Alexander TEI of Thessaloniki
School of Technological
Applications Department of
Informatics

Studying Routing Issues in VANETs by Using NS-3

Bachelor Thesis on Informatics

by
Christos Profentzas

Thesis supervisor
Dr. Periklis Chatzimisios

Alexander Technological Educational Institute of Thessaloniki
Department of Informatics

A.T.E.I. of Thessaloniki
P.O. Box 141
GR -547 00 Thessaloniki,
Macedonia, Greece

November 2012

Acknowledgements

This research project would not have been possible without the support of many people. The author wishes to express his gratitude to his supervisor, Assistant Professor **Periklis Chatzimisios** (Alexander TEI of Thessaloniki, Greece) and Assistant Professor **Gennaro Boggia** (Politecnico di Bari, Italy) who was abundantly helpful and offered invaluable assistance, support and guidance. Deepest gratitude are also due to the members of the supervisory committee, Assistant Professor **Luigi Alfredo Grieco** and Ph.D Student **Giuseppe Piro** without whose knowledge and assistance this study would not have been successful. Special thanks also to all group members of Telematics Lab at the Electrical & Electronics Engineering Department of Politecnico di Bari, for sharing the literature, invaluable assistance and laboratory facilities. The author would also like to convey thanks to the Office of Erasmus Program and Faculty of Alexander Technological Educational Institution of Thessaloniki for providing the financial means.

Abstract

A Vehicular Ad-hoc Network (VANET) is a system of nodes (vehicles) that are being connected with each other by wireless technologies. Usually the nodes are moving with very high speeds and, thus, the topology is unpredictable and frequently changing. Such networks can be stand alone and making paths along vehicles or may be connected by an infrastructure internet. System characteristics such as multi-hop paths, node mobility, large network size combined with device heterogeneity, bandwidth and unlimited battery power make the design of routing protocols a major challenging. At the last year many routing protocols have been proposed for VANETs. The most of them has been implemented using Network Simulator 2 (NS-2). Network Simulator 3 (NS-3) that is going to replace NS-2 has not employed so far for implementing any routing protocol protocols for VANETs. The available routing protocols are Ad-Hoc On Demand Distance Vector Routing (AODV), Optimized Link State Routing (OLSR), Destination-Sequenced Distance-Vector (DSDV) and Dynamic Source Routing (DSR) that have been proposed for Mobile Ad-hoc Networks (MANETs). In this thesis, we implement and study the performance of a Position-Based Routing (PBR) protocol for VANETs by employing NS-3 simulator and we finally compare its performance against AODV and OLSR routing protocols.

Contents

List of Figures

Listings

Introduction

During the recent years Vehicular Ad-hoc Networks (VANETs) attract a significant amount of interest in the research community. VANETs can establish connections between Vehicle To Vehicle (V2V) and Vehicle to Infrastructure (V2I), so a great number of applications could be applied. These applications include safety and entertainment applications that require high availability of the communication system. For example in the case of an accident a big amount of messages is being transmitted into a large scale area. Furthermore the topology is quickly changing during the transmission and vehicles are moving with high speed. In order to make the communication of vehicles possible, a stable routing protocol is needed to ensures that the messages is being transmitted efficiently and correctly. So the design of routing protocols is an emergency research area in VANET that have to deal with this particulars problems of frequently topology changing and quickly movements of nodes.

Among the ad-hoc routing protocols, several Topology-Based Routing Protocols (TBRPs) have been proposed, such as Ad-Hoc On Demand Distance Vector (AODV) and Optimized Link State Routing (OLSR). These protocols have make great success into Mobility Ad-hoc NETworks (MANETs) but in the VANETs shows poor performance. Position-Based Routing Protocols (PBRPs) eliminate some of the limitation of TBRPs by using the information of physical position of nodes. By now, many PBRP algorithms have been proposed. A well promising area of algorithms is the extensions of Greedy Perimeter Stateless Routing(GPSR) [32].

GPSR finds the next hop using the position between neighbour and destination which have been taken by Global Position System (GPS) and local services, the most closest node to destination is the next hop. Because a possibility to failure to find a path, GPSR needs to extent his capabilities in order to cope with this problem, in this thesis represented an extension of GPRS algorithm that finds the

path considering the number of neighbours of every node. The implementation is into Network Simulator 3 (NS-3) and being compared with OLSR and AODV that already being implemented in NS-3.

Following the introduction, the rest of the thesis is organised as follow: In chapter one is being introducing VANET technology focusing into routing protocols. Chapter two presents the NS3 in details. Chapter three presents other simulation tools that have being used. Chapter four describes the implementation of AODV, OLSR, Destination-Sequenced Distance-Vector (DSDV) and Dynamic Source Routing (DSR) into NS-3 ending with Extended-GPSR (E-GPSR) implementation. Chapter five is showing the performance results of several scenarios into NS-3. Chapter six has the conclusion and the future work of theses. In the end Appendixes have been placed that have particular information of tool being used.

Chapter 1

Introduction to VANETs

1.1 VANETs in General

VANETs are considered as a special type of MANETs, in which each node is a vehicle (i.e. car,bus,truck) [5]. This kind of networks have face up new challenges. They are characterized by very high node mobility and topology changing that dependent by means of wireless technologies. These features make VANETs very prone to transmission errors, topology changes and intermittent connectivity. This is an effect of high moving speed of nodes and highly dynamic operating environments. So the main goals for VANETs are to achieve high packet delivery rates and low packet latency.

At the last years, car manufacturers have gave to vehicles the ability to generate and analyse large amounts of data, although this data associated only with a single vehicle. With the VANET technologies vehicles have the new ability to connect each over as well as to network infrastructures that companies in the last year have equipped into the highways (also a hybrid combination of them is possible). Roadside infrastructure can also be used as a gateway to the Internet. Thus data and context information can be collected, stored and processed somewhere (cloud computing). Communication can be either done directly between vehicles as one-hop communication or vehicles can retransmit messages thereby enabling multi-hop communication. Furthermore packets can be transited with unicast or multi-cast way [13].

Despite of the difficult challenges VANETs represent an emerging wireless technology, allowing efficient communication among vehicles and fixed devices positioned along the street with a very promising area of safety, traffic control and

user applications. Ongoing research is exploring novel protocol stacks and network architectures to efficiently afford these challenging issues [40].

The area of ad-hoc networks consists for many years and has a large variety of technologies (e.g. MANETs). Although VANET has some distinguishing feature which make it more challenging from other technologies being summarized as following.

- High dynamic topology: Since the high speeds of vehicles and the large area that its cover the topology is very frequently changing.
- Frequently disconnected network: The often topology changes has the result that a node is become frequently out of range of the networks. This problem is also caused by node density changing.
- Unlimited battery power and storage: Every node is a vehicle so the power of every device is being provided by fuels of the car.
- On board sensors: Nodes consists of sophisticated sensors which provide very useful information such as GPS which gives location informations.

1.2 Types of communication in VANETs

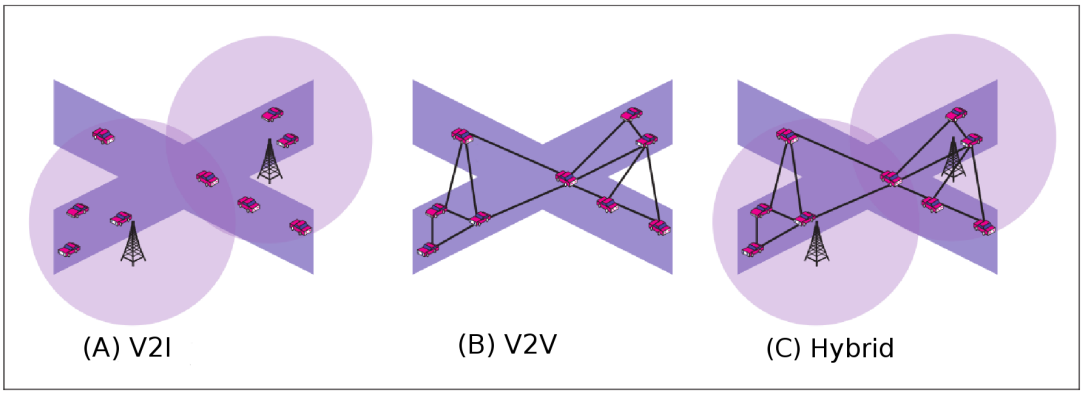

Figure 1.1: Types of Communication in VANETs

There are three possible types of communication that could be established within a VANET: Vehicle to Infrastructure (V2I), Vehicle to Vehicle (V2V) and a hybrid combination of them [5].

1.2.1 V2V

V2V is an ad-hoc network among vehicles (car, bus, track etc) which allow vehicles send data to each over. Using V2V communication, a vehicle can detect the position and movement of other vehicles up to a quarter of a mile away. Vehicles being equipped with a simple antenna, a computer chip and GPS technology; they will know the position of other vehicles and will communicating directly with them. In this way vehicles could share informations about blond spots, car accidents and road condition. Thus vehicles can anticipate and react to dangerously driving situations informing drivers. If the driver does not respond to the alerts, the vehicle could act itself and stop to a safe point avoiding a collision.

The V2V systems is equipped with long range scanning sensor for adaptive cruise control, forward vision sensors for object detection, mid-range blind spot detection sensors and long-range lane change assist sensors. The vehicle alerts the driver for vehicles in blind spots with a steady amber light in the side mirror. If the turn signal is activated, a flashing amber light and gentle seat vibration on the side notifies the driver of a potentially dangerous situation. The vibration is enough to get our attention but not a sudden distraction. In addition, V2V technology can warn the driver when vehicles ahead, regardless of lane, are stopped or travelling much slower or any vehicle ahead brakes hard, allowing the driver to brake or change lanes as needed.

In general V2V is a research for applications linked to road safety but also to include entertainment applications. The technology draws on several disciplines, including transport engineering, electrical engineering, automotive engineering and computer science.

1.2.2 V2I

V2I is an wireless network (wifi ,wimax or cellular) that offers a communication between nodes (vehicles) and roadside bases (access points). So for this kind of communication needs a integrated communication system under the roads called Vehicle Infrastructure Integration (VII).

VII is an initiative fostering research and applications for a series of technologies directly linking road vehicles to their physical surroundings. This technology draws on several disciplines, including transport engineering, electrical engineering, automotive engineering and computer science.

VII specifically covers road transport although similar technologies are in place or under development for other modes of transport. Planes, for example, use ground-based beacons for automated guidance, allowing the autopilot to fly the plane without human intervention. In highway engineering, improving the safety of a roadway can enhance overall efficiency. VII targets improvements in both safety and efficiency.

Finally VII is that branch of engineering, which deals with the study and implementation of a series of techniques to achieve communication among vehicles and infrastructure bases in order to improve road safety.

1.2.3 Hybrid architecture

Hybrid architecture consists of both infrastructure networks and ad-hoc networks. That means that vehicles has the ability to make a connection and exchange data and informations with a roadside base (access point) or vehicle which also might be connect with other vehicles or bases. In this way VANET could take the benefits from both V2V and V2I.

1.3 Standards for VANETs

The U.S. Federal Communication Commission has allocated 75 MHz spectrum at 5.9 GHz exclusively for V2V and V2I communications in North America. Process of frequency allocation in Europe is considerably more complex, since all European countries and their authorities are involved. In August 2008, after spectrum requirements have been analysed and frequency regulation and redeployment have been worked out, the Commission of European Communities has carried out the decision on the harmonized use of radio spectrum in the 5875-5905 MHz frequency band for safety related applications of Intelligent Transport Systems (ITS) [15].

According to nominal carrier frequency allocation, defined by European Telecommunications Standards Institute (ETSI) in its Harmonized Standard EN 302 571 [11], the whole 75 MHz frequency band is divided in seven channels, each of which is 10 MHz wide (Figure 1.2).

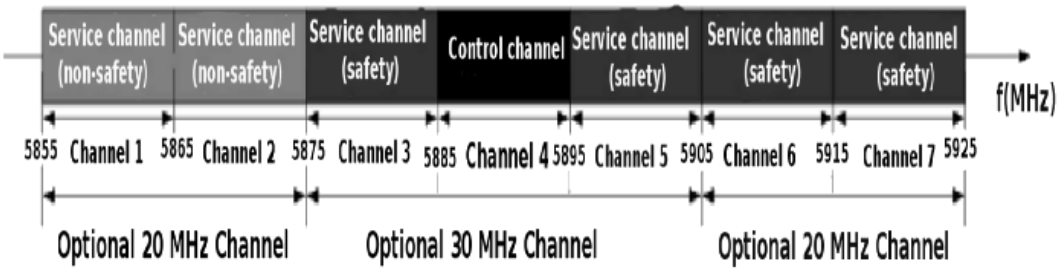

Figure 1.2: Channels in 75 MHz Frequency Band

The IEEE 1609 Family [1] of Standards for Wireless Access in Vehicular Environments (WAVE): 802.11p [3] defines a communication architecture and a standardized set of interfaces that collectively enable secure V2V and V2I wireless communications. The objective of communication architecture is to provide communication facilities for a wide range of applications. The architecture is

usually based on the layered Open Systems Interconnection (OSI) model, where each level provides certain functions (figure 1.3).

