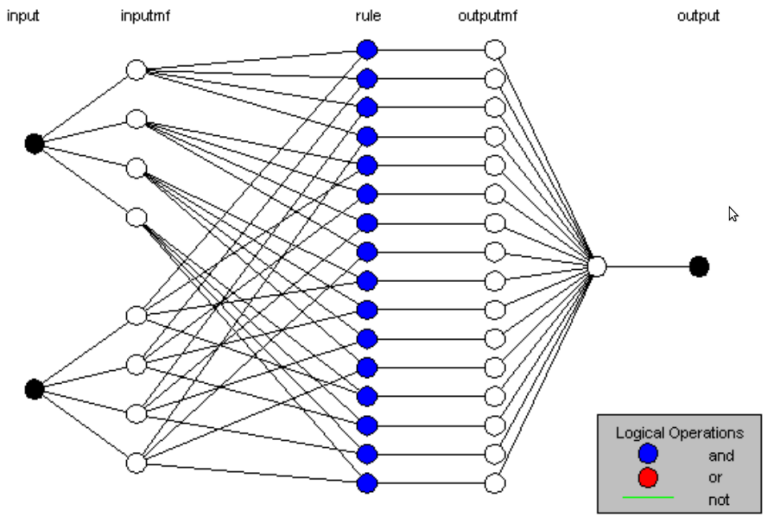

Figure 6.5: *Neuro-fuzzy system [9].*

Chapter 7
Conclusion

In this thesis, a number of achievements have been obtained:

First, background information such as AODV, the concept of fuzzy logic, various forwarding algorithm related to AODV and the challenges in general routing algorithm, their features and classification are studied.

Second, a novel algorithm named Four-inputs Fuzzy-logic-based AODV Routing (FFAR) is proposed. FFAR uses four input parameters to find the most stable nodes and increase the network throughput.

Third, several simulations have been done to prove that FFAR has better performance when a network is getting crowded. FFAR has improved normal AODV in different metrics such as end to end delay and route discovery frequency. The performance of the fuzzy logic system is studied by using NS2, BonnMotion and the jFuzzyLogic library. At the end, the future work and possible QoS which could be used in relation to VANET are described.

Bibliography

[1] SHENG-HAI AN, B.-H. L. DONG-RYEOL SHIN, 2011, "A Survey of Intelligent Transportation Systems", Third International Conference on Computational Intelligence, Communication Systems and Networks, CA 795-895.

[2] FEMA Position Paper, 2011, "Intelligent Transport Systems (ITS)".

[3] Chai Keong Toh, 2002, "Ad Hoc Mobile Wireless Networks", Prentice Hall Publishers.

[4] FAN, L, YU, W, 2007, "Routing in vehicular ad hoc networks: A survey", Vehicular Technology Magazine, IEEE, 2, 12-22.

[5] Basu Dev Shivahare, Charu Wahi, Shalini Shivhare, 2012, "Comparison Of Proactive And Reactive Routing Protocols In Mobile Ad hoc Network Using Routing Protocol Property", ISSN 2250-2459, Volume 2, Issue 3.

[6] [RFC3561] C. Perkins, E. Belding-Royer and S. Das, July 2003, "Ad hoc On-Demand Distance Vector (AODV) routing".

[7] Teerawat Issariyakul, Ekram Hossain, 2010, "Introduction to Network Simulator NS2", 2nd edition, Springer, pp. 305-315.

[8] mathworks, 2013, "what-is-fuzzy-logic", http://www.mathworks.se/help/fuzzy/what-is-fuzzy-logic.html, [Accessed 14 January 14].

[9] Franck Dernoncourt, January 2013, "Introduction to fuzzy logic", MIT, USA.

[10] Stanford Encyclopedia of Philosophy, 2006, "Fuzzy Logic", Stanford University, 2006-07-23, Retrieved 2008-09-30.

[11] Zadeh, L.A, (1965), "Fuzzy sets", Information and Control 8 (3): 338–353.

[12] Zadeh, L. A. et al, 1996, "Fuzzy Sets, Fuzzy Logic, Fuzzy Systems", World Scientific Press, ISBN 981-02-2421-4.

[13] calvin, 2014, "Fuzzy Operations", https://www.calvin.edu/ pribeiro/othrlnks/-Fuzzy/fuzzyops.htm, [Accessed 14 January 14].

[14] Rakesh Kumar, and Mayank Dave, "A Review of Various VANET Data Dissemination Protocols", Department of Information Technology, M. M. University, Mullana, Haryana, India.

[15] Annu Mor, "Study of Different Type of Data Dissemination Strategy in VANET", Research Scholar, Deptt. Of Computer Science Applications, Kurukshetra University, Kurukshetra, Haryana, India.

[16] Anup Dhamgaye, Nekita Chavhan, "Survey on security challenges in VANET", Wireless Communication and Computing, Dept. of CSE, G. H. Raisoni College of Engineering, Nagpur, India.

[17] Kevin C. Lee, Uichin Lee, Mario Gerla, "Survey of Routing Protocols in Vehicular Ad Hoc Networks", UCLA, USA.

[18] Uma Nagaraj, Dr. M. U. Kharat, Poonam, "Study of Various Routing Protocols in VANET", Dhamal 1,3Dept. of Computer Engg, M.A.E., Pune, India 2Dept. of Computer Engg., M.E.T., Nashik, India.

[19] M. Zapata and N. Asokan, 2002, "Securing Ad-hoc Routing Protocols", in Proc. of ACM Workshop on Wireless Security (WiSe), Atlanta, GA, Sept.

[20] W. Peng and X.C. Lu, August 2000, "On the reduction of broadcast redundancy in mobile ad hoc networks", In ACM MobiHoc 2000, pages 129 – 130, Boston, Massachusetts, USA.

[21] I. Stojmenovic and M. Seddigh, 2000, "Broadcasting algorithms in wireless networks", In Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet SSGRR, L'Aquila, Italy.

[22] S. Lee, W. Su, and M. Gerla, 1999, "Ad hoc Wireless Multicast with Mobility Prediction", Proceeding of IEEE ICCCN'99, Boston, MA, pp. 4-9.

[23] W. Su, 1999, "Motion Prediction in Mobile/Wireless Networks", PhD Dissertation, UCLA Computer Science Department, Los Angeles, CA.

[24] Yufeng Chen Zhengtao Xiang, Wei Jian and Weirong Jiang, 2009, "An Improved AOMDV Routing Protocol for V2V Communication", Intelligent Vehicles Symposium, IEEE Conferences.

[25] Abedi, O. Fathy, M. Taghiloo, J, 2008, "Enhancing AODV Routing Protocol Using Mobility Parameters in VANET", Iran Univ. of Sci. and Technol, Tehran Computer Systems and Applications, AICCSA 2008. IEEE/ACS International Conference on.

[26] Hafez Maowad, Eman Shaaban, "Enhancing AOMDV Routing Protocol for V2V Communication", Department of Computer System, Faculty of Computer and Information science, Ain shams University Cairo, Egypt.

[27] M.M.Goswami, R.V. Dharaskar, V.M.Thakare, 2009, "Fuzzy Ant Colony Based Routing Protocol For Mobile Ad Hoc Network", International Conference on Computer Engineering and Technology, pp.438-444.

[28] Hui Liu, Jie Li, Yan-Qing Zhang and Yi Pan, 2005, "An Adaptive Genetic Fuzzy Multi-path Routing Protocol for Wireless Ad-Hoc Networks". Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN'05). Towson University, Towson, Maryland, USA, pp. 468-475.

[29] WANG Xiao-bo, YANG Yu-liang, AN Jian-wei, 2009, "Multi-Metric Routing Decisions in VANET", Department of Communication Engineering, University of Science and Technology Beijing, Beijing, China.

[30] S. Lee, W. Su, and M. Gerla, 1999, "Ad hoc Wireless Multicast with Mobility Prediction", Proceeding of IEEE ICCCN'99, Boston, MA, pp. 4-9.

[31] W. Su, 1999, "Motion Prediction in Mobile/Wireless Networks", PhD Dissertation, UCLA Computer Science Department, Los Angeles, CA.

[32] C. E. Perkins and E. M. Royer, 1999, "Ad-hoc On-Demand Distance Vector Routing", In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, pages 90–100, New Orleans, LA.

[33] C. Perkins, 1997, "Ad Hoc On Demand Distance Vector (AODV) routing", Internet-Draft, draft – ietf-MANET-aodv-00. txt.