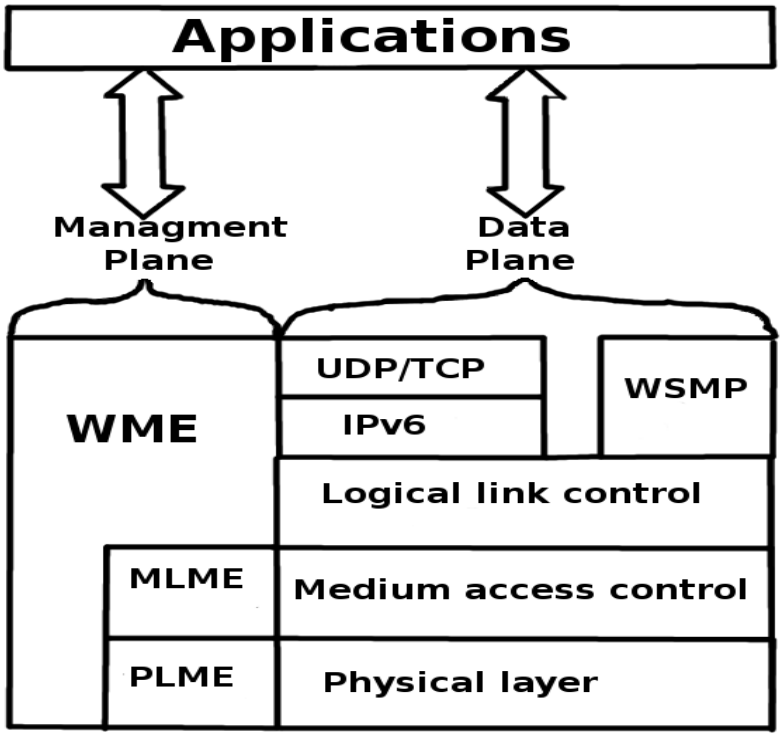

Figure 1.3: WAVE Protocol Stack

WAVE communication being operated into a bandwidth of 10 MHz center at 5.8 GHz, allowing physical transmission rates from 3 to 27 Mbps. The WAVE architecture has introduced two different channels, the Control CHannel (CCH) and the Service CHannel (SCH), the first dedicated to safety and traffic control messages and second to user applications. WAVE specifications allow physical transmission rates from 3 to 27 Mbps and impose to use the spectrum 5.850-8.925 GHz, divided in seven orthogonal channels of 10 MHz each. Six of them, which is SCHs, can be used for traffic control and user applications, whereas a dedicated CCH must be reserved exclusively for exchanging management and control messages. Upper layers support functions required by applications such as data transfer, resource management, system configuration and notification. In partic-

ular the services and interfaces of the WAVE Resource Manager application are specified in standard IEEE 1609.1 [1]. It defines numerous message formats (i.e., command, status and request) and the appropriate responses to those messages, as well as data storage formats that must be used by applications to communicate between architecture components. Networking services, defined in IEEE 1609.3 standard [1], consist of data and management components. Data plane components include Physiacl layer, Medium Access Layer (MAC), Logical Link Control (LLC), User Datagram Protocol (UDP) and Transmission Control Protocol (TCP), as well as WAVE Short Messages Protocol (WSMP). Management services cover application registration, as well as monitoring of channel usage and received channel power indicator.

1.3.1 Physical layer

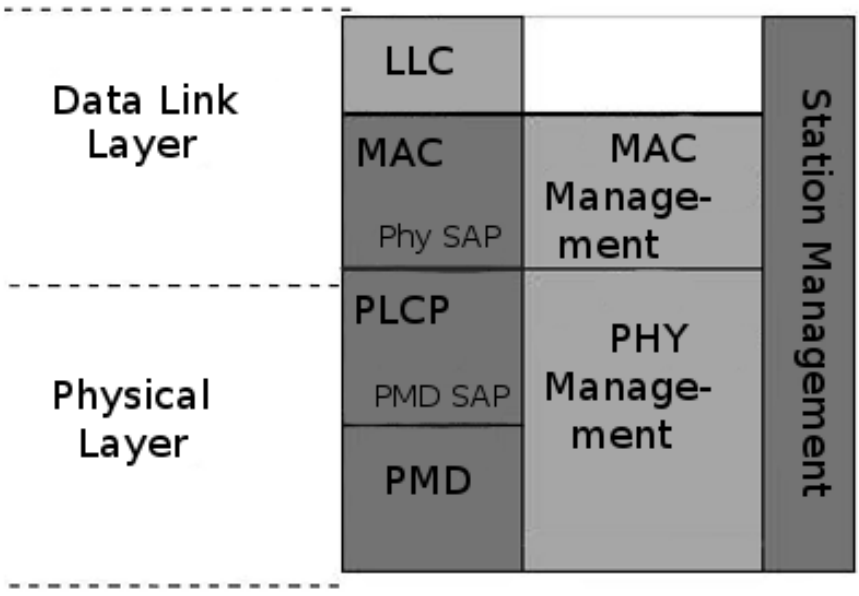

Figure 1.4: PHY Layer in IEEE 802.11

Physical layer (PHY) is being defined in standard IEEE 802.11 [2] consists of Physical Medium Dependent (PMD) sub-layer and Physical Layer Convergence Protocol (PLCP) sub-layer. The PMD sub-layer specifies signal build up parameters, such as modulation and channel coding and convert signal into analogue

Table 1.1: Differences between IEEE 802.11a and IEEE 802.11p

Parameters	IEEE 802.11A	IEEE 802.11p	Changes
Bit rate(Mb/s)	6, 9, 12, 18, 24, 36, 48, 54	3, 4.5, 6, 9, 12, 18, 24, 27	Half
Modulation mode	BPSK, QPSK, 16QAM, 64QAM	BPSK, QPSK, 16QAM, 64QAM	No change
Code rate	1/2,2/3,3/4	1/2,2/3,3/4	No change
Number of sub-carries	52	52	No change
Symbol of sub-carries	4s	8S	Double
Guard duration	0.8s	1.6S	Double
FFT period	3.2s	6.4S	Double
Preamble duration	16s	32S	Double
Subcarrier spacing	0.3125	0.15625	Half

form with regard to specific PHY type. PLCP deals with differences among various PHYs and ensures that the MAC layers receive packets of common format, independently of particular PMD sub-layer.

For transmission an OFDM technique with 64 sub-carriers being used. To design easy-to-implement transmission system, 64 carriers are defined but only the inner 52 carriers are utilized. Out of 52 carriers 48 contains the actual data and 4 sub-carriers, called pilot sub-carriers, transmit a fixed pattern, used to disregard frequency and phase offset at the receiver side. Each of 48 data sub-carriers can be modulated with BPSK, QPSK, 16QAM or 64QAM. In combination with different coding rates, this leads to a nominal data rate of 6 to 54 Mb/s if full clocked mode with 20 MHz bandwidth is used. IEEE 802.11p [2] uses the half clocked mode with 10 MHz bandwidth, in order to make signal more robust against fading, resulting in corresponding data rate reduction. Some other differences between IEEE 802.11a and IEEE 802.11p due to reduced sampling rate are emphasized in table 1.1.

1.3.2 MAC layer

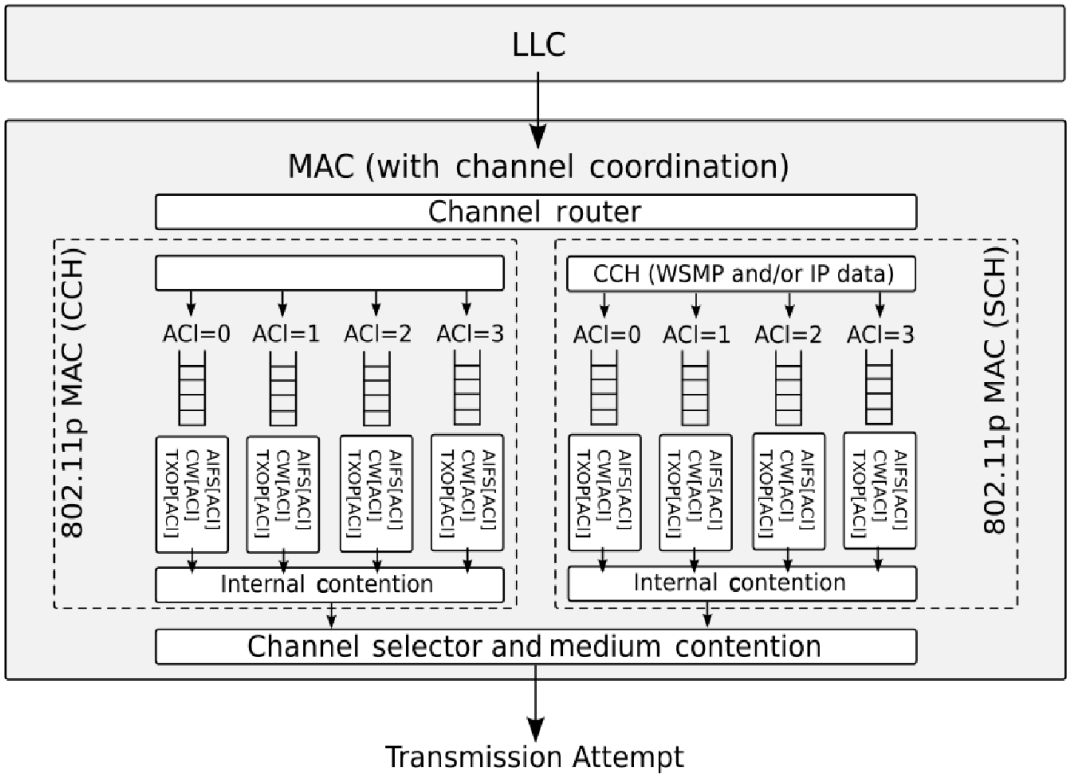

Figure 1.5: MAC Layer

MAC layer has two main channels, one is SCH, can be used for traffic control and user applications and the other is CCH must be reserved exclusively for exchanging management and control messages. The MAC layer, based on the Enhanced Distributed Channel Access (EDCA), supports services differentiation among four traffic categories (i.e. voice, video, best-effort and background) by defining for each of them a dedicated queue and specific Contention Windows (CW) and Arbitration Inter-Frame Spaces (AIFS) parameters.

1.3.2.1 Problems

In VANET MAC layer faces up the follow major problems: transmission collision, hidden terminal problem and exposed terminal problem [22] .

Transmission collision Because of a shared communication medium, when two nodes (A,B) try to transmit at the same time to a third terminal (C) a collision it's appears (figure 1.6).

Figure 1.6: Transmission Collision Problem

Hidden node In this problem nodes could be hidden from each other. For example there are a no connection with A and B in which could be done the communicate but a different third node could a establish the connection from each other. MAC layer has to make this connection possible (figure 1.7).

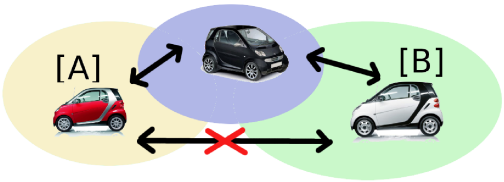

Figure 1.7: Hidden Node Problem

Exposed node Here a node (S2) is prevented from sending packets to other nodes (R2) due to a neighbouring transmitter (figure 1.8).

Figure 1.8: Exposed Node Problem

1.3.2.2 MAC protocols

To solve transmission collisions, hidden node and exposed node problem IEEE 802.11 has proposed several protocols such as Request To Send/Clear To Send (RTS-CTS), Carrier Sense Multiple Access with Collision Detection (CSMA/CD) and Distributed Coordination Function (DCF) [22].

RTS-CTS In this protocol when source node wants to send data initiates the process by sending a RTS frame. The destination node replies with a CTS frame. Any other node receiving the RTS or CTS frame should refrain from sending data for a given time (solving the hidden node problem). The amount of time the node should wait before trying to get access to the medium is included in both the RTS and the CTS frame.

This protocol was designed under the assumption that all nodes have the same transmission range. If the nodes are not synchronised (or if the packet sizes and data rates are different) the problem may occur that the sender will not hear the CTS or the ACKnowledgement (ACK) during the transmission of data of the second sender. This kind of protocols is preferable into V2I communications [21].

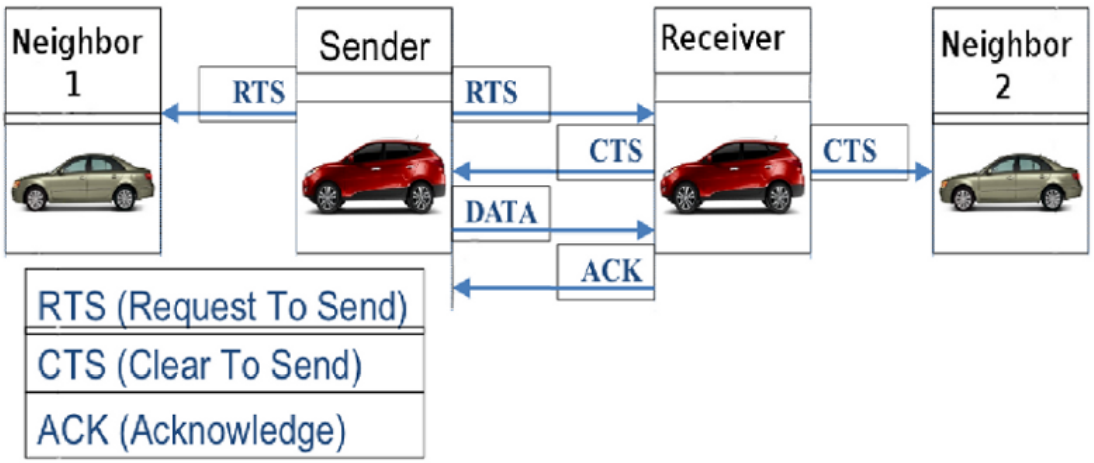

Figure 1.9: RTS - CTS

CSMA/CD In this protocol when a node wants to send data check if there is an ongoing transmission before begging the sending, in case that there is it waits for a random time till it finds free the medium and start transmit data. If a node detects a collision stops the transmission, increase the waiting time and send control packets to inform also the other nodes about it.

IEEE 802.11 uses a CSMA with a per packet MAC acknowledgement, in order to improve robustness of the transmission and also could detect collisions indirectly. Furthermore IEEE 802.11 has an optional handshake scheme between performance source and destination which is useful with hidden node problem. This kind of protocols is preferable into V2V communications.

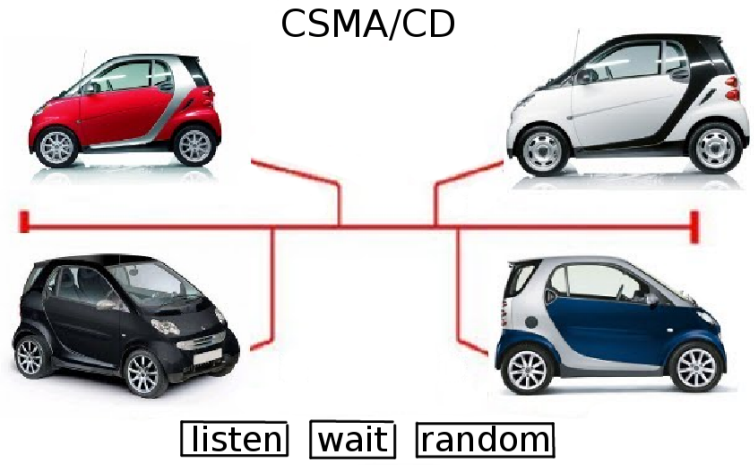

Figure 1.10: CSMA Mechanism

DCF This protocol employs a CSMA/CA with binary exponential back-off algorithm.

The DCF uses carrier sensing along with a four way handshake to maximize the throughput while preventing packet collisions. A packet collision is defined as any case where a node is receiving more than one packet at a time, resulting in neither packet being correctly received.

The basic functionality of 802.11 is the follow description: Assume that a node has data that it needs to transmit, first it will wait a random back-off time. This is a random number of time slots which is within a contention window. If at any time the node senses that another node is using the channel, it will pause its timer until the other node has finished transmitting. When the back-off time has expired, the node will listen the channel to determine if there is another node transmitting. If the channel is clear, it will then wait for a short time and listen the channel again. If the channel is still free, it will transmit RTS to the destination. The destination will respond with a CTS if it is available to receive data (i.e. if it is not receiving data from another node). When the source node receives the CTS, it will transmit its data. Along with both the RTS and CTS, a Network Allocation Vector (NAV) is transmitted, which is a virtual carrier sensing mechanism. After correct reception of the data, the destination will transmit an ACK back to the sender. At this point, if the sender has more data to transmit, it will again begin its back-off and repeat the process. This process is demonstrated in Figure 1.11.

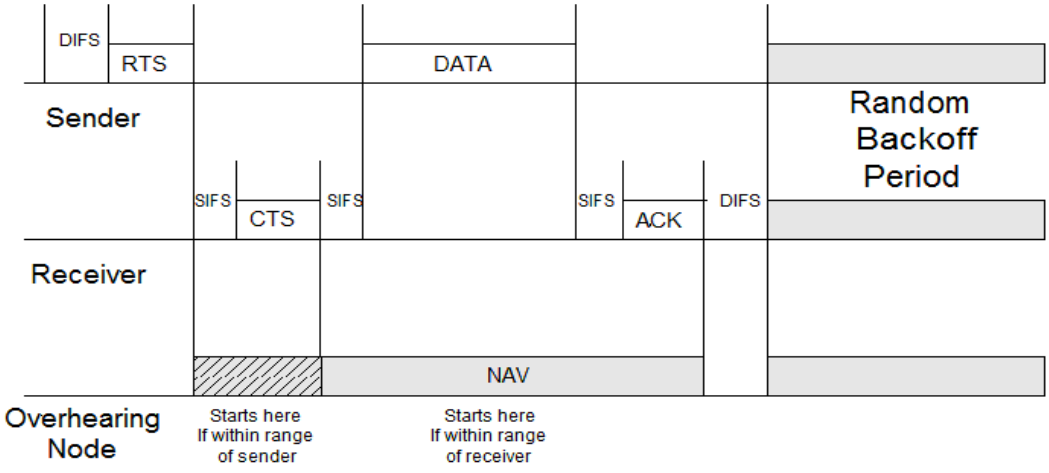

Figure 1.11: DCF Mechanism

1.3.3 LLC

The IEEE Std 802.2 [14] for LLC defines a programming interface between that part of the communications software that controls the network interface

card (MAC and PHY) and the overlying protocol stack (IPv6, WSMP). The connection between the network interface card and the rest of the communications system is through a structure called a Service Access Point(SAP). The SAP differentiates between communications protocols; there's a SAP for NetBIOS, another for SNA, another for NetWare, and so on.

There are two types of LLC. A programmer can select LLC Type II, where the frames have sequence numbers as they pass through the SAP and the 802.2 LLC layer, so the receiver provides an acknowledgement for received frames. This creates a reliable data transfer mechanism at the Data Link Layer (DLL). LLC Type I simply provides the differentiation function, with no sequence and acknowledgement process.

WAVE Networking Service shall support the connectionless unacknowledged Type I operation of the LLC [1] as specified in IEEE Std 802.2, the Subnetwork Access Protocol (SAP) specified in IEEE Std 802, and the standard for transmission of IP datagrams over IEEE 802 networks specified in RFC 1042 [30]. Different Ethernet Type (EtherType) values identify the different network layer protocols used in WAVE. EtherType is also described in IEEE Std 1609.4 [1]. IPv6 type packets received from the lower layers with an Ethernet Type value of 0x86DD are delivered to the IPv6 protocol. WSM packets received from the lower layers with an Ethernet Type of 0x88DC are delivered to the WSM protocol. IPv6 packets for transmission shall have Ethernet Type set to 0x86DD. WSM packets for transmission shall have Ethernet Type set to 0x88DC.

1.3.4 Network and Transport layer

WAVE is providing communications services to application providing support for two protocol stacks, the Wave Short Message Protocol (WSMP) and IPv6 (figure 1.3). While WSMP is developed within the IEEE 1609 family of standards [1], considerations for operation of IPv6 for WAVE are less developed. The WAVE architecture specification [3] makes reference to the specification of IPv6 [10] and makes minimal observations regarding the use of IPv6 addresses, but no further

specific recommendations as to IPv6 operation for WAVE are provided.

1.3.4.1 IPv6

IPv6, as defined in [10] principally concerns the data frame layout (header format, header extensibility, rules governing header construction and processing etc.), IPv6 also implies operation of a set of basic protocols at the network layer. The IPv6 protocol stack provides additional protocols at other layers, such as the transport layer and the application layer. Most of these protocols make certain assumptions about properties of an underlying link model for their proper operation and assume certain relationships between assigned IP addresses and communications ability across the underlying data link layer.

At the transport layer, the IPv6 protocol stack proposes two types of protocols: TCP that is a reliable, rate-adapting mechanism enabling end-to-end transport of application data across several IP hops and requiring bi-directional communication between the peers for acknowledgements etc. The second IPv6 transport protocol is UDP, much simpler protocol providing no rate-adapting or reliability mechanism and so no signalling from the destination to the sender in a traffic flow. It is worth noting that TCP is often very inefficient in wireless ad-hoc environments, especially when faced with mobility: TCP was designed to interpret packet loss as traffic congestion and to diminish sending rates in this case, whereas in wireless networks, packet loss may have causes that are other than traffic congestion, such as interfaces moving out of reach, collisions or interference. Also, if a TCP connection is established between two air-interfaces, subsequently moving out of range before the connection is terminated, connection-state remains for timing out (and possibly causing extraneous transmissions), not cleared up by the usual end-of-connection signalling.

In general TCP is usually not employed in VANETs, which leaves UDP as the only viable alternative within the standard IPv6 stack. Applications requiring rate-adapting or end to end transport reliability services may not be satisfied with what the standard IPv6 protocol stack has to offer.

1.3.4.2 WSMP

WSMP is used to increase message transmission and services on WAVE system. WAVE short messages can be sent both on control channels and service channels. It allows physical layer parameters like transmitter power to be controlled by application. WAVE short messages can be delivered to multiple destinations. Applications take responsibility for message signing [1] and providing the channel information for transmission.

On receipt of Wave Short Message (WSM) request, WSMP shall verify the length of WSM Data is less than the value of the WAVE Management Entity (WME) and Management Information Base (MIB) parameter WsmMaxLength. The WSM request may be received from a local application or from a remote application via the forwarding function's UDP port, as specified in the WME-MIB. Upon verification of WsmMaxLength, WSMP shall pass it to the LLC layer for air interface transmission by means of DL-UNITDATA request. Otherwise, WSM Data is not passed. On receipt of DL-UNITDATA indication from LLC, WSMP shall pass it to the destination application, as determined by the PSID, in the form of a WSM indication. The destination application may reside on the WAVE device, or on a separate device, in which case the application registration will have indicated how to perform delivery via the UDP/IP stack, using the address and port number from the User Service Info.

1.3.5 Applications

All services available into a VANET can be classified into three different groups: safety, traffic control and user applications [40]. Safety messages should be exchanged among vehicles for avoiding road accidents and nodes pile-up. The traffic control has been thought for improving the travel quality by preventing traffic jam and congestions. Finally, user applications have been introduced in order to provide entertainment services (i.e. music and video sharing, chat, games and Internet connectivity) to passengers in vehicle.

Safety and traffic control This kind of application in tied related because they can be combined to achieve road safety driving and prevent a lot of car accidents. There two types of applications that could be useful: car accidents prevision and road congestion.

- Car accidents prevision: The time reaction of a driver in compare the speed of vehicle is very low. So a car accident could be happen in a few seconds if the drivers has not strongly reflexes. Safety applications could provide warnings and informations about the road situation and other facts that could be useful to driver in order to avoid a car crash.

- Road congestion: Safety and traffic control applications could provide to drivers with the best routes to reach their destination, considering the traffic jams and car accidents. This would decrease congestion on the road and maintain a smooth flow of traffic, thus increasing the capacity of the roads and preventing traffic jams. It will decrease also, at one point the possibility to have car accident, because drivers would be less frustrated and will be advised to keep the traffic rules.

User Applications These applications could make the travelling more interesting and entertainment, given to passengers a pleasant trip. There are two basic types: Internet connection and Peer-to-Peer (P2P).

- Internet connection: Web is a part of people day-life nowadays, so more and more people need to have constant connect to internet social networks, e-mail and all this useful services. VANET is challenging with this issue that also will make new opportunity for already or new-one services by the web.

- P2P: This is an another interesting idea that will allow passengers to share news, music, videos and any kind of informations among vehicles. Furthermore it will be used for online chatting and gaming.