[34] C. Perkins, E. Royer, and S. Das, 1997, "Ad hoc on-demand distance vector (AODV) routing". Internet Draft, Internet Engineering Task Force, Mar.

[35] Pedro Albertos, Antonio Sala, Manuel Olivares, "Fuzzy Logic Controllers. Methodology. Advantages and Drawbacks".

[36] Asmussen, Soren, Glynn, Peter W, 2007, "Stochastic Simulation: Algorithms and Analysis", Springer, Series: Stochastic Modelling and Applied Probability, Vol. 57, 2007.

[37] Banks, Carson, Nelson Nicol, "Discrete Event System Simulation", Pearson.

[38] ns, "ns2", http://www.isi.edu/nsnam/ns/, [Accessed 25 January 14].

[39] J. L. Font, P. Inigo, M. Dominguez, J. L. Sevillano, and C. Amaya, 2011, "Analysis of source code metrics from ns-2 and ns-3 network simulators," Simulation modelling practice and theory.

[40] Alex Ali Hamidian, January 2003, "A Study of Internet Connectivity for Mobile Ad Hoc Networks in NS 2", Department of Communication Systems Lund Institute of Technology, Lund University, Box 118, S-221 00 Lund, Sweden.

[41] Tony Larsson, Nicklas Hedman, 1998, "Routing Protocols in Wireless Ad-hoc Networks-A Simulation Study", Stockholm.

[42] Stutz, Michael (September 19, 2006), "Get started with GAWK: AWK language fundamentals", developerWorks, IBM, Retrieved 2010-10-23, "[AWK is] often called a data-driven language – the program statements describe the input data to match and process rather than a sequence of program steps".

[43] Sun, Jun-Zhao and; Jaakko Sauvola (2002), "Mobility and mobility management: a conceptual framework", Proc. 10th IEEE International Conference on Networks, Retrieved 23 February 2009.

[44] N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, and M. Schwamborn, 2010, BonnMotion: 10 - a Mobility Scenario Generation and Analysis Tool, in Proc. of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools '10), Torremolinos, Malaga, Spain.

[45] Cingolani, Pablo, and Jesus Alcala-Fdez, "jFuzzyLogic: a Java Library to Design Fuzzy Logic Controllers According to the Standard for Fuzzy Control Programming".

[46] Cingolani, Pablo, and Jesus Alcala-Fdez, 2012, "jFuzzyLogic: a robust and flexible Fuzzy-Logic inference system language implementation". Fuzzy Systems (FUZZ-IEEE), IEEE International Conference on. IEEE, 2012.

[47] Bai, Fan; Helmy, Ahmed (2006), "A Survey of Mobility Models in Wireless Adhoc Networks", (Chapter 1 in Wireless Ad Hoc Networks, Kluwer Academic, 2006, Based on result of Google Scholar search, Additional work needed to identify this book.)

[48] Jiajia Liu, Xiaohong jiang, Hiroki Nishiyama, Nei Kato, and Xuemin (Sherman) Shen, Apr. 2012, "End-to-End Delay in Mobile Ad Hoc Networks with Generalized Transmission Range and Limited Packet Redundancy," IEEE Wireless Communications and Networ king Conference (WCNC 2012), Paris, France, pp. 1731-1736.

[49] John David Cavanaugh, "Protocol Overhead in IP/A TM Networks", Minnesota Super, computer Center, Inc.

[50] Jennifer Yick, Biswanath Mukherjee, Dipak GhosalCorresponding, "Wireless sensor network survey", Department of Computer Science, University of California, Davis, CA 95616, United State.

[51] Quek, C. & Zhou, R. W, (2001), "The POP learning algorithms: reducing work in identifying fuzzy rules", Neural Networks, 14(10), 1431-1445.

[52] Zhou, R. W. & Quek, C, (1996), "POPFNN: A Pseudo Outer-product Based Fuzzy Neural Network", Neural Networks, 9(9), 1569-1581.

[53] Bhadeshia H. K. D. H, 1999, “Neural Networks in Materials Science”, ISIJ International 39 (10): 966–979.

[54] M. Figueiredo and F. Gomide, “Design of Fuzzy Systems Using Neuro Fuzzy Networks”, IEEE Transactions on Neural Networks, 1999, Vol. 10, no. 4, pp.815 827.

[55] Abbas, A.M. and Kure, “Quality of Service in mobile ad hoc networks: a survey”, Int. J. Ad Hoc and Ubiquitous Computing.

[56] Leonard Franken, 1996, “Quality of Service Management: A Model-Based Approach”. PhD thesis, Centre for Telematics and Information Technology.

[57] Shouzhi Xu, Pengfei Guo, Bo Xu, Huan Zhou, 2013, “QoS Evaluation of VANET Routing Protocols”, JOURNAL OF NETWORKS, VOL. 8, NO. 1, Collage of Computer and Information Technology, China Three Gorges University, Yichang, China.

[58] Shraddha Saharan1 & Rakesh Kumar, June 2010, “QoS Provisioning in VANETs using Mobile Agent”, ol. 1, No. 1, pp. 199-202, India.

[59] P.W.H.M. Hornman, “QoS support for traffic safety applications in VANET communication infrastructures”, University of Twente, Enschede, The Netherlands.

[60] Yang X., Liu L., Vaidya N.H., and Zhao F, Aug. 2004, “A vehicle-to-vehicle communication protocol for cooperative collision warning”, pages 114-123.

[61] Korkmaz G., Ekici E., and Ozguner F, 2006, “Internet access protocol providing QoS in vehicular networks with infrastructure support”, In IEEE Intelligent Transportation Systems Conference, ITSC’06, pages 1412-1417.

[62] Korkmaz G., Ekici E., and Ozguner F, 2006, “A cross-layer multihop data delivery protocol with fairness guarantees for vehicular networks”, IEEE Transactions on Vehicular Technology,55(3):865-875.

[63] Peng J. and Cheng L, 2007, “A distributed MAC scheme for emergency message dissemination in vehicular ad hoc networks”, IEEE Transactions on Vehicular Technology, 56(6):3300-3308, Nov.

[64] C. Siva Ram Murthy and B. S. Manoj, May 2004, “Ad hoc Wireless Networks: Architectures and Protocols”, Prentice Hall PTR.

[65] isi, 2014, “Running Wireless Simulations in ns”, http://www.isi.edu/nsnam/ns/tutorial/nsscript5.html, [Accessed 25 January 14].

[66] topcoder, “Shortest distance between a point and a line”, http://www. topcoder.com/, [Accessed 25 January 14].

[67] github, “AODV.Tcl”, https://github.com/softvar/ns2-oadv/blob/master/aodv.tcl, [Accessed 25 January 14].

[68] code.google, "AODV.Tcl", https://code.google.com/p/ns-allinone-2-34-imp-protocol/source/browse/trunk/src/ns-2.34/aodv/example.tcl?r=7, [Accessed 25 January 14].

[69] nsnam, "awk-scripts-for-ns2-to-process-data", http://www.nsnam.com/2013/03/awk-scripts-for-ns2-to-process-data.html, [Accessed 25 January 14].

[70] mohittahiliani, "awk-script-for-ns2", http://mohittahiliani.blogspot.no/2009/12/awk-script-for-ns2.html, [Accessed 25 January 14].

[71] Hannes Hartenstein, University of Karlsruhe Kenneth P. Laberteaux, "A Tutorial Survey on Vehicular Ad Hoc. Networks", Toyota Technical Center.

[72] Henderson, Tom (2012-06-09), "upcoming ns-3.1 release", ns-announce, Retrieved 2013-05-31.

[73] Henderson, Tom (2013-12-20), "ns-3.19 released", ns-announce, Retrieved 2014-1-11.

[74] nsnam, 2014, "publications", http://www.nsnam.org/overview/publications/, [Accessed 14 January 14].

Chapter A
Appendix A

In this section, various AWK functions are listed. These files were copied from NS2 forums and INTERNET (for example, see [69], [70]) while the author was anonymous. Moreover, they do not have the direct effects in this report. In other words, they are just used to drive required information from trace files.