1.4 Routing Protocols

Because of the high mobility of nodes into a VANET system to design a routing protocol able to compute and maintain efficiently routing paths among vehicles, represents nowadays a challenging research issue. So far several routing protocols have been developed, some of them have been obtained, adapted and improved from algorithms that proposed in the past for MANETs. This protocols despite the fact that has been demonstrated how they can reach good performance for MANETs, they are not able to guarantee the same level of efficiency into a VANET scenarios yet. Hence, new ideas and more sophisticated strategies have been developed. A lot of these novel approaches compute routes starting from the information about node position where other protocols divide the nodes into cluster (small groups) [16].

Here is being summarized and compared the most well known VANET routing algorithms, which can be classified in five different categories: Topology-based, Broadcast, Cluster-based, Position-based and Infrastructure-based [18]. In order to give a general view of the existing routing protocols is provided a description of the most spread routing algorithms and a comparison among considered strategies.

1.4.1 Topology-based Ad-hoc Routing Protocols

In this category are also some algorithms that have been proposed in the past for MANETs and have adjusted in order to be used in VANETs. It could be found two categories in this kind of protocols: proactive and reactive, proactive means that then a node wants to send a packet tries to find a route using the topology, proactive is a technique in which nodes maintain a routing table exchanging periodically information about topology. Most known topology-based routing protocols are: AODV, OLSR, DSDV, DSR, Temporally Ordered Routing Algorithm (TORA), Fisheye State Routing (FSR), Zone Routing Protocol (ZRP) [31].

1.4.1.1 AODV

AODV is a reactive routing protocol ,so a route is created when a node wants to send a packet. Every node sends periodically hello messages to find its neighbours. When a node has to send a packet to destination which is not neighbour send a Route Request (RREQ) to neighbours to find a path, AODV makes sure these routes do not contain loops and tries to find the shortest route possible. AODV is also able to handle changes in routes and can create new routes if there is an error [28]. For AODV it can be found a lot of extensions, example of an extension of AODV is Multi-hop AODV-2T [34].

1.4.1.2 OLSR

OLSR is a table driven and proactive protocol [6], this means that a routing table is maintained and is being exchanging topology information with other nodes of the network regularly. The nodes which are selected as a Multi-Point Relay (MPR) by some neighbour nodes announce this information periodically in their control messages. Thereby a node announces to the network that it has reachability to the nodes which have selected it as MPR. In route calculation, the MPRs are used to form the route from a given node to any destination in the network. The protocol uses the MPRs to facilitate efficient flooding of control messages in the network.

1.4.1.3 DSDV

DSDV is a table-driven routing scheme for ad-hoc mobile networks based on the Bellman-Ford algorithm and it was developed by C. Perkins and P.Bhagwat [29]. The main contribution of the algorithm was to solve the routing loop problem. Each entry in the routing table contains a sequence number, the sequence numbers are generally even if a link is present; else, an odd number is used. The number is generated by the destination and the emitter needs to send out the next update with this number. Routing information is distributed between nodes by sending full dumps infrequently and smaller incremental updates more frequently.

1.4.1.4 DSR

DSR is similar to AODV with some differences [9]. First in Route Discovery phase every node transmit the whole path unlike AODV that sends only the next-hop. Second the discovered route is being maintained for period of time. Another things is that a source node sends a RREQ packet by flooding the network unlike with AODV that send periodically hello messages to know its neighbours. Also every node is responsible for confirming that the next-hop receives the packet. If a packet can not be received by a node, it is retransmitted up to some maximum number of times until a confirmation received from the next-hop.

1.4.1.5 TORA

Like DSR, TORA has route discovery and route maintenance phase, but in addition has also route erase phase [20]. In DSR when a source sends RREQ, every node constructs a graph with links that the path has been taken place, in this way every node makes a routing table. When a node detects a network partition, it generates a clear packet that resets the routing state and removes invalid routes from the network.

1.4.1.6 FSR

FSR is a proactive routing protocol that means packets are constantly broadcasting and flooded among nodes to maintain the path. In FSR nodes maintain a Topology Table (TT), based on the latest informations that already have by the exchanging messages, for making the paths[17].

1.4.1.7 ZRP

ZRP is a hybrid routing protocol. In this routing protocol in every node has been set a zone around that defines its neighbourhood, all nodes that is in the some transmission range make a zone. For routes inside of the zone the route is discovering reactively, for routes outside of zone the node transmit a route request to the others zones [36].

1.4.2 Broadcast Routing Protocols

As described in [39], broadcast routing protocols are often used for sharing traffic, weather, emergency, road condition among vehicles and delivering advertisements and announcements.

In general, the role of a routing protocol is to find a path for connecting two nodes over a multi-hop route. However, routing algorithms based on the broadcast approach have a different aim. When a broadcast routing protocol is used into a vehicular network, in fact a node that wants to send a packet, does not start any routing algorithm (e.g. to find an entry into a routing table to send a control message for establishing the path) but sends it to all neighbours that will re-broadcast packets to their neighbours. The only main objective of a broadcast routing algorithm is to avoid waste bandwidth by sending packets to disinterested node (e.g. node that have already sent/received same packets). Accordingly broadcast routing protocols differ among them for the strategy adopted for avoiding waste of bandwidth.

Most known broadcast routing protocols are: Secure Ring Broadcasting (SRB), Distributed Vehicular broadCAST (DVCAST), Parameterless Broadcast in Static to highly Mobile (PBSM), Edge-Aware Epidemic Protocol (EAEP) [39].

1.4.2.1 SRB

This routing protocol divides the nodes into three different classes according the received power. First are Inner Nodes (IN) which are close to the sending node. Second are Outer Nodes (ON) which are far away from the sending node. The last are Secure Ring Nodes (SRN) which are at preferred distance form the sending node. Only nodes that belong to secure ring can broadcast packets more than one times.

1.4.2.2 DVCAST

DV-CAST protocol is based on the local information that every nodes posses by send periodically hello messages. Each vehicle continuously checks its connections

with over nodes to broadcast the arrival packets according the connectivity.

1.4.2.3 PBSM

This routing protocol does not exchange information about the position of nodes because of GPS usage. Every node divides the neighbours into two groups ,nodes that Received (R) and nodes that did Not Receive (NR) the packet. So it retransmit packets only to the NR nodes.

1.4.2.4 EAEP

This routing protocol does not divide the nodes into cluster or zones so no extra hello messages are needed. Furthermore nodes does not transmit informations about geographical potions. Every node calculate within this routing protocol a time probability to broadcast messages, so the flooding problem is being avoided.

1.4.3 Cluster-based Routing Protocols

In VANETs the topology is very frequently changed and covers big areas. In this way making routing scalability is a difficult challenge. You can succeed that by dividing the network in different regions named clusters which coordinate and communicate each over in order to achieve connection between nodes. In details, a cluster-based routing protocol creates a virtual infrastructure network which is consist by clusters. Furthermore each cluster is being organized in order to have a cluster-head which is responsible for communication among node into the cluster and a cluster-gateway which is the node that is responsible for the communication with other clusters. Every over node is being clustering as cluster-member. Also the geographical informations of nodes is needed, where a GPS system or a infrastructure local services can be used[16].

Most known clustered-based routing protocols are: Cluster Based Location Protocol (CBLR), Cluster Based Directional Routing Protocol (CBDRP), Euclidean Distance Cluster Based Routing Protocol (EDCBRP) [16][38].

1.4.3.1 CBLR

In the initialization state each node broadcast a hello message and coordinates with other nodes to define the cluster-heads. Every cluster-head maintains table about location and address of cluster-members and cluster-gateways. Additional has a table with neighbouring clusters. The packets with destination in the same cluster are being send to the closest neighbour with destination, otherwise communicates with cluster-head which ask from cluster-gateways to coordinate with other cluster in order to find the location of destination. The location information is updated every time when a packet has to be forwarded [16].

1.4.3.2 CBDRP

In this protocol, the nodes are being clustered according move direction [35], the nodes with the same direction made a cluster and after one of them become cluster-head. When a node wants to send a packet is forwarded to header, which forwards the packet to the header of destination cluster.

1.4.3.3 EDCBRP

In this technique of clustering, the nodes are being divided with calculation of euclidean distance between nodes. A threshold is defined and every node below that is classified to the same cluster. Every node uses a GPS system to get the information about the position of every node and maintains the network topological structure using also the information of hello-beacon message that is being transmitted periodically from all nodes. When a node wants to send a packet checks its topology table and forward the packet to the next-hop. If there is no record about the destination, the node uses a reactive technique broadcasting route-request to nodes of the cluster.

1.4.4 Position-based Routing Protocols

VANET technology has to cope with a big challenges such as the high speed of nodes and the quickly changing of topology. For this reason the Topology-based ad-hoc routing protocols are not very efficiency and robustness, so other

ways of making paths and forward the packets have to be found. In the last years a very promising approach with very good results is Position-based routing protocols. This kind of routing protocols makes routing decisions according to the geographical position of nodes. This information is being taken by GPS technology and local services (V2I) [19]. A specific category of position-based routing protocol is Geocast routing, in which the nodes are being divided into predefine geographical regions. It can be found three strategies to forward packet: greedy forwarding, restricted directional flooding and hierarchical.

- Greedy forwarding: in this strategy protocols do not create a path from source to the destination, they find the next-hop considering some parameters about position of other nodes (i.e the closest neighbour to the destination).
- Restricted directional flooding: in this approach routing protocols forward the packet to several nodes (broadcasting) to increase the possibility to have a correct path to the destination, every node when received a packet has some criteria to whether or not retransmit the packet.
- Hierarchical: this kind of routing protocols create a hierarchy according to the position of the vehicle to escalate large number of nodes.

Position-based routing protocols exploit geographical location information of nodes belonging to the network (i.e. obtained from street maps, traffic models and navigational systems) for creating paths [12]. Since in a vehicular scenario node movements are usually restricted in just few directions because of road constraints, performance reached by these strategies outperform all other routing algorithms developed for VANETs. Hence, position-based routing algorithms represent nowadays the most promising routing paradigm for VANETs.

Most known position-based routing protocols are: Greedy Perimeter Stateless Protocol (GPSR), Adaptive Movement Routing (AMAR), GreedY Traffic Aware Routing (GYTAR), Distance Routing Effect Algorithm for Mobility (DREAM), Location Area Based Ad-hoc Routing (LABAR), RObust VEhicular Routing (ROVER) [32] [40] [19].

1.4.4.1 GPSR

This routing protocol in order to find the neighbours is sending periodically hello messages. GPSR finds the next-hop using the position between neighbour and destination which has been taken by GPS and local services, the most closest node to destination is the next-hop [32]. Because a possibility to failure to find a path, GPSR has been extended with perimeter routing strategy recovery which transfers the connectivity graph into a planar graph by eliminating redundant edges in order to get out of local maximum.

1.4.4.2 AMAR

This is another routing protocol that uses greedy forward to select the next-hop. AMAR for selecting the next-hop uses additional informations which are being taken from GPS or navigation system such as the direction, position and the speed of each vehicle. Furthermore AMAR uses this type $Wi = \alpha P + \beta D + \gamma S$ [32] , where α, β and γ are weighs for Position (P), Direction (D) and Speed (S), to choose the next-hop.

1.4.4.3 GYTAR

This protocol uses greedy forward too. But it has improved to use addition informations. It contains two modules: junction selection and improved greedy forwarding data. The junction selection checks the traffic density of nodes in each junction and compares it in order to find the junction with highest values, when forwards the packet to this junction. In improved greedy forwarding method each node maintains a table with informations about position, velocity and direction of each nodes, this data are obtained from GPS or local services and updated periodically. So GYTAR evaluates this information (data for nodes from GPS and high-density junction) to select the next-hop[32].

1.4.4.4 DREAM

DREAM is a restricted directional flooding protocol that means the packet is being forwarded to several next-hop nodes in order to reach the destination.

DREAM limits the number of broadcasts, forwarding the packets to specific regions. Every node uses local services to take the position of each node in the network. With this informations it measures the region that is possible the destination belongs to and broadcast the packet to this region, called expected region[19].

1.4.4.5 LABAR

LABAR is an example of hierarchical position-based routing protocol. LABAR creates location areas using specific nodes (G-nodes) using V2I communication. So a virtual back-bone network is needed, where G-nodes are able to communicate with those local services to collect data about other nodes (S-nodes). To forward a packet LABAR creates zone formation using virtual backbone formation and directional routing. In the beginning LABAR defines the zones of G-nodes, in order to every G-node corresponds several S-nodes. Then G-nodes communicate with local services to obtain information about S-nodes. At the end G-nodes are able to make direction routing and choose among zones where nodes are appropriate to make the connection between source and destination[19].

1.4.4.6 ROVER

This is an example of Geocast routing protocol. ROVER has some predefined geographical regions called Zone of Relevance (ZOR). ZORs are rectangles on digital map of vehicles. With ROVER nodes can broadcast packets to a specific ZOR where packets are being broadcasting to all nodes which belong to specific ZORE. So every packet has [A, M, Z] where A is application, M is message and Z is Zone. Furthermore every node communicates with local services to define in which zone belongs, so it can distinguish if the packets refers to its zone. When a node wants to forward a packet makes a Zone of Forward (ZOF) and sends the packet.

1.4.5 Infrastructure-based Routing Protocols

This kind of protocols relay on fixed infrastructure bases to help about routing issues. Most well-known infrastructure-based routing protocol is Roadside-Aided

Routing (RAR).

1.4.5.1 RAR

RAR is a group of techniques that provided by road-side bases in order to help routing. Here geographical areas are divided into sectors by using Road Side Units (RSUs), thus the transmitted data pass by vehicles and RSUs, in order to reach the destination from the source. This protocol is preferable into urban roads because required road side infrastructure bases[27].

1.4.6 Final Comparison

This section presents a comparison table among all mentioned routing protocols. In the first column we put the name of every protocol. The category refers to one of the five different categories (Topology-based, Broadcast, Cluster-based, Position-based and Infrastructure-based). Position info is about the needing of position information of vehicles to create the routing path. Routing table refers to the option of routing protocol to keeps already found routing paths. Type takes three values: reactive, proactive, hybrid (a mix of reactive and proactive) and other, those values indicate the way in which routing protocol create the path between source and destination. Input parameters are values that some routing protocols need to create routing paths which is Hello message (nodes send periodic messages to discover their neighbours) and position information (the physical position of other nodes). Finally, table 1.2 has the simulation tool in which routing protocols have been implemented.

Table 1.2: Comparison among Routing Algoritms

Protocol Name	**Category**	**Position info**	**Routing table**	**Type**	**Input parameter**	**Simulation Tool**	

AODV	Topology-Based	No	No	Reactive	Hello message	NS2, NS3, OPNET	
OLSR	Topology-Based	No	Yes	Proactive	Topology information	NS2, NS3, OPNET	
DSDV	Topology-Based	No	No	Reactive	Hello message	NS2, NS3	
DSR	Topology-Based	No	No	Reactive	Hello message	NS2, NS3	
TORA	Topology-Based	No	Yes	Reactive	Hello message	NS2, OPNET	
FSR	Topology-Based	No	Yes	Proactive	Topology information	NS2	
ZRP	Topology-Based	No	Yes	Hybrid	Hello message,Topology information	NS2	
SRB	Broadcast	No	No	Reactive	Not needed	WibDat, Jist/SWANS	
DV-CAST	Broadcast	No	No	Proactive	Topology information	WibDat, Jist/SWANS	
PBSM	Broadcast	No	No	Reactive	Position informations	WibDat, Jist/SWANS	
EAEP	Broadcast	No	No	Reactive	Not needed	WibDat, Jist/SWANS	
CBDRP	Cluster-Based	Yes	Yes	Proactive	Position informations	NS2	

EDCBRP	Cluster-Based	Yes	Yes	Proactive	Position informations, Hello message	NS2	
CBLR	Cluster-Based	Yes	Yes	Hybrid	Position informations	NS2	
GPSR	Positions-Based	Yes	No	Other	Hello message, Position informations	NS2	
AMAR	Positions-Based	Yes	No	Other	Position informations	NS2	
GyTAR	Positions-Based	Yes	No	Other	Position informations	NS2	
DREAM	Positions-Based	Yes	No	Other	Position informations	NS2	

Chapter 2

NS-3 Simulator

2.1 About NS-3

NS-3 is a discrete-event network simulator, targeted primarily for research and educational use. NS-3 is free software, licensed under the GNU GPLv2 license and is publicly available for research, development and use. The goal of the NS-3 project is to develop a preferred, open simulation environment for networking research: it should be aligned with the simulation needs of modern networking research and should encourage community contribution, peer review and validation of the software [25]. NS-3 is available for Linux, Mac OS and MS Windows using cygwin [7].

The NS-3 project is committed to building a solid simulation core that is well documented, easy to use and debug that caters to the needs of the entire simulation work-flow, from simulation configuration to trace collection and analysis. Furthermore, the NS-3 software infrastructure encourages the development of simulation models which are sufficiently realistic to allow NS-3 to be used as a real-time network emulator, interconnected with the real world and which allows many existing real-world protocol implementations to be reused within NS-3.

The NS-3 simulation core supports research on both IP and non-IP based networks. However, the large majority of its users focuses on wireless/IP simulations which involve models for Wi-Fi, WiMAX or LTE for layers 1 and 2 and and a variety of static or dynamic routing protocols such as OLSR and AODV for IP-based applications. NS-3 also supports a real-time scheduler that facilitates a number of "simulation-in-the-loop" use cases for interacting with real systems. For instance, users can emit and receive NS-3-generated packets on real network

devices, and NS-3 can serve as an interconnection framework to add link effects between virtual machines. Another emphasis of the simulator is on the reuse of real application and kernel code. Frameworks for running unmodified applications or the entire Linux kernel networking stack within NS-3 are presently being tested and evaluated.

Because creating a network simulator that supports a sufficient number of high-quality validated, and maintained models requires a lot of work, NS-3 attempts to spread this workload over a large community of users and developers. Every three months, it ships a new stable version of NS-3 with new models developed, documented, validated, and maintained by enthusiastic researchers. It encourages the open validation of these models by third parties on its mailing-lists to ensure that the models it ships and stay of the highest quality possible.

2.2 Release Process

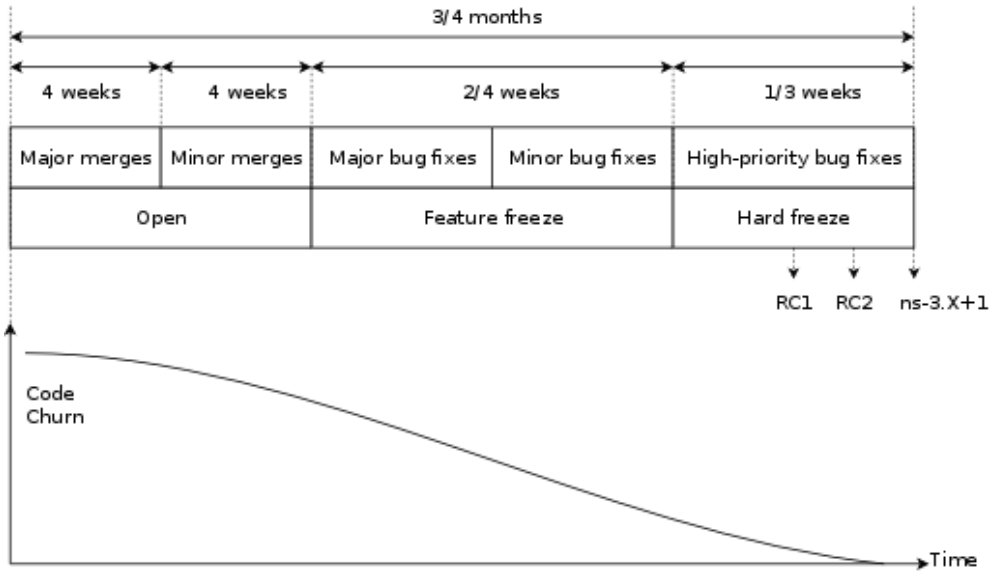

Figure 2.1: NS-3 Release Process

NS-3 releases are based on date-driven schedules: rather than target a set of features for a specific version number, it aims instead for a release date and ship whatever is ready by that date. If a new feature misses that date, it is not a big hassle because the next release is never too far away. This interval is typically 3-4 months.

The objective of this deterministic and predictable release rhythm is to allow external contributors to synchronize their own development schedule on the NS-3 release schedule and to be able to plan their contributions accordingly. This relieves the NS-3 maintainers from the burden of having to make strong unpopular decisions about the merging of new features and allows them to focus instead on maintaining the NS-3 models to achieve the highest-quality possible releases.

A NS-3 Release Manager (RM) manages each release. Following the ns-3.X release, a new RM is selected for the ns-3.X+1 release. The old ns-3.X RM is typically responsible for any maintenance ns-3.X.Y releases. The RM is allowed to veto and remove any new feature addition if it begins to cause problems and looks like it threatens the stability of the release at any time in the release process. Each release is roughly structured as follows: large major intrusive changes that are the most destabilizing to the code-base are always merged first early on during the cycle. As the release deadline be approached, the number and the size of the changes should decrease until only high priority bugs are fixed during the hard freeze period.

During the open phase, people wanting to merge a new feature should contact with RM and arrange to have their features merged into the main development tree. RM expects from everyone to follow the code submission guidelines. One of the NS-3 maintainers will take a quick look at the proposed addition and determine if a code review is required. According to the book of instructions a code review requiring positive acknowledgement by maintainers and is indicated if:

- The proposed feature does not work with all models or on all platforms

- The feature changes pre-existing APIs
- The feature crosses maintainer boundaries

NS-3 commission will probably run a feature submission by at least one maintainer according to the general area of applicability of the feature. For example, if an entirely new device driver model be submitted, this submission will be run by the maintainers of the current devices. The maintainers will not have any responsibility to positively acknowledgement the submission, but it will take some time to allow a reasonable review.

During the feature-freeze (also called maintenance) phase, no new features may be added, but the maintainers may check in fixes to bugs; and people with new features that have been accepted and previously merged may fix bugs in existing features.It will be a properly behaviour to not penetrate in more new features or it may have the whole feature set removed at the release manager's discretion. It can be asked if want to add small, self-contained features, but there are no guarantees that it will be applied.

During the hard code freeze, the primary goal is stability: only P1 bug-fixes will be allowed to be checked in. The goal is to reduce the number of P1 bugs to zero before the release.

2.3 Architecture of NS-3

NS-3 is a C++ library which provides a set of network simulation models implemented as C++ objects and wrapped through python. Users normally interact with this library by writing a C++ or a python application which instantiates a set of simulation models to set up the simulation scenario of interest, enters the simulation main-loop and exits when the simulation is completed.

The NS-3 library is wrapped to python using the pybindgen library which delegates the parsing of the NS-3 C++ headers to gccxml and pygccxml to gen-

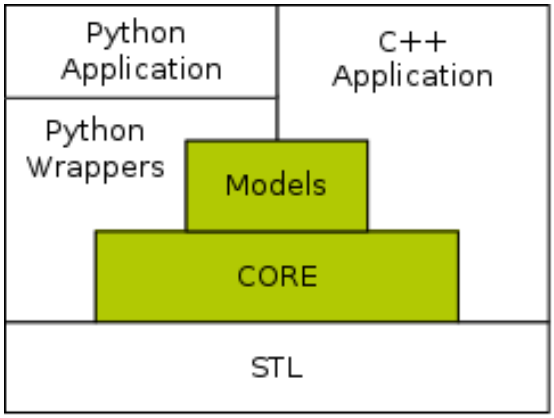

Figure 2.2: NS-3 Architecture

erate automatically the corresponding C++ binding glue. These automatically-generated C++ files are finally compiled into the NS-3 python module to allow users to interact with the C++ NS-3 models and core through python scripts.

In comparison with other discrete-event network simulators, NS-3 is distinguished by the high level design goals. Many simulators use a domain-specific modelling language to describe models and program flow. NS-3 uses C++ or Python, allowing users to take advantage of the full support of each language.

Simulation events in NS-3 are simply function calls that are scheduled to execute at a prescribed simulation time. Any function can be made into an event and scheduled, by use of a callback function. This is in contrast to specialized handler functions that centralize the processing of events in each simulation object. Callbacks are also heavily used in the simulator to reduce compile-time dependencies between simulation objects.

NS-3 features a powerful low-level API that allows power users the flexibility to configure things in different ways. Layered on top of this is a set of helper layer APIs that provide easier-to-use functions with reasonable default behaviour. NS-3 users can mix and match between the simpler APIs at the helper layer and the full APIs underneath.

The simulation design is oriented towards use cases that allow the simulator to interact with the real world. NS-3 packet objects are stored internally as packet byte buffers (similar to packets in real operating systems) ready to be serialized and sent on a real network interface. Several different simulation-in-the-loop and virtual machine integration frameworks have been developed, and NS-3 experiments have been carried out on wireless test-beds.

NS-3 nodes are patterned after the Linux networking architecture, and key interfaces and objects (sockets, net devices) are aligned with those in a Linux computer. This better facilitates code reuse and improves realism of the models, and makes the simulator control flow easier to compare with real systems.

The NS-3 simulator features an integrated attribute-based system to manage default and per-instance values for simulation parameters. All of the configurable default values for parameters are managed by this system, integrated with command-line argument processing, Doxygen documentation, and an XML-based and optional GTK-based configuration subsystem.

The project does not maintain an Integrated Development Environment (IDE) to configure, debug, execute, and visualize simulations in a single application window, such as found in other simulators. Instead, the typical work-flow is to work at the command line and integrate configuration and visualization tools as needed.