A.1

Listing A.1: *AWK 1.*

```
BEGIN {

seqno = -1;

droppedPackets = 0;

receivedPackets = 0;

count = 0;

}

{

#packet delivery ratio

if($4 == "AGT" && $1 == "s" && seqno < $6) {

seqno = $6;

} else if(($4 == "AGT") && ($1 == "r")) {

receivedPackets++;
```

```
} else if ($1 == "D" && $7 == "cbr" && $8 > 512){

droppedPackets++;

}

#end-to-end delay

if($4 == "AGT" && $1 == "s") {

start_time[$6] = $2;

} else if(($7 == "cbr") && ($1 == "r")) {

end_time[$6] = $2;

} else if($1 == "D" && $7 == "cbr") {

end_time[$6] = -1;

}

}


END {

for(i=0; i<=seqno; i++) {

if(end_time[i] > 0) {

delay[i] = end_time[i] - start_time[i];

count++;

}

else

{

delay[i] = -1;

}
```

```
}

for(i=0; i<count; i++) {

if(delay[i] > 0) {

n_to_n_delay = n_to_n_delay + delay[i];

}

}

n_to_n_delay = n_to_n_delay/count;

print "\n";

print "GeneratedPackets = " seqno+1;

print "ReceivedPackets = " receivedPackets;

print "Packet Delivery Ratio = " receivedPackets/(seqno+1)*100

"%";

print "Total Dropped Packets = " droppedPackets;

print "Average End-to-End Delay = " n_to_n_delay * 1000 " ms";

print "\n";

}
```

A.2

Listing A.2: *AWK 2, normalized routing load*

```
##########################################################################
#        AWK Script to calculate Normalized Routing Load     #
#             Works with AODV, DSDV, DSR and OLSR            #
##########################################################################

BEGIN{
```

```
recvd = 0;#################### to calculate total number of data packets
    received
rt_pkts = 0;################## to calculate total number of routing packets
    received
}

{
##### Check if it is a data packet
if (( $1 == "r") && ( $7 == "cbr" || $7 =="tcp" ) && ( $4=="AGT" )) recvd++;

##### Check if it is a routing packet
if (($1 == "s" || $1 == "f") && $4 == "RTR" && ($7 =="AODV" || $7 =="message"
 || $7 =="DSR" || $7 =="OLSR")) rt_pkts++;
}

END{
printf("#####################################################################\n");
printf("\n");
printf("                    Normalized Routing Load = %.3f\n",
    rt_pkts/recvd);
printf("\n");
printf("#####################################################################\n");
}
```

A.3

Listing A.3: *AWK 3, average throughput*

```
  BEGIN {

        recvdSize = 0

        startTime = 400

        stopTime = 0

    }



    {

            event = $1
```

```
            time = $2
            node_id = $3
            pkt_size = $8
            level = $4

  # Store start time
  if (level == "AGT" && event == "s" && pkt_size >= 512) {
    if (time < startTime) {
            startTime = time
            }
      }

# Update total received packets' size and store packets arrival time
if (level == "AGT" && event == "r" && pkt_size >= 512) {
     if (time > stopTime) {
           stopTime = time
         }
    # Rip off the header
    hdr_size = pkt_size % 512
  pkt_size -= hdr_size
     # Store received packet's size
     recvdSize += pkt_size
       }
```

```
    }



    END {
       printf("Average Throughput[kbps] = %.2f\t\t StartTime=%.2f
       \tStopTime=%.2f\n",(recvdSize/(stopTime-startTime))
       *(8/1000),startTime,stopTime)

  }
```

A.4

Listing A.4: *AWK 4*

```
BEGIN {
 TotalSendPacket = 0;
 TotalRecvPacket = 0;
 aodv = 0;
 total_pkt_size = 0;

 highest_packet_id = 0;
      duration_total = 0;
      packet_number = 0;
      drop_packet = 0;
        requests = 0;
        frequency = 0;
}
{
 # field parameters of normal trace
 event          = $1;    #; Event : r , s , d , f
 time           = $2;    #; Time : send time , receive time , drop time
 node           = $3;    #; Node : source node , receive node
 trace_type     = $4;    #; Trace type MAC trace
 pkt_id         = $6;    #; Event ID : Frame sequence number for total flows
 pkt_type       = $7;    #; Packet type : RTS , CTS , Data = cbr , ACK
 pkt_size       = $8;    #; Packet size (unit : bytes)

        source_ip      =$23;

 if(event == "s" && trace_type == "AGT" && pkt_type == "cbr") {
  TotalSendPacket++;
 }
 if(event == "r" && trace_type == "AGT" && pkt_type == "cbr") {
```

```
 TotalRecvPacket++;
}
if((event == "s"||event == "f") && (pkt_type == "AODV"||pkt_type ==
    "AOMDV") ){
 aodv++;
}
if (event == "s" && trace_type == "AGT" && pkt_type == "cbr" &&
 start_time[pkt_id] == 0 ){
 start_time[pkt_id] = time;
   if ( pkt_id > highest_packet_id )
  highest_packet_id = pkt_id;
  }
  if (event == "r" && trace_type == "AGT" && pkt_type == "cbr" ) {
 total_pkt_size = total_pkt_size + pkt_size;
 end_time[pkt_id] = time;
}
  if (event == "d" && pkt_type == "cbr"){
     drop_packet++;
     end_time[pkt_id] = -1;
  }
      source_ip=substr(source_ip,2);
      node=substr(node,2,2);
      if (event =="s" && $25=="(REQUEST)" && node==source_ip){
       requests++;
      }
}
END {
 throughput = total_pkt_size * 8 / 300;

 for ( pkt_id = 0; pkt_id <= highest_packet_id; pkt_id++ ){
       start = start_time[pkt_id];
       end = end_time[pkt_id];
       if ( end!=-1 && start < end ){
          packet_duration = end - start;  # single distance
          duration_total += packet_duration; # total duration
          packet_number++;                # count packet number
       }

   }
 frequency=requests/300;
 printf("Route Discovery Frequency:%f\n",frequency);
 delay=duration_total / packet_number;
printf("Packet Delivery Ratio is: %f\n",TotalRecvPacket/TotalSendPacket);
printf("Packet Drop ratio is: %f\n",(TotalSendPacket-
TotalRecvPacket)/TotalSendPacket);
```

```
 printf("Routing Overhead ratio is: %f\n",aodv/TotalRecvPacket);
 printf("The Average End-to-End Delay of this Network is: %f\n",delay);
 printf("Throughput is: %f\n",throughput);
 printf("The totalrecvpacket is: %f\n",TotalRecvPacket);
 printf("the totalcomandpacket is: %f\n",aodv);
}
```

Chapter B

Appendix B

This appendix illustrates the source codes regarding to fuzzy logic [45], [46] applied in this project. The basis of these codes are taken form jFuzzylogic library and afterwards, it is modified based on issues such as membership functions and defined rules.