2.4 The difference between NS-2 and NS-3

NS-2 is a popular discrete-event network simulator developed under several previous research grants and activities. It remains in active use and will continue to be maintained. NS-2 was built in C++ and provides a simulation interface through OTcl, an object-oriented dialect of Tcl [26]. The user describes a network topology by writing OTcl scripts, and then the main NS-2 program simulates that topology with specified parameters. It runs on Linux, Mac OS and MS Windows

using Cygwin [7]. It is licensed for use under version 2 of the GNU General Public.

NS-3 is a complete new software development effort focused on improving upon the core architecture, software integration, models, and educational components of NS-2. The project commenced in July 2006 and the first release was made on June 30, 2008. NS-2 is often criticized because modelling is a very complex and time-consuming task, since it has no GUI and one needs to learn scripting language, queuing theory and modelling techniques. NS-3 is often criticized for its lack of support for some protocols which were supported in NS-2.

As previously mentioned to write a script into NS-2 you have to use a specific language called Tcl. NS-2 scripts will not run within NS-3 ,as well NS-2 uses OTcl as its scripting environment and NS-3 uses C++ programs or python scripts to define simulations.

Some NS-2 models that are mostly written in C++ have already been ported to NS-3: OLSR and Error Model were originally written for NS-2. OTcl-based models have not been and will not be ported since this would be equivalent to a complete rewrite.

It is expected that NS-3 it will eventually replace NS-2 in most universities that are currently using NS-2.

2.5 Installation

NS-3 is primarily developed on GNU/Linux platforms and the minimal requirements to run basic simulations are a gcc installation of gcc-3.4/g++-3.4 or greater and python 2.4 or greater. NS-3 is supported on the following primary platforms:

1. Linux x86 gcc versions 4.1 through 4.6 and, 3.4.6.
2. Linux x86-64 gcc versions 4.6 through 4.1 and 3.4.6
3. MacOS X ppc and x86 (gcc 4.0.x and 4.2.x)

4. Cygwin gcc 3.4.4 (debug only), gcc 4.3.2 (debug and optimized).

By supported, means that the project tries to support most or all of the build options on these platforms unless there is a good reason to exclude the option; and at least the debug build will compile. If you intend to do serious work using NS-3, and are forced by circumstances to use a Windows platform, consider virtualization of a popular Linux platform. This may be more time-consuming than installing a minimal Cygwin, for example, but you end up with a fully functional Linux system and NS-3 distribution. Some quick performance comparisons between Cygwin and VirtualBoxed Fedora 11 indicate that examples can actually run faster under VirtualBox.

NS-3 may also run on currently unsupported platforms. For example, an alternative Windows platform is MinGW. There are maintainers who attempt to keep a subset of NS-3 running on MinGW, but it is not "officially" supported. This means that bugs filed against MinGW will be addressed as time permits. Additionally, the Eclipse IDE is used by some developers, but the project does not actively support this environment. Additional maintainers are invited to make more platforms, compilers and environments supported.

There are a few options that are not enabled by default and are not available on all platforms. At the end of the configuration process (explained below), the status of these options are shown as detected by waf script:

```
---- Summary of optional NS-3 features:
Threading Primitives          : enabled
Real Time Simulator           : enabled
Emulated Net Device           : enabled
Tap Bridge                    : enabled
GtkConfigStore                : enabled
XmlIo                         : enabled
SQlite stats data output      : enabled
Network Simulation Cradle     : enabled
Python Bindings               : enabled
```

```
Python API Scanning Support   : not enabled
 (Missing 'pygccxml' Python module)
MPI Support                   : not enabled
 (option --enable-mpi not selected)
Use sudo to set suid bit      : not enabled
 (option --enable-sudo not selected)
Build examples and samples    : enabled
Static build                  : not enabled
 (option --enable-static not selected)
GNU Scientific Library (GSL)  : enabled
```

Generally if the platform is missing some requirement for an option it is marked as "not enabled." Note that "disabled by user request" will be shown when the user explicitly disables a feature (such as "–disable-python"); and if a feature defaults to disabled this will also be noted (e.g., option –enable-sudo not selected).

The core of NS-3 requires a gcc/g++ installation of 3.4 or greater, and python 2.4 or greater. As mentioned above, different options require additional support. Thee is a list of packages (for Debian/Ubuntu systems) that are needed to support different NS-3 options. Note that other distributions (e.g., Fedora, FreeBSD) may have different package names or capitalization (e.g. ImageMagik). Installation should be similar for Red Hat/Fedora based systems, with "yum" replacing "apt-get", but some differences exist, see the appendix for more informations.

2.6 Set-up NS-3

NS-3 makes unpackaged source releases only at this time. Downloading the latest stable release should be straightforward, from the main project page. Archived older releases are also linked there [25]. Once users have downloaded NS-3 (a development or stable version), they will want to then invoke build.py to start a coordinated build. In the ns-allinone-xxx folder execute the script:

```
./build.py
```

If all goes well, user can change directory (cd) into ns-3-dev and run the NS-3 tests:

```
cd ns-3-dev
./waf check
```

Cygwin works reasonably-well by default: just make sure you grab the cygwin installer from:

```
http://www.cygwin.com/setup.exe
```

cygwin includes support for mercurial, gcc, and, python so, nothing else should be needed.

2.7 Writing Scripts

While experienced users of NS-3 often write new simulation scripts from scratch, most users merely copy/paste an existing script which incorporates the simulation models they need and modify it until it matches what they want. Consult the extensive example scripts that ship with every module.

Once you have identified which simulation script you would like to start from, building a modified version of this script is a matter of dropping it in the scratch directory and running waf again:

```
cp examples/csma/csma-broadcast.cc scratch/csma-modified.cc
./waf
```

Running a new script which was dropped in the scratch directory is similar to running other examples.

If your simulation script becomes complex and if you split it in more than one C++ source file, you need to create a new sub-directory in the scratch directory:

```
mkdir scratch/modified
cp x.cc scratch/modified
cp y.cc scratch/modified
./waf
```

This will build a new program named after the name of your sub-directory (modified here) and you can run it just like any other example:

```
./waf --run modified
```

2.8 Running Scripts

Once the build is done and all tests pass and someone generally wants to run a couple of examples as per the tutorial, the easiest way to do this is to run:

```
./waf --run progname
```

or (a program inside to scratch)

```
./waf --run scratch/progname
```

To find out the list of available programs, you can do:

```
./waf --run non-existent-program-name.
```

The method to run a python script is fairly similar to that of a C++ script except that you need to specify a complete path to the script file. For example:

```
./waf --run examples/wireless/mixed-wireless.py
```

Another way to run NS-3 programs that does not require using the ./waf run command is to use the NS-3 shell which takes care of setting up all the environment variables necessary to do so:

```
./waf shell
```

And, then:

```
./build/debug/examples/csma-broadcast
```

If you do not use the NS-3 shell, and if you want to run your simulation under a special tool (valgrind or gdb), you need to use a command template:

```
./waf --run progname --command-template="gdb %s"
```

or

```
./waf --run progname --command-template="valgrind %s"
```

2.9 Documentation

The NS-3 documentation is available in several forms. Documentation is maintained for the current release, older releases and the current development tree.

- New users will want to consult the NS-3 tutorial, available in html or pdf format. It is being maintained a reference manual on the NS-3 core, and a separate model library documentation set.
- All of NS-3 APIs are documented using Doxygen. Doxygen is typically used for API documentation and organizes such documentation across different modules. This project uses Doxygen for building the definitive maintained API documentation. Additional NS-3 project documentation can be found at the project web site http://www.nsnam.org/ns-3-13/documentation/.
- It has a number of other archived documents such as older tutorials or talks presented about NS-3.
- The NS-3 coding style documentation is maintained on NS-3 site.

2.10 Tools

To download, manage the source code history, track the bugs, build and review patches, could be used many tools that were not developed for NS-3. If someone

are already familiar with them, the following list should give a good idea of what can be found.

- NS-3 bugzilla. [http://www.nsnam.org/developers/tools/bugzilla/]
- NS-3 mercurial. [http://www.nsnam.org/developers/tools/mercurial/]
- NS-3 build: waf. [http://www.nsnam.org/developers/tools/waf/]
- Code reviews: rietveld. [http://www.nsnam.org/developers/tools/rietveld/]
- NS-3 mailing-lists. [http://www.nsnam.org/developers/tools/mailing-lists/]
- Buildbot. [http://www.nsnam.org/developers/tools/buildbot/]
- IRC. [http://www.nsnam.org/developers/tools/irc/]
- Wiki [http://www.nsnam.org/wiki/]

2.11 Mailing-list and Community

NS-3 has a mailing-list for community who want to focused on running and analysing the output of a simulation based on built-in models, as well as to modify or extend an existing model. Furthermore if a person needs to implement a new model or if is simply lost and does not know where to start, could ask a question on NS-3 user mailing-list.

This mailing list is hosted on google groups: anyone can join and browse the archives in the follow site http://groups.google.com/group/ns-3-users.

Chapter 3

Simulation Tools

3.1 Simulation of Urban Mobility (SUMO)

3.1.1 About SUMO

SUMO is an open source, highly portable, microscopic road traffic simulation package designed to handle large road networks. It is mainly developed by employees of the Institute of Transportation Systems at the German Aerospace Center. SUMO is licensed under the GPL [37]. The most significant features are being listed above.

- SUMO includes all applications needed to prepare and perform a traffic simulation (network and routes import, DUA, simulation).
- Simulations has:
  - space-continuous and time-discrete vehicle movement,
  - different vehicle types,
  - multi-lane streets with lane changing,
  - different right-of-way rules, traffic lights,
  - a fast openGL graphical user interface,
  - manages networks with several 10.000 edges (streets),
  - fast execution speed (up to 100.000 vehicle updates/s on a 1GHz machine),
  - interoperability with other application at run-time,
  - network-wide, edge-based, vehicle-based, and detector-based outputs,
  - supports person-based inter-modal trips.

- Network import is possible from other tools:
  - imports VISUM, Vissim, Shapefiles, OSM, RoboCup, MATsim, openDRIVE, and XML-Descriptions,
  - missing values are determined via heuristics.
- Routing facilities:
  - microscopic routes - each vehicle has an own one,
  - different Dynamic User Assignment algorithms.
- High portability:
  - only standard c++ and portable libraries are used,
  - packages for MS Windows and main Linux distributions exist.
- High interoperability through usage of XML-data only.
- Open source(GPL).
- Its available for below operations systems:
  - MS Windows
  - Linux
  - Mac OS

3.1.2 Simulation in SUMO

3.1.2.1 Highway Topology for VANET Simulations

To run a VANET simulation scenario is a high-compatible challenge because of the very high speeds that the nodes reach and the rapidly changing of topology, so it is very important to have a realistic approach of nodes and theirs topology, in order to have coherence facts into simulation scenarios. In this way SUMO is being used in order to export a realistic highway topology of moving cars, which is used for a cohesion examination of VANETs routing protocols.

It has been planned a topology with three types of Vehicles

- Car with length equal to 5 meters (m) and max speed equal to 100 km/h.
- Bus with length equal to 11 m and max speed equal to 80 km/h.
- Truck with length equal to 18 m and max speed equal to 60 km/h.

The vehicles is moving around a rectangular which has width 2000 m and height 600 m. The traffic topology is shown to the figure 3.1.

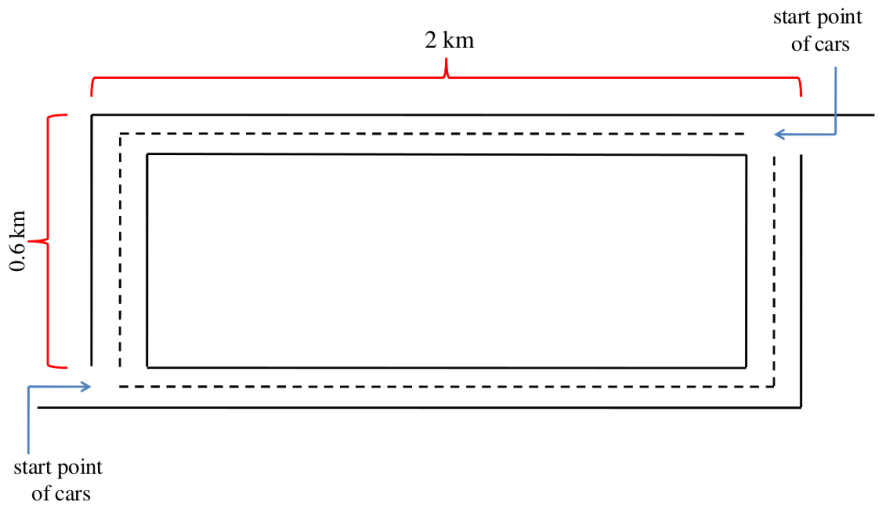

Figure 3.1: High-way Topology

3.1.2.2 Create Scenario in SUMO

In SUMO to create a scenario you need to create specific XML files and put into XML code that sumo is able to recognize and built. XML code doesn't need to pass from XML parser but needs to be well-formed. In general, we have to give information into three files, which will be read in SUMO as input data. Two of the files contain the network information, which will be converted into node and link information in SUMO and is usually named with extension nod.xml and edg.xml respectively. The file containing the traffic demand and route information will be named with extension rou.xml. In addition, two more files with extensions

con.xml and typ.xml will be included. Specification of allowed traffic movements and lane connections at intersections as well as link types are required. A general overview of the required input files is indicated in Figure 3.2.

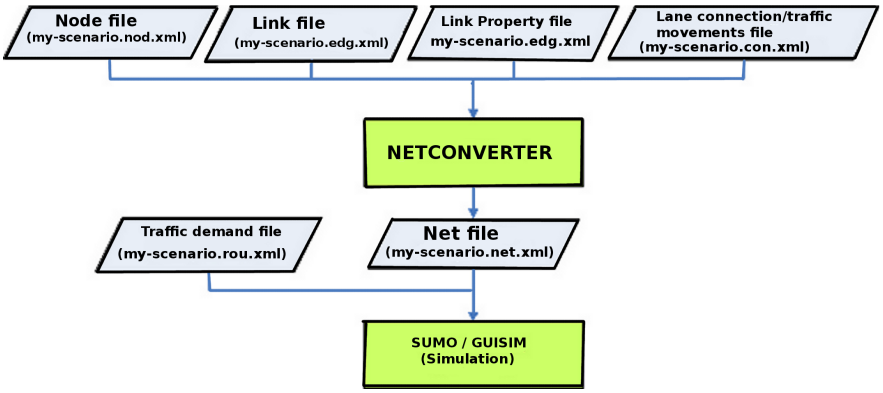

Figure 3.2: SUMO XML-Files

In SUMO a street network consists of nodes (junctions) and edges (streets connecting the junctions). All nodes have at location (x and y coordinate, describing distance to the origin in meters) and an id for future reference. The definitions of the attributes in the node file are listed below:

(a) `id is the ID name of the node, defined by users with numbers, word strings or both.`
(b) `x is the x-coordinate location of the defined node (in meters),`
(c) `y is the y-coordinate location of the defined node (in meters),`
(d) `type is the signal control type of the defined node. It is an optional attribute and defined with priority and traffic-light for unsignalized and signalized intersections respectively.`

The node file of the network is shown in Listing 3.1.

Listing 3.1: Nodes (my-scenario.nod.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<nodes xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation=
```

```
 "http://sumo.sf.net/xsd/nodes_file.xsd">
<nodes>
    <node id="1" x="-2000.0" y="0.0" />
    <node id="4" x="+2000.0" y="0.0" />
    <node id="5" x="+2000.0" y="600.0" />
    <node id="8" x="-2000.0" y="600.0" />
    <node id="9" x="-2100.0" y="0.0" />
    <node id="10" x="+2100.0" y="600.0" />
</nodes>
```

Roads are represented as links in SUMO like in other traffic simulation software. To define link characteristics the identification (id) of each link has to be first defined either with numbers, word strings or both. The information about the upstream node, the downstream node and the link type is then stored in a file with extension edg.xml. The information about each link type can either be defined in an additional file with extension typ.xml or edg.xml right after the downstream node. Listing 3.2 shows the link type file used for the network and four attributes are defined:

(a) id: defined by users with numbers, word strings or both,
(b) priority: driving priority based on traffic regulations and is defined with numbers. The higher the number, the higher the priority for the respective road. The priority information will override information from the node file, if both of them exist,
(c) numLanes: number of lanes on the respective road,
(d) speed: maximum allowed link speed.

Type a are the 4 roads in the rectangular topology with 2 lanes respectively. Type c is non used, so no priority is been set up, finally the maximum speed is 100 km/h.

Listing 3.2: Link Type File (my-scenario.typ.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<types>
  <type id="a" numlanes="2" speed="100.000" />
</types>
```

With the use of the link type file the link file is generated and shown in Listing 3.3. The defined attributes include:

(a) id: link ID, defined by users with numbers, word strings or both,
(b) from: ID of the upstream node of the respective link,
(c) to: ID of the downstream node of the respective link,
(d) type: ID of the link type, defined in the link type file,
(e) allow/disallow: ID of the vehicle group which is defined in the SUMO and might not be identical with the vehicle types defined by users.

Listing 3.3: Link File of the Network (my-scenario.edg.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<edges xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation=
 "http://sumo.sf.net/xsd/edges_file.xsd">
<edges>
    <edge fromnode="1" id="lr" tonode="4" type="a" />
    <edge fromnode="4" id="up" tonode="5" type="a" />
    <edge fromnode="5" id="rl" tonode="8" type="a" />
    <edge fromnode="8" id="down" tonode="1" type="a" />
    <edge fromnode="9" id="start1" tonode="1" type="a" />
    <edge fromnode="10" id="start2" tonode="5" type="a" />
</edges>
```

The default in SUMO based on the given geometric design all possible and logical traffic movements, including u-turns, are allowed, if there is no corresponding

specification. The default setting is that the rightmost lane of each link is aligned to the rightmost lane of the respective downstream link. To specify traffic movements and lane connections an additional file with extension con.xml is required. Listing 3.4 shows the corresponding settings for network. The meaning of each attribute is as following:

```
(a) from: ID of the link which the traffic movements will
be specified,
(b) to: ID of the link which is the downstream link of the
above defined link,
(c) fromLane/toLane: lane number of the defined link in (a)
and the lane number of the link in (b), which are connected.
```

Listing 3.4: Specification of Traffic Movements (my-scenario.con.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<connections xmlns:xsi=
 "http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation=
 "http://sumo.sf.net/xsd/connections_file.xsd">
<connections>
  <connection from="lr" to="up" lane="0:0"/>
  <connection from="lr" to="up" lane="1:1"/>
  <connection from="up" to="rl" lane="0:0"/>
  <connection from="up" to="rl" lane="1:1"/>
  <connection from="rl" to="down" lane="0:0"/>
  <connection from="rl" to="down" lane="1:1"/>
  <connection from="down" to="lr" lane="0:0"/>
  <connection from="down" to="lr" lane="1:1"/>
  <connection from="start1" to="lr" lane="0:0"/>
  <connection from="start1" to="lr" lane="1:1"/>
  <connection from="start2" to="rl" lane="0:0"/>
  <connection from="start2" to="rl" lane="1:1"/>
</connections>
```

Network file in SUMO is named with extension net.xml. With the above defined files, my-scenario.nod.xml, my-scenario.edg.xml, my-scenario.con.xml and my-scenario.typ.xml, the network file my-scenario.net.xml will be generated by apply the module NETCONVERT. For efficient execution, a configuration file that includes the names of the input files, the name of the output network file and other required actions should be created. If the files and the module NETCONVERT are not being located in the same directory, the respective path for each file should be specified. The configuration file for the example network is shown in Listing 2.5 and named my-scenario.netc.cfg. If u-turn movements are not allowed, the command <no-turnarounds value="true"/> should be added to the configuration file. As stated previously, the prohibition of u-turn movements can only be conducted globally.

Listing 3.5: Generating the Network File (my-scenario.netc.cfg)

```
<configuration xmlns:xsi=
 "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
  "http://sumo.sf.net/xsd/netconvertConfiguration.xsd">
    <input>
      <xml-node-files value="my-scenario.nod.xml"/>
      <xml-edge-files value="my-scenario.edg.xml"/>
      <xml-connection-files value="my-scenario.con.xml"/>
      <xml-type-files value="my-scenario.typ.xml"/>
    </input>
    <output>
        <output-file value="my-scenario.net.xml"/>
    </output>
</configuration>
```

The network file my-scenario.net.xml will be generated by executing the following command in the command line.

```
netconvert -c my-scenario.netc.cfg
```

The generated network file can be viewed with the use of SUMO-GUI (see above) for checking , if the network is built accurately , as shown in figure 3.3.

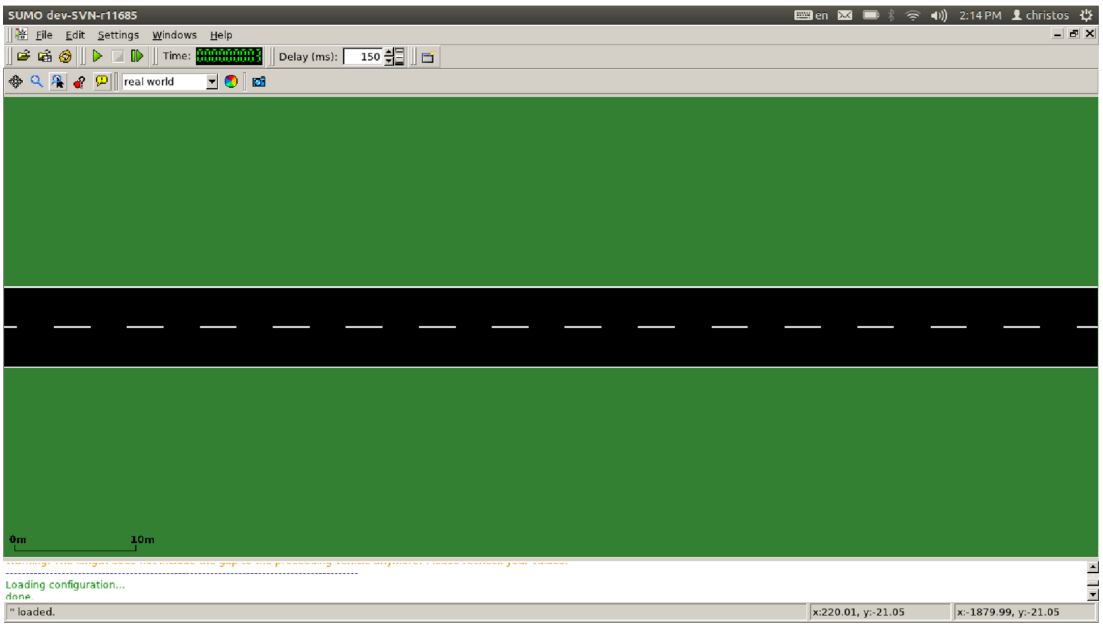

Figure 3.3: SUMO-GUI

Traffic demand and route data is defined together with vehicle type data in a file with the extension name rou.xml. Traffic demand data are defined with eight attributes:

(a) depart: departure time of a certain vehicle,
(b) id: ID of a certain vehicle and defined by users with numbers, word strings or both,
(c) route: the route used by the defined vehicle,
(d) type: ID of the defined vehicle type.

Listing 3.7 shows that there 3 vehicle types (Car, Bus and Truck) with attributes that have been like the plan as previously mentioned on Highway topology. The related attributes include:

(a) id: ID of the vehicle type, defined by users

```
    with numbers, word strings or both,
(b) accel: maximum acceleration of the respective
    vehicle type (in m/s2),
(c) decal: maximum deceleration of the respective
    vehicle type (in m/s2),
(d) sigma: driver's imperfection in driving
    (between 0 and 1),
(e) length: vehicle length (in meters),
(f) maxSpeed: maximum vehicular velocity (in m/s),
(g) color: color of the vehicle type. It is defined
    with 3 numbers (between 0 and 1) for red, green and
    blue respectively. Values are separated by , and is
    in quotes with no space between the values. For example,
    1,0,0 represents the red color, 0,1,0 represents green
    color and 0,0,1 represents blue color.
```

The sequence of the attributes can be changed. The attribute sigma is assigned as 0.0 for all vehicle types.

Following the vehicle type information traffic route data need to be defined as well. The input attributes include:

```
(a) id: ID of a certain route and defined by users
    with numbers, word strings or both,
(b) edges: The sequence of the names of the links,
    composing the defined route.
```

Listing 3.6: Traffic Demand and Route Data (my-scenario.rou.xml)

```
<routes>
<vtype accel="1.0" decel="3.0" id="Car"
 length="5.0" maxspeed="100.0" sigma="0.0" />
<vtype accel="1.0" decel="3.0" id="Bus"
 length="11.0" maxspeed="80.0" sigma="0.0" />
<vtype accel="1.0" decel="3.0" id="Truck"
```

```
 length="18.0" maxspeed="60.0" sigma="0.0" />

<route id="route1" edges="start1 lr up rl
 down lr up ... down " />
<route id="route2" edges="start2 rl down lr
 up rl down ... up " />

 <vehicle depart="1" id="veh1" route="route1"
 type="Truck" />
 <vehicle depart="5" id="veh3" route="route1"
  type="Car" />
 <vehicle depart="9" id="veh5" route="route1"
  type="Bus" />
 <vehicle depart="11" id="veh6" route="route2"
  type="Truck" />
 <vehicle depart="13" id="veh7" route="route1"
  type="Truck" />
 <vehicle depart="15" id="veh8" route="route2"
  type="Bus" />
 <vehicle depart="17" id="veh9" route="route1"
  type="Bus" />
                          .
                          .
                          .
</routes>
```

The three dots in the XML document means that there many such entries. For the purposes of testing VANETs routing protocols it has been produced four different topologies with 50, 100, 150 and 200 vehicles. Because of the multitude lines an automatically way of create the XML documents is needed, in Appendix A is showing an example of shell script for UNIX terminal for this purpose.