B.1

Listing B.1: *Fuzzy logic codes written in C++*

```
#include <stdio.h>

#include <stdlib.h>
double ruleAccumulationMethod_max(double defuzzifierValue, double
    valueToAggregate)
{ return ( defuzzifierValue > valueToAggregate ? defuzzifierValue :
    valueToAggregate ); }

double ruleActivationMethod_min(double degreeOfSupport, double membership)
{ return (degreeOfSupport < membership ? degreeOfSupport : membership); }

double ruleConnectionMethod_and(double antecedent1, double antecedent2)
{ return (antecedent1 < antecedent2 ? antecedent1 : antecedent2); }

class FunctionBlock_tipper {

       public:
       // VAR_INPUT
       double Angle;
       double Direction;
```

```
double Position;
double Velocity;

// VAR_OUTPUT
double Stability;

private:
// FUZZIFY Angle
double Angle_bad;
double Angle_good;

// FUZZIFY Direction
double Direction_bad;
double Direction_good;

// FUZZIFY Position
double Position_excelent;
double Position_good;
double Position_poor;

// FUZZIFY Velocity
double Velocity_bad;
double Velocity_good;


// DEFUZZIFY Stability
double defuzzify_Stability[1000];


public:
FunctionBlock_tipper();
void calc();
void print();

private:
void defuzzify();
void fuzzify();
void reset();
double membership_Angle_bad(double x);
double membership_Angle_good(double x);
double membership_Direction_bad(double x);
double membership_Direction_good(double x);
double membership_Position_excelent(double x);
double membership_Position_good(double x);
double membership_Position_poor(double x);
```

```
        double membership_Stability_bad(double x);
        double membership_Stability_excelent(double x);
        double membership_Stability_good(double x);
        double membership_Stability_normal(double x);
        double membership_Stability_toobad(double x);
        double membership_Stability_verybad(double x);
        double membership_Stability_verygood(double x);
        double membership_Velocity_bad(double x);
        double membership_Velocity_good(double x);
        void calc_No1();

};

// Constructor
FunctionBlock_tipper::FunctionBlock_tipper() {
        Stability = 0.0;
}

// Calculate function block
void FunctionBlock_tipper::calc() {
        reset();
        fuzzify();
        calc_No1();
        defuzzify();
}

// RULEBLOCK No1
void FunctionBlock_tipper::calc_No1() {
        // RULE 1 : IF (((Position IS excelent) AND (Direction IS good)) AND
        (Velocity IS good))
         AND (Angle IS good) THEN Stability IS excelent;
        double degreeOfSupport_1 = 1.0 * (
            ruleConnectionMethod_and(ruleConnectionMethod_and(ruleConnectionMethod_
        and(Position_excelent , Direction_good) , Velocity_good) ,
            Angle_good) );
        if( degreeOfSupport_1 > 0 ) {
               for (int i = 0 ; i < 1000 ; i++ ) {
                      double x = 0.0 + i * 0.1;
                      double membership = membership_Stability_excelent(x);
                      double y = ruleActivationMethod_min( degreeOfSupport_1
                           , membership );
                      defuzzify_Stability[i] += ruleAccumulationMethod_max(
                          defuzzify_Stability[i], y );
               }
        }
```

```
// RULE 1 : IF (((Position IS excelent) AND (Direction IS good)) AND
 (Velocity IS good)) AND (Angle IS bad) THEN Stability IS excelent;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_excelent , Direction_good) ,
     Velocity_good) , Angle_bad) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
              double x = 0.0 + i * 0.1;
              double membership = membership_Stability_excelent(x);
              double y = ruleActivationMethod_min( degreeOfSupport_1
                  , membership );
              defuzzify_Stability[i] += ruleAccumulationMethod_max(
                  defuzzify_Stability[i], y );
       }
}

// RULE 1 : IF (((Position IS excelent) AND (Direction IS good)) AND
(Velocity IS bad)) AND (Angle IS good) THEN Stability IS excelent;
 degreeOfSupport_1 = 1.0 * ( ruleConnectionMethod_and
     (ruleConnectionMethod_and(ruleConnectionMethod_and(Position_excelent
     , Direction_good) ,
 Velocity_bad) , Angle_good) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
              double x = 0.0 + i * 0.1;
              double membership = membership_Stability_excelent(x);
              double y = ruleActivationMethod_min( degreeOfSupport_1
                  , membership );
              defuzzify_Stability[i] += ruleAccumulationMethod_max(
                  defuzzify_Stability[i], y );
       }
}

// RULE 1 : IF (((Position IS excelent) AND (Direction IS good)) AND
    (Velocity IS bad))
AND (Angle IS bad) THEN Stability IS excelent;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_excelent , Direction_good) ,
     Velocity_bad) , Angle_bad) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
              double x = 0.0 + i * 0.1;
```

```
                double membership = membership_Stability_excelent(x);
                double y = ruleActivationMethod_min( degreeOfSupport_1
                    , membership );
                defuzzify_Stability[i] += ruleAccumulationMethod_max(
                    defuzzify_Stability[i], y );
        }
}

// RULE 1 : IF (((Position IS excelent) AND (Direction IS bad))
AND (Velocity IS good))
 AND (Angle IS good) THEN Stability IS good;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_excelent , Direction_bad) ,
     Velocity_good) , Angle_good) );
if( degreeOfSupport_1 > 0 ) {
        for (int i = 0 ; i < 1000 ; i++ ) {
                double x = 0.0 + i * 0.1;
                double membership = membership_Stability_good(x);
                double y = ruleActivationMethod_min( degreeOfSupport_1
                    , membership );
                defuzzify_Stability[i] += ruleAccumulationMethod_max(
                    defuzzify_Stability[i], y );
        }
}

// RULE 1 : IF (((Position IS excelent) AND (Direction IS bad))
AND (Velocity IS good))
AND (Angle IS bad) THEN Stability IS good;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_excelent , Direction_bad) ,
     Velocity_good) , Angle_bad) );
if( degreeOfSupport_1 > 0 ) {
        for (int i = 0 ; i < 1000 ; i++ ) {
                double x = 0.0 + i * 0.1;
                double membership = membership_Stability_good(x);
                double y = ruleActivationMethod_min( degreeOfSupport_1
                    , membership );
                defuzzify_Stability[i] += ruleAccumulationMethod_max(
                    defuzzify_Stability[i], y );
        }
}
```

```
// RULE 1 : IF (((Position IS excelent) AND (Direction IS bad)) AND
    (Velocity IS bad))
 AND (Angle IS good) THEN Stability IS good;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_excelent , Direction_bad) ,
     Velocity_bad) , Angle_good) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
              double x = 0.0 + i * 0.1;
              double membership = membership_Stability_good(x);
              double y = ruleActivationMethod_min( degreeOfSupport_1
                  , membership );
              defuzzify_Stability[i] += ruleAccumulationMethod_max(
                  defuzzify_Stability[i], y );
       }
}

// RULE 1 : IF (((Position IS excelent) AND (Direction IS bad)) AND
    (Velocity IS bad))
 AND (Angle IS bad) THEN Stability IS good;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_excelent , Direction_bad) ,
     Velocity_bad) , Angle_bad) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
              double x = 0.0 + i * 0.1;
              double membership = membership_Stability_good(x);
              double y = ruleActivationMethod_min( degreeOfSupport_1
                  , membership );
              defuzzify_Stability[i] += ruleAccumulationMethod_max(
                  defuzzify_Stability[i], y );
       }
}

// RULE 1 : IF (((Position IS good) AND (Direction IS good)) AND
    (Velocity IS good))
AND (Angle IS good) THEN Stability IS verygood;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_good , Direction_good) ,
     Velocity_good) , Angle_good) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
```

```
                double x = 0.0 + i * 0.1;
                double membership = membership_Stability_verygood(x);
                double y = ruleActivationMethod_min( degreeOfSupport_1
                    , membership );
                defuzzify_Stability[i] += ruleAccumulationMethod_max(
                    defuzzify_Stability[i], y );
        }
}

// RULE 1 : IF (((Position IS good) AND (Direction IS good))
AND (Velocity IS good))
AND (Angle IS bad) THEN Stability IS verygood;
 degreeOfSupport_1 = 1.0 * ( ruleConnectionMethod_and
 (ruleConnectionMethod_and(ruleConnectionMethod_and(Position_good ,
     Direction_good) , Velocity_good) , Angle_bad) );
if( degreeOfSupport_1 > 0 ) {
        for (int i = 0 ; i < 1000 ; i++ ) {
                double x = 0.0 + i * 0.1;
                double membership = membership_Stability_verygood(x);
                double y = ruleActivationMethod_min( degreeOfSupport_1
                    , membership );
                defuzzify_Stability[i] += ruleAccumulationMethod_max(
                    defuzzify_Stability[i], y );
        }
}

// RULE 1 : IF (((Position IS good) AND (Direction IS good)) AND
    (Velocity IS bad))
AND (Angle IS good) THEN Stability IS good;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_good , Direction_good) ,
     Velocity_bad) , Angle_good) );
if( degreeOfSupport_1 > 0 ) {
        for (int i = 0 ; i < 1000 ; i++ ) {
                double x = 0.0 + i * 0.1;
                double membership = membership_Stability_good(x);
                double y = ruleActivationMethod_min( degreeOfSupport_1
                    , membership );
                defuzzify_Stability[i] += ruleAccumulationMethod_max(
                    defuzzify_Stability[i], y );
        }
}
```

```
// RULE 1 : IF (((Position IS good) AND (Direction IS good)) AND
    (Velocity IS bad))
 AND (Angle IS bad) THEN Stability IS normal;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_good , Direction_good) ,
     Velocity_bad) , Angle_bad) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
              double x = 0.0 + i * 0.1;
              double membership = membership_Stability_normal(x);
              double y = ruleActivationMethod_min( degreeOfSupport_1
                  , membership );
              defuzzify_Stability[i] += ruleAccumulationMethod_max(
                  defuzzify_Stability[i], y );
       }
}

// RULE 1 : IF (((Position IS good) AND (Direction IS bad)) AND
    (Velocity IS good))
 AND (Angle IS good) THEN Stability IS normal;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_good , Direction_bad) ,
     Velocity_good) , Angle_good) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
              double x = 0.0 + i * 0.1;
              double membership = membership_Stability_normal(x);
              double y = ruleActivationMethod_min( degreeOfSupport_1
                  , membership );
              defuzzify_Stability[i] += ruleAccumulationMethod_max(
                  defuzzify_Stability[i], y );
       }
}

// RULE 1 : IF (((Position IS good) AND (Direction IS bad)) AND
    (Velocity IS good))
AND (Angle IS bad) THEN Stability IS normal;
degreeOfSupport_1 = 1.0 * (
    ruleConnectionMethod_and(ruleConnectionMethod_and
(ruleConnectionMethod_and(Position_good , Direction_bad) ,
    Velocity_good) , Angle_bad) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
```

```
                double x = 0.0 + i * 0.1;
                double membership = membership_Stability_normal(x);
                double y = ruleActivationMethod_min( degreeOfSupport_1
                    , membership );
                defuzzify_Stability[i] += ruleAccumulationMethod_max(
                    defuzzify_Stability[i], y );
        }
}

// RULE 1 : IF (((Position IS good) AND (Direction IS bad)) AND
    (Velocity IS bad))
 AND (Angle IS good) THEN Stability IS bad;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_good , Direction_bad) ,
     Velocity_bad) , Angle_good) );
if( degreeOfSupport_1 > 0 ) {
        for (int i = 0 ; i < 1000 ; i++ ) {
                double x = 0.0 + i * 0.1;
                double membership = membership_Stability_bad(x);
                double y = ruleActivationMethod_min( degreeOfSupport_1
                    , membership );
                defuzzify_Stability[i] += ruleAccumulationMethod_max(
                    defuzzify_Stability[i], y );
        }
}

// RULE 1 : IF (((Position IS good) AND (Direction IS bad)) AND
    (Velocity IS bad))
AND (Angle IS bad) THEN Stability IS bad;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_good , Direction_bad) ,
     Velocity_bad) , Angle_bad) );
if( degreeOfSupport_1 > 0 ) {
        for (int i = 0 ; i < 1000 ; i++ ) {
                double x = 0.0 + i * 0.1;
                double membership = membership_Stability_bad(x);
                double y = ruleActivationMethod_min( degreeOfSupport_1
                    , membership );
                defuzzify_Stability[i] += ruleAccumulationMethod_max(
                    defuzzify_Stability[i], y );
        }
}
```

```
// RULE 1 : IF (((Position IS poor) AND (Direction IS good)) AND
    (Velocity IS good))
AND (Angle IS good) THEN Stability IS verybad;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_poor , Direction_good) ,
     Velocity_good) , Angle_good) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
              double x = 0.0 + i * 0.1;
              double membership = membership_Stability_verybad(x);
              double y = ruleActivationMethod_min( degreeOfSupport_1
                  , membership );
              defuzzify_Stability[i] += ruleAccumulationMethod_max(
                  defuzzify_Stability[i], y );
       }
}

// RULE 1 : IF (((Position IS poor) AND (Direction IS good)) AND
    (Velocity IS good))
AND (Angle IS bad) THEN Stability IS verybad;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_poor , Direction_good) ,
     Velocity_good) , Angle_bad) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
              double x = 0.0 + i * 0.1;
              double membership = membership_Stability_verybad(x);
              double y = ruleActivationMethod_min( degreeOfSupport_1
                  , membership );
              defuzzify_Stability[i] += ruleAccumulationMethod_max(
                  defuzzify_Stability[i], y );
       }
}

// RULE 1 : IF (((Position IS poor) AND (Direction IS good)) AND
    (Velocity IS bad))
AND (Angle IS good) THEN Stability IS verybad;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_poor , Direction_good) ,
     Velocity_bad) , Angle_good) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
```

```
                double x = 0.0 + i * 0.1;
                double membership = membership_Stability_verybad(x);
                double y = ruleActivationMethod_min( degreeOfSupport_1
                    , membership );
                defuzzify_Stability[i] += ruleAccumulationMethod_max(
                    defuzzify_Stability[i], y );
        }
}

// RULE 1 : IF (((Position IS poor) AND (Direction IS good)) AND
    (Velocity IS bad))
AND (Angle IS bad) THEN Stability IS verybad;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_poor , Direction_good) ,
     Velocity_bad) , Angle_bad) );
if( degreeOfSupport_1 > 0 ) {
        for (int i = 0 ; i < 1000 ; i++ ) {
                double x = 0.0 + i * 0.1;
                double membership = membership_Stability_verybad(x);
                double y = ruleActivationMethod_min( degreeOfSupport_1
                    , membership );
                defuzzify_Stability[i] += ruleAccumulationMethod_max(
                    defuzzify_Stability[i], y );
        }
}

// RULE 1 : IF (((Position IS poor) AND (Direction IS bad)) AND
    (Velocity IS good))
 AND (Angle IS good) THEN Stability IS verybad;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_poor , Direction_bad) ,
     Velocity_good) , Angle_good) );
if( degreeOfSupport_1 > 0 ) {
        for (int i = 0 ; i < 1000 ; i++ ) {
                double x = 0.0 + i * 0.1;
                double membership = membership_Stability_verybad(x);
                double y = ruleActivationMethod_min( degreeOfSupport_1
                    , membership );
                defuzzify_Stability[i] += ruleAccumulationMethod_max(
                    defuzzify_Stability[i], y );
        }
}
```

```
// RULE 1 : IF (((Position IS poor) AND (Direction IS bad)) AND
    (Velocity IS good))
 AND (Angle IS bad) THEN Stability IS toobad;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_poor , Direction_bad) ,
     Velocity_good) , Angle_bad) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
              double x = 0.0 + i * 0.1;
              double membership = membership_Stability_toobad(x);
              double y = ruleActivationMethod_min( degreeOfSupport_1
                  , membership );
              defuzzify_Stability[i] += ruleAccumulationMethod_max(
                  defuzzify_Stability[i], y );
       }
}

// RULE 1 : IF (((Position IS poor) AND (Direction IS bad)) AND
    (Velocity IS bad))
 AND (Angle IS good) THEN Stability IS toobad;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_poor , Direction_bad) ,
     Velocity_bad) , Angle_good) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
              double x = 0.0 + i * 0.1;
              double membership = membership_Stability_toobad(x);
              double y = ruleActivationMethod_min( degreeOfSupport_1
                  , membership );
              defuzzify_Stability[i] += ruleAccumulationMethod_max(
                  defuzzify_Stability[i], y );
       }
}

// RULE 1 : IF (((Position IS poor) AND (Direction IS bad)) AND
    (Velocity IS bad))
AND (Angle IS bad) THEN Stability IS toobad;
 degreeOfSupport_1 = 1.0 * (
     ruleConnectionMethod_and(ruleConnectionMethod_and
 (ruleConnectionMethod_and(Position_poor , Direction_bad) ,
     Velocity_bad) , Angle_bad) );
if( degreeOfSupport_1 > 0 ) {
       for (int i = 0 ; i < 1000 ; i++ ) {
```

```
                        double x = 0.0 + i * 0.1;
                        double membership = membership_Stability_toobad(x);
                        double y = ruleActivationMethod_min( degreeOfSupport_1
                            , membership );
                        defuzzify_Stability[i] += ruleAccumulationMethod_max(
                            defuzzify_Stability[i], y );
                }
        }

}

// Defuzzify
void FunctionBlock_tipper::defuzzify() {
        double sum_Stability = 0.0;
        double wsum_Stability = 0.0;
        for (int i = 0; i < 1000 ; i++ ) {
                double x = 0.0 + i * 0.1;
                sum_Stability += defuzzify_Stability[i];
                wsum_Stability += x * defuzzify_Stability[i];
        }
        Stability = wsum_Stability / sum_Stability;
}

// Fuzzify all variables
void FunctionBlock_tipper::fuzzify() {
        Angle_bad = membership_Angle_bad(Angle);
        Angle_good = membership_Angle_good(Angle);
        Direction_bad = membership_Direction_bad(Direction);
        Direction_good = membership_Direction_good(Direction);
        Position_excelent = membership_Position_excelent(Position);
        Position_good = membership_Position_good(Position);
        Position_poor = membership_Position_poor(Position);
        Velocity_bad = membership_Velocity_bad(Velocity);
        Velocity_good = membership_Velocity_good(Velocity);
}

// Membership functions
double FunctionBlock_tipper::membership_Angle_bad(double x) {
        if ( x <= 20.0 )      return 0.0;
        if ( x > 90.0 ) return 1.0;
        if ( x <= 40.0 )      return 0.0 + ( 1.0 - 0.0 ) * ( ( x - 20.0 ) / (
            40.0 - 20.0 ) );
        if ( x <= 90.0 )      return 1.0 + ( 1.0 - 1.0 ) * ( ( x - 40.0 ) / (
            90.0 - 40.0 ) );
}
```

```
double FunctionBlock_tipper::membership_Angle_good(double x) {
        if ( x <= 0.0 ) return 1.0;
        if ( x > 30.0 ) return 0.0;
        if ( x <= 20.0 )      return 1.0 + ( 1.0 - 1.0 ) * ( ( x - 0.0 ) / (
            20.0 - 0.0 ) );
        if ( x <= 30.0 )      return 1.0 + ( 0.0 - 1.0 ) * ( ( x - 20.0 ) / (
            30.0 - 20.0 ) );
}

double FunctionBlock_tipper::membership_Direction_bad(double x) {
        if ( x <= 10.0 )      return 0.0;
        if ( x > 180.0 )      return 1.0;
        if ( x <= 30.0 )      return 0.0 + ( 1.0 - 0.0 ) * ( ( x - 10.0 ) / (
            30.0 - 10.0 ) );
        if ( x <= 180.0 )     return 1.0 + ( 1.0 - 1.0 ) * ( ( x - 30.0 ) / (
            180.0 - 30.0 ) );
}

double FunctionBlock_tipper::membership_Direction_good(double x) {
        if ( x <= 0.0 ) return 1.0;
        if ( x > 20.0 ) return 0.0;
        if ( x <= 20.0 )      return 1.0 + ( 0.0 - 1.0 ) * ( ( x - 0.0 ) / (
            20.0 - 0.0 ) );
}

double FunctionBlock_tipper::membership_Position_excelent(double x) {
        if ( x <= 0.0 ) return 1.0;
        if ( x > 3.0 ) return 0.0;
        if ( x <= 3.0 ) return 1.0 + ( 0.0 - 1.0 ) * ( ( x - 0.0 ) / ( 3.0 -
            0.0 ) );
}

double FunctionBlock_tipper::membership_Position_good(double x) {
        if ( x <= 1.0 ) return 0.0;
        if ( x > 10.0 ) return 0.0;
        if ( x <= 4.0 ) return 0.0 + ( 1.0 - 0.0 ) * ( ( x - 1.0 ) / ( 4.0 -
            1.0 ) );
        if ( x <= 6.0 ) return 1.0 + ( 1.0 - 1.0 ) * ( ( x - 4.0 ) / ( 6.0 -
            4.0 ) );
        if ( x <= 10.0 )      return 1.0 + ( 0.0 - 1.0 ) * ( ( x - 6.0 ) / (
            10.0 - 6.0 ) );
}

double FunctionBlock_tipper::membership_Position_poor(double x) {
```

```
        if ( x <= 8.0 ) return 0.0;
        if ( x > 14.0 ) return 1.0;
        if ( x <= 14.0 )      return 0.0 + ( 1.0 - 0.0 ) * ( ( x - 8.0 ) / (
            14.0 - 8.0 ) );
}

double FunctionBlock_tipper::membership_Stability_bad(double x) {
        if ( x <= 15.0 )      return 0.0;
        if ( x > 25.0 ) return 0.0;
        if ( x <= 20.0 )      return 0.0 + ( 1.0 - 0.0 ) * ( ( x - 15.0 ) / (
            20.0 - 15.0 ) );
        if ( x <= 25.0 )      return 1.0 + ( 0.0 - 1.0 ) * ( ( x - 20.0 ) / (
            25.0 - 20.0 ) );
}

double FunctionBlock_tipper::membership_Stability_excelent(double x) {
        if ( x <= 70.0 )      return 0.0;
        if ( x > 100.0 )      return 0.0;
        if ( x <= 85.0 )      return 0.0 + ( 1.0 - 0.0 ) * ( ( x - 70.0 ) / (
            85.0 - 70.0 ) );
        if ( x <= 100.0 )     return 1.0 + ( 0.0 - 1.0 ) * ( ( x - 85.0 ) / (
            100.0 - 85.0 ) );
}

double FunctionBlock_tipper::membership_Stability_good(double x) {
        if ( x <= 35.0 )      return 0.0;
        if ( x > 60.0 ) return 0.0;
        if ( x <= 50.0 )      return 0.0 + ( 1.0 - 0.0 ) * ( ( x - 35.0 ) / (
            50.0 - 35.0 ) );
        if ( x <= 60.0 )      return 1.0 + ( 0.0 - 1.0 ) * ( ( x - 50.0 ) / (
            60.0 - 50.0 ) );
}

double FunctionBlock_tipper::membership_Stability_normal(double x) {
        if ( x <= 20.0 )      return 0.0;
        if ( x > 40.0 ) return 0.0;
        if ( x <= 30.0 )      return 0.0 + ( 1.0 - 0.0 ) * ( ( x - 20.0 ) / (
            30.0 - 20.0 ) );
        if ( x <= 40.0 )      return 1.0 + ( 0.0 - 1.0 ) * ( ( x - 30.0 ) / (
            40.0 - 30.0 ) );
}

double FunctionBlock_tipper::membership_Stability_toobad(double x) {
        if ( x <= 0.0 ) return 0.0;
        if ( x > 10.0 ) return 0.0;
```

```
        if ( x <= 5.0 ) return 0.0 + ( 1.0 - 0.0 ) * ( ( x - 0.0 ) / ( 5.0 -
            0.0 ) );
        if ( x <= 10.0 )      return 1.0 + ( 0.0 - 1.0 ) * ( ( x - 5.0 ) / (
            10.0 - 5.0 ) );
}

double FunctionBlock_tipper::membership_Stability_verybad(double x) {
        if ( x <= 10.0 )      return 0.0;
        if ( x > 20.0 ) return 0.0;
        if ( x <= 15.0 )      return 0.0 + ( 1.0 - 0.0 ) * ( ( x - 10.0 ) / (
            15.0 - 10.0 ) );
        if ( x <= 20.0 )      return 1.0 + ( 0.0 - 1.0 ) * ( ( x - 15.0 ) / (
            20.0 - 15.0 ) );
}

double FunctionBlock_tipper::membership_Stability_verygood(double x) {
        if ( x <= 55.0 )      return 0.0;
        if ( x > 80.0 ) return 0.0;
        if ( x <= 65.0 )      return 0.0 + ( 1.0 - 0.0 ) * ( ( x - 55.0 ) / (
            65.0 - 55.0 ) );
        if ( x <= 80.0 )      return 1.0 + ( 0.0 - 1.0 ) * ( ( x - 65.0 ) / (
            80.0 - 65.0 ) );
}

double FunctionBlock_tipper::membership_Velocity_bad(double x) {
        if ( x <= 10.0 )      return 0.0;
        if ( x > 30.0 ) return 1.0;
        if ( x <= 20.0 )      return 0.0 + ( 1.0 - 0.0 ) * ( ( x - 10.0 ) / (
            20.0 - 10.0 ) );
        if ( x <= 30.0 )      return 1.0 + ( 1.0 - 1.0 ) * ( ( x - 20.0 ) / (
            30.0 - 20.0 ) );
}

double FunctionBlock_tipper::membership_Velocity_good(double x) {
        if ( x <= 0.0 ) return 1.0;
        if ( x > 15.0 ) return 0.0;
        if ( x <= 10.0 )      return 1.0 + ( 1.0 - 1.0 ) * ( ( x - 0.0 ) / (
            10.0 - 0.0 ) );
        if ( x <= 15.0 )      return 1.0 + ( 0.0 - 1.0 ) * ( ( x - 10.0 ) / (
            15.0 - 10.0 ) );
}


// Print
void FunctionBlock_tipper::print() {
```

```
        printf("Function block tipper:\n");
        printf("       Input %20s : %f\n", "Angle" , Angle);
        printf("             %20s : %f\n", "Angle_bad" , Angle_bad);
        printf("             %20s : %f\n", "Angle_good" , Angle_good);
        printf("       Input %20s : %f\n", "Direction" , Direction);
        printf("             %20s : %f\n", "Direction_bad" , Direction_bad);
        printf("             %20s : %f\n", "Direction_good" , Direction_good);
        printf("       Input %20s : %f\n", "Position" , Position);
        printf("             %20s : %f\n", "Position_excelent" ,
            Position_excelent);
        printf("             %20s : %f\n", "Position_good" , Position_good);
        printf("             %20s : %f\n", "Position_poor" , Position_poor);
        printf("      Output %20s : %f\n", "Stability" , Stability);
        printf("       Input %20s : %f\n", "Velocity" , Velocity);
        printf("             %20s : %f\n", "Velocity_bad" , Velocity_bad);
        printf("             %20s : %f\n", "Velocity_good" , Velocity_good);
}

// Reset output
void FunctionBlock_tipper::reset() {
        for( int i=0 ; i < 1000 ; i++ ) { defuzzify_Stability[i] = 0.0; }
}

int main() {
        // Create function blocks
        FunctionBlock_tipper tipper;

        // Parse input
tipper.Angle = 90 ;
tipper.Direction = 90 ;
 tipper.Position = 1 ;
tipper.Velocity = 4.510434;

        // Calculate
        tipper.calc();

        // Show results
        tipper.print();
}
```

Chapter

Appendix C

This section will describe the parameters which are participated in FFAR.

C.1

The codes of same direction to previous hop function are shown here.

Listing C.1: *Same direction function*

```
double
 AODV :: direction (Packet *p )

{
    double param;
    double result ;
    double x1;
    double x2;
    double y1;
    double y2;
    double sum;
    struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
     x1 = rq-> dx ;
     y1= rq-> dy;
     iNode = (MobileNode *) (Node::get_node_by_address (index) );
     x2=iNode->dX();
     y2=iNode->dY();
     sum=(sqrt((pow(x1,2.0) +pow(y1,2.0)))*(sqrt((pow(x2,2.0)
         +pow(y2,2.0))))) ;
if ( sum!=0)
   {
      param= (((x1 *x2)+(y1 *y2))/ sum);
           if (param <=1 && param >=-1)
```

```
        {
          result =(((acos (param) )* 180.0 )/ PI);
        }
   }
      return abs(result);
  }
```

C.2

These codes explian the "same direction compared to source and destination" function.

Listing C.2: *Same angle function*

```
double
 AODV :: angle (Packet *p )

{

    double param1;
    double result1 ;
    double param2;
    double result2 ;
    double x1;
    double x2;
    double x3;
    double y1;
    double y2;
    double y3;
    double sum1;
    double sum2;
    struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
    iNodeDst = (MobileNode *) (Node::get_node_by_address (rq-> rq_dst) );
    iNodeSrc = (MobileNode *) (Node::get_node_by_address (rq-> rq_src) );
     x1 = iNodeSrc->dX() ;
     y1=  iNodeSrc->dY();
     x3 = iNodeDst->dX() ;
     y3=  iNodeDst->dY();
     iNode = (MobileNode *) (Node::get_node_by_address (index) );
     x2=iNode->dX();
     y2=iNode->dY();


sum1=(sqrt((pow(x1,2.0) +pow(y1,2.0)))*(sqrt((pow(x2,2.0) +pow(y2,2.0))))) ;
if ( sum1!=0)
```

```
   {
       param1= (((x1 *x2)+(y1 *y2))/ sum1);
          if (param1 <=1 && param1 >=-1)
         {
           result1 =(((acos (param1) )* 180.0 )/ PI);
         }
   }


sum2=(sqrt((pow(x2,2.0) +pow(y2,2.0)))*(sqrt((pow(x3,2.0) +pow(y3,2.0))))) ;
if ( sum2!=0)
   {
       param2= (((x3 *x2)+(y3 *y2))/ sum2);
            if (param2 <=1 && param2 >=-1)
         {
           result2 =(((acos (param2) )* 180.0 )/ PI);
         }
   }

            return (abs(result2)) + (abs(result1)) ;
 }
```

C.3

The below codes illustrates the position function which is applied in FFAR. The functions of linePointDist, dot, cross and distance are cited from [66].

Listing C.3: *Position function*

```
double
 AODV :: nodePosition (Packet *p )

{
    double x1;
    double x2;
    double xm;
    double y1;
    double y2;
    double ym;
    double X ;
    MobileNode *iNodeDst;
    MobileNode *iNodeSrc;
     struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
     iNodeDst = (MobileNode *) (Node::get_node_by_address (rq-> rq_dst) );
```

```
    iNodeSrc = (MobileNode *) (Node::get_node_by_address (rq-> rq_src) );
    iNode = (MobileNode *) (Node::get_node_by_address (index) );
    x1 = iNodeSrc->dX() ;
    y1=  iNodeSrc->dY();
    x2 = iNodeDst->dX() ;
    y2=  iNodeDst->dY();
    xm=iNode->dX();
    ym=iNode->dY();
    int V []= {x1,y1};
    int W []= {x2,y2};
    int  D [] = {xm,ym};
    X = linePointDist(V,W,D,false) ;
return X;
 }

*****************We put these functions in quotation *****************
``int
 AODV :: dot (int A [], int B [], int C [])

{

     int AB [2];
     int BC [2];
        AB[0] = B[0]- A[0];
      AB[1] = B[1]-A[1];
         BC[0] = C[0]-B[0];
       BC[1] = C[1]-B[1];
       int dot = AB[0] * BC[0] + AB[1] * BC[1];
       return dot;
   }

int
 AODV :: cross (int A [], int B [], int C []){
       int AB [2];
     int AC [2];

       AB[0] = B[0]-A[0];
       AB[1] = B[1]-A[1];
       AC[0] = C[0]-A[0];
       AC[1] = C[1]-A[1];
       int cross = AB[0] * AC[1] - AB[1] * AC[0];
       return cross;
   }
```

```
double
 AODV :: distance (int A [], int B []){
       int d1 = A[0] - B[0];
       int d2 = A[1] - B[1];
       return sqrt(d1*d1+d2*d2);
   }

double
 AODV :: linePointDist (int A [], int B [], int C [], bool isSegment)
 {
      if (distance(A,B)!=0)
     {
       double dist = cross(A,B,C) / distance(A,B);
       if(isSegment){
           int dot1 = dot(A,B,C);
           if(dot1 > 0)return distance(B,C);
           int dot2 = dot(B,A,C);
           if(dot2 > 0)return distance(A,C);
       }

       return abs(dist);
  }
     else
        return 1000;
   }''
```

C.4

The relative velocity function are described by these codes.

Listing C.4: *Velocity function*

```
double
 AODV :: nodeVelocity (Packet *p )

{
    MobileNode *iNodeDst;
    MobileNode *iNodeSrc;
    double v1;
    double v3;
    double v2;
    struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
    iNodeDst = (MobileNode *) (Node::get_node_by_address (rq-> rq_dst) );
    iNodeSrc = (MobileNode *) (Node::get_node_by_address (rq-> rq_src) );
```

```
    iNode = (MobileNode *) (Node::get_node_by_address (index) );
    v1=iNodeDst->speed();
    v3=iNodeSrc->speed();
    v2= iNode->speed();
    return (abs(v1-v2) + abs(v2-v3) );

}
```

Chapter D
Appendix D

The below codes explain the TCL scenario which is applied in this report. The basis of my codes is cited from [65], [67], [68], INTERNET and NS2 forums, then it is modified to satisfy my design.

D.1

Listing D.1: *TCL*

```
# Define options
set val(chan)         Channel/WirelessChannel  ;# channel type
set val(prop)         Propagation/TwoRayGround ;# radio-propagation model
set val(netif)        Phy/WirelessPhy          ;# network interface type
set val(mac)          Mac/802_11               ;# MAC type
set val(ifq)          Queue/DropTail/PriQueue  ;# interface queue type
set val(ll)           LL                       ;# link layer type
set val(ant)          Antenna/OmniAntenna      ;# antenna model
set val(ifqlen)       50                       ;# max packet in ifq
set val(nn)         20               ;# number of mobilenodes
set val(rp)           AODV                     ;# routing protocol
set val(x)            2000                      ;# X dimension of topography
set val(y)            2000                    ;# Y dimension of topography
set val(stop)        1000                     ;# time of simulation endset ns
    [new Simulator]


LL set mindelay_            50us
LL set delay_               25us
LL set bandwidth_           0       ;# not used
LL set off_prune_           0       ;# not used
LL set off_CtrMcast_        0       ;# not used

Agent/Null set sport_       0
```

```
Agent/Null set dport_        0

Agent/CBR set sport_         0
Agent/CBR set dport_         0

Agent/TCPSink set sport_     0
Agent/TCPSink set dport_     0

Agent/TCP set sport_         0
Agent/TCP set dport_         0
Agent/TCP set packetSize_    1460
Queue/DropTail/PriQueue set Prefer_Routing_Protocols 1

# unity gain, omni-directional antennas
# set up the antennas to be centered in the node and 1.5 meters above it
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0

# Initialize the SharedMedia interface with parameters to make
# it work like the 914MHz Lucent WaveLAN DSSS radio interface
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11
Phy/WirelessPhy set RXThresh_ 3.652e-10
Phy/WirelessPhy set Rb_ 2*1e6
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 914e+6
Phy/WirelessPhy set L_ 1.0

set ns_ [new Simulator]
#create trace file and nam file
set tracefile [open aodv10.tr w]
$ns_ trace-all $tracefile
set namtrace [open out.nam w]
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)
# set up topography object
set topo [new Topography]
 $topo load_flatgrid $val(x) $val(y)
# Create God
create-god $val(nn)
# For model 'TwoRayGround'
      set dist(5m) 7.69113e-06
      set dist(9m) 2.37381e-06
```

```
      set dist(10m) 1.92278e-06
      set dist(11m) 1.58908e-06
      set dist(12m) 1.33527e-06
      set dist(13m) 1.13774e-06
      set dist(14m) 9.81011e-07
      set dist(15m) 8.54570e-07
      set dist(16m) 7.51087e-07
      set dist(20m) 4.80696e-07
      set dist(25m) 3.07645e-07
      set dist(30m) 2.13643e-07
      set dist(35m) 1.56962e-07
      set dist(40m) 1.56962e-10
      set dist(45m) 1.56962e-11
      set dist(50m) 1.20174e-13
      Phy/WirelessPhy set CSThresh_ $dist(40m)
      Phy/WirelessPhy set RXThresh_ $dist(40m)

set chan [new $val(chan)]
# configure the nodes
 $ns_ node-config -adhocRouting $val(rp) \
                -llType $val(ll) \
                -macType $val(mac)\
                -ifqType $val(ifq) \
                -ifqLen $val(ifqlen) \
                -antType $val(ant) \
                -propType $val(prop) \
                -phyType $val(netif) \
                -topoInstance $topo \
                -agentTrace ON \
                -routerTrace ON \
                -macTrace ON \
                -movementTrace ON \
                -channel $chan
## Creating node objects...
for {set i 0} {$i < $val(nn) } { incr i } {
       set node_($i) [$ns_ node]
       $node_($i) random-motion 0 ;# disable random motion
     }
set opt(seed) 0.1
      set a [ns-random $opt(seed)]
      set i 0
      while {$i < 5} {
      incr i
      }
#setting animation rate
```

```
$ns_ at 0.0 "$ns_ set-animation-rate 20.0ms"
# set plan nodes
#30 defines the size for nam
 for {set i 0} {$i < $val(nn)} { incr i } {
 $ns_ initial_node_pos $node_($i) 30
}

# connect 2 nodes
  set udp1 [$ns_ create-connection UDP $node_(6) LossMonitor $node_(8) 0]
  $udp1 set fid_ 1
  set cbr1 [$udp1 attach-app Traffic/CBR]
  $cbr1 set packetSize_ 1000
  $cbr1 set interval_ .07
  $ns_ at 0.0 "$cbr1 start"
  $ns_ at 1000.0 "$cbr1 stop"

# connect 2 nodes
  set udp1 [$ns_ create-connection UDP $node_(0) LossMonitor $node_(2) 0]
  $udp1 set fid_ 1
  set cbr1 [$udp1 attach-app Traffic/CBR]
  $cbr1 set packetSize_ 1000
  $cbr1 set interval_ .07
  $ns_ at 0.0 "$cbr1 start"
  $ns_ at 1000.0 "$cbr1 stop"


# connect 2 nodes
  set udp1 [$ns_ create-connection UDP $node_(3) LossMonitor $node_(4) 0]
  $udp1 set fid_ 1
  set cbr1 [$udp1 attach-app Traffic/CBR]
  $cbr1 set packetSize_ 1000
  $cbr1 set interval_ .07
  $ns_ at 0.0 "$cbr1 start"
  $ns_ at 1000.0 "$cbr1 stop"

# connect 2 nodes
  set udp1 [$ns_ create-connection UDP $node_(6) LossMonitor $node_(7) 0]
  $udp1 set fid_ 1
  set cbr1 [$udp1 attach-app Traffic/CBR]
  $cbr1 set packetSize_ 1000
  $cbr1 set interval_ .07
  $ns_ at 0.0 "$cbr1 start"
  $ns_ at 1000.0 "$cbr1 stop"

# connect 2 nodes
```

```
  set udp1 [$ns_ create-connection UDP $node_(5) LossMonitor $node_(0) 0]
  $udp1 set fid_ 1
  set cbr1 [$udp1 attach-app Traffic/CBR]
  $cbr1 set packetSize_ 1000
  $cbr1 set interval_ .07
  $ns_ at 0.0 "$cbr1 start"
  $ns_ at 1000.0 "$cbr1 stop"

# Telling nodes when the simulation ends
 for {set i 0} {$i < $val(nn) } { incr i } {
    $ns_ at $val(stop) "$node_($i) reset";
    }
    $ns_ at $val(stop) "stop"

# Ending nam and the simulation
  $ns_ at $val(stop) "$ns_ nam-end-wireless $val(stop)"
  $ns_ at 1000.0 "puts \"end simulation\"; $ns halt"
     puts "Starting Simulation"
#stop procedure:
proc stop { } {
   global ns_ tracefile namtrace
   $ns_ flush-trace
      close $tracefile
      close $namtrace
      exec nam out.nam &
      exit 0
    }
   $ns_ at 1000 "stop"
$ns_ run
```

Chapter E
Appendix E

This part presents the mobility scenario which is generated by BonnMotion.

E.1

Listing E.1: *Mobility scenario by BonnMotion*

```
$ns_ at 9067.033118438478 "$node_(10) setdest 259.871130474348 650.0
    0.7705007778923962"
$ns_ at 9112.45812124735 "$node_(10) setdest 254.87113047434798 650.0
    0.7790790898130234"
$ns_ at 9118.87595489913 "$node_(10) setdest 199.87113047434798 650.0
    0.8572696045126093"
$ns_ at 9183.033134519734 "$node_(10) setdest 194.87113047434798 650.0
    1.2495308161951517"
$ns_ at 9187.034636471662 "$node_(10) setdest 184.87113047434798 650.0
    0.7796770929389434"
$ns_ at 9199.86045896993 "$node_(10) setdest 89.87113047434798 650.0
    0.9576805363563702"
$ns_ at 9299.058465396416 "$node_(10) setdest 79.87113047434798 650.0
    1.2754190426932142"
$ns_ at 9306.89902576541 "$node_(10) setdest 64.87113047434798 650.0
    1.0182450487830874"
$ns_ at 9321.63025379144 "$node_(10) setdest 44.87113047434798 650.0
    1.2885906888567502"
$ns_ at 9337.151086103013 "$node_(10) setdest 39.87113047434798 650.0
    0.9067072526549765"
$ns_ at 9342.665545136355 "$node_(10) setdest 24.87113047434798 650.0
    1.2828699394215537"
$ns_ at 9354.35807883682 "$node_(10) setdest 10.0 650.0 1.1781006351303598"
$ns_ at 9366.981050110155 "$node_(10) setdest 10.0 655.128869525652
    1.178100635130306"
```

```
$ns_ at 9371.334557249284 "$node_(10) setdest 10.0 675.128869525652
    0.7365729383909928"
$ns_ at 9398.487333244135 "$node_(10) setdest 10.0 685.128869525652
    0.7555200734346452"
$ns_ at 9411.723248941966 "$node_(10) setdest 10.0 695.128869525652
    0.8940949655087873"
$ns_ at 9422.907743191197 "$node_(10) setdest 10.0 730.128869525652
    0.8358929818771894"
$ns_ at 9464.779131944097 "$node_(10) setdest 10.0 750.128869525652
    1.142973753818449"
$ns_ at 9482.277346520927 "$node_(10) setdest 10.0 795.128869525652
    1.1702191683992649"
$ns_ at 9520.731681584977 "$node_(10) setdest 10.0 810.128869525652
    0.9387694307406182"
$ns_ at 9536.710045929634 "$node_(10) setdest 10.0 820.128869525652
    1.102893506752886"
$ns_ at 9545.777104479504 "$node_(10) setdest 10.0 835.128869525652
    1.0666440401762602"
$ns_ at 9559.83990278429 "$node_(10) setdest 10.0 865.128869525652
    0.5705871402310542"
```