Traffic simulation in SUMO can be conducted in two ways as described below. The overview of the simulation process is given in Figure 3.4.

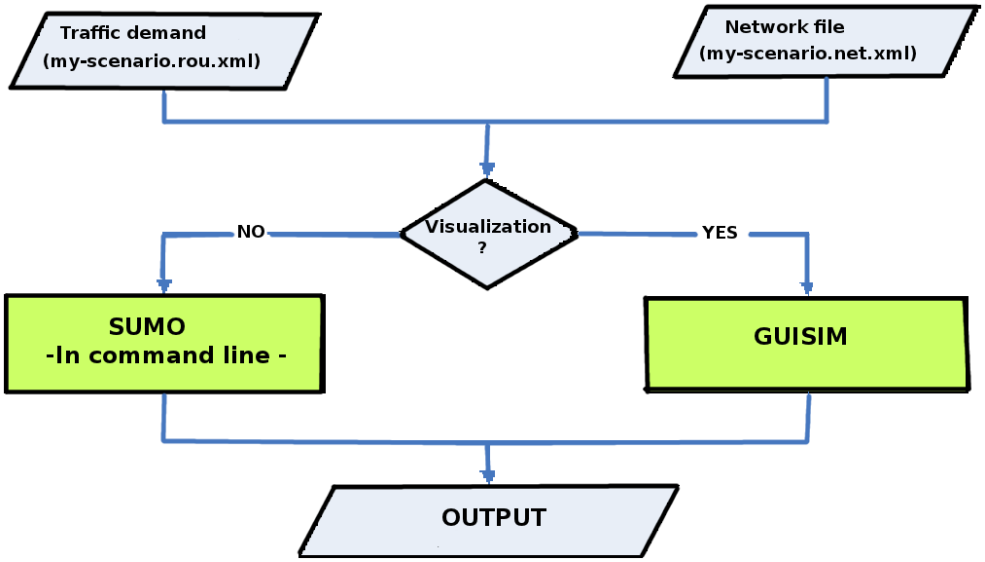

Figure 3.4: SUMO Simulation Process

An efficient traffic simulation execution can be achieved with the use of command line, especially when dealing with large and sophisticated traffic networks. To simplify the execution process it is recommended that all the required execution actions, e.g. the path and the name of the input files, the output types, the output directory and the simulation time period is specified in a configuration file. For my-scenario the respective configuration file is shown in listing 3.9 and the traffic simulation can then be carried out with the use of the following command (listing 3.8).

Listing 3.7: Terminal-Command to Start SUMO (my-scenario.sumo.cfg)

```
sumo -c my-scenario.sumo.cfg
```

Listing 3.8: Traffic Simulation of the Network (my-scenario.sumo.cfg)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<configuration xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
```

```
 xsi:noNamespaceSchemaLocation=
 "http://sumo.sf.net/xsd/sumoConfiguration.xsd">
<configuration>
    <input>
        <net-file value="my-scenario.net.xml"/>
        <route-files value="my-scenario.rou.xml"/>
    </input>
    <time>
        <begin value="0"/>
        <end value="20000"/>
    </time>
         <time-to-teleport value="-1"/>
</configuration>
```

The application SUMO-GUI is the other way to execute the traffic simulation with SUMO. During the execution each vehicular movement and the traffic progression can be observed and the possible bottlenecks can be visually identified. A configuration file for all execution actions, like my-scenario.sumo.cfg , is required in SUMO-GUI. Running the command of listing 3.10, SUMO-GUI will be activated and a work window will be automatically open. The investigated network can be activated by opening the corresponding configuration file under the File-Menu of the menu bar. Traffic simulation can then be performed by pressing the green triangle button in the main tool bar. The simulation can be stopped any time when the user presses the red squared button. A stopped simulation can be further performed by pressing green triangle, if the simulation time is not up. An illustration example is given in Figure 3.5.

Listing 3.9: Terminal-Bash-Command to Start SUMO (my-scenario.sumo.cfg)

```
sumo-gui -c my-scenario.sumo.cfg
```

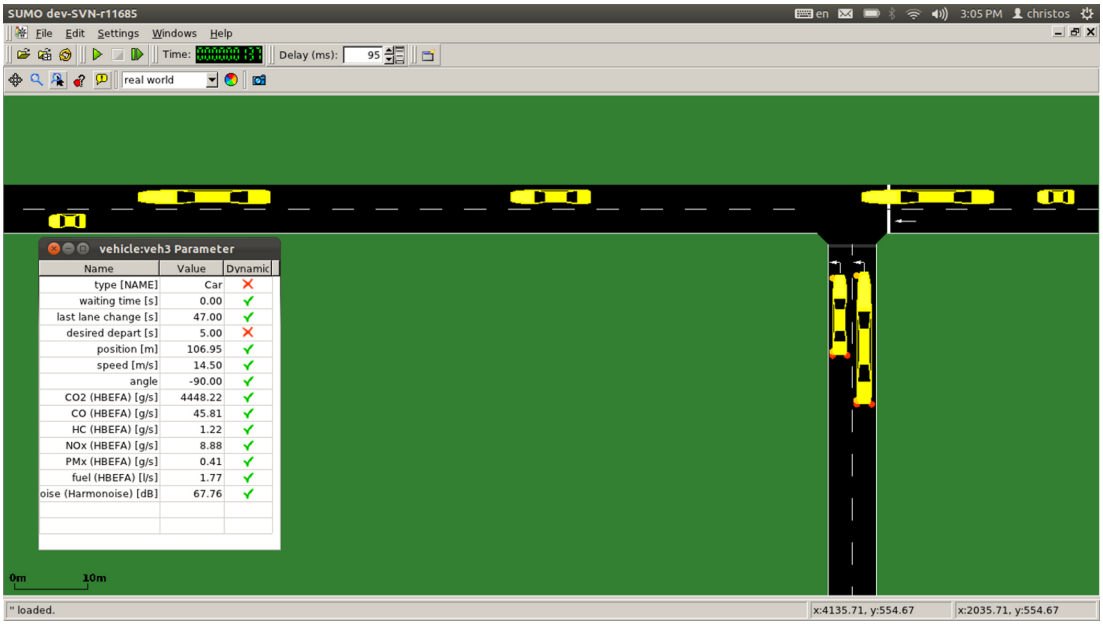

Figure 3.5: SUMO-GUI Running Simulation

3.2 MOVE

3.2.1 About MOVE

Move (MObility model generator for VEhicular networks) is a tool to facilitate users to rapidly generate realistic mobility models for VANET simulations. MOVE is built on top of an open source micro-traffic simulator SUMO. The output of MOVE is a mobility trace file that contains information of realistic vehicle movements which can be immediately used by popular simulation tool NS-2. In addition, MOVE provides a set of GUIs that allows the user to quickly generate realistic simulation scenarios without the hassle of writing simulation scripts as well as learning about the internal details of the simulator. MOVE has being written into Java, so it can be run in every operation system, with the requirement that a JVM (Java Virtual Machine) has been already installed [23].

3.2.2 Create Scenario for NS3

Because NS-3 it doesn't have the ability to create vehicular-nodes is being made a traffic scenario with moving cars using SUMO, after this scenario is being imported to MOVE which export NS-2 trace files, in the end it being connected NS-2 tcl files to NS-3 with ns2-mobility-helper class, which make this connection.

To open the MOVE you have to put the fellow command

Listing 3.10: Terminal-Command to Start MOVE

```
java -jar MOVE.jar
```

For more details about how to use MOVE see Appendix B.

Chapter 4

Routing Protocols in NS3

4.1 VANET Simulations in NS-3

At this moment, it is not yet possible to simulate a complete WAVE device in NS-3, however MAC and PHY extension provided by the IEEE 802.11p are already available. Regarding users mobility, NS-3 supports simple mobility models (i.e. constant position, constant acceleration and constant velocity) as well as more complex models such as random walk, random direction and random waypoints. No mobility models specific for vehicular scenarios are included into the simulator. However, it is possible to generate a mobility trace by using external tools (SUMO, MOVE) and use it during the simulation.

Furthermore MANET routing protocols such as AODV, OLSR, DSDV and DSR are also supported. In this section present the AODV, OLSR, DSDV, DSR and Extended-GPSR (E-GPSR). Starting from these promises, it seems that with the current version of NS-3 it is possible working with VANETs by exploiting the wifi model with IEEE 802.11p extensions, traces defining the mobility of cars belonging to such network and choosing one of the four available MANET routing protocols.

The comparison of a VANET routing protocol with the available routing protocols is not very righteous, the reason is that AODV, OLSR, DSDV and DSR routing protocols have been taken for MANETs, so might not have good performance using a VANET simulation. According of that it can be understood the needing of implement new routing algorithms for NS-3,which are able to perform VANET simulations.

4.2 AODV Routing Protocol

4.2.1 AODV Routing

NS-2 has an already implemented and available model for AODV. This model implements the base specification of RFC-3561 [28]. The model was written by Elena Buchatskaia and Pavel Boyko of ITTP RAS, and is based on the NS-2 AODV model developed by the CMU/MONARCH group and optimized and tuned by Samir Das and Mahesh Marina, University of Cincinnati and also on the AODV-UU implementation by Erik Nordstrm of Uppsala University.

4.2.2 AODV Routing Overview

The source code for the AODV model it can be found in the directory "src/aodv" and the class hierarchy is showing into the figure 4.1.

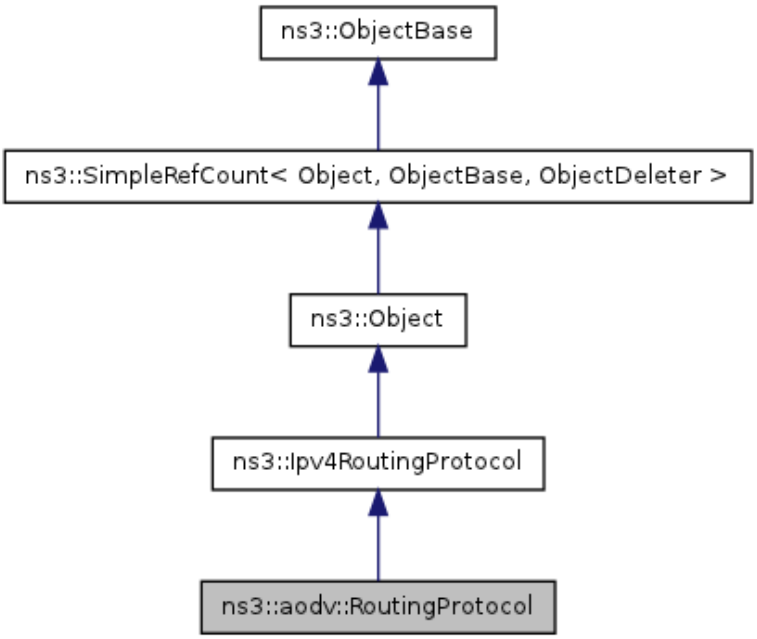

Figure 4.1: AODV Hierarchy Diagram

Class ns3::aodv::RoutingProtocol implements all functionality of service packet exchange and inherits from ns3::Ipv4RoutingProtocol. The base class defines two virtual functions for packet routing and forwarding. The first one, ns3::aodv::RouteOutput, is used for locally originated packets and the second one, ns3::aodv::RouteInput, is used for forwarding and/or delivering received packets.

Protocol operation depends on many adjustable parameters. Parameters for this functionality are attributes of ns3::aodv::RoutingProtocol. Parameter default values are drawn from the RFC and allow the enabling/disabling protocol features, such as broadcasting HELLO messages, broadcasting data packets and so on.

AODV discovers routes on demand. Therefore, the AODV model buffers all packets while a route request packet (RREQ) is disseminated. A packet queue is implemented in aodv-rqueue.cc. A smart pointer to the packet, ns3::Ipv4RoutingProtocol::ErrorCallback, ns3::Ipv4RoutingProtocol::UnicastForwardCallback, and the IP header are stored in this queue. The packet queue implements garbage collection of old packets and a queue size limit.

The routing table implementation supports garbage collection of old entries and state machine, defined in the standard. It is implemented as a STL map container. The key is a destination IP address.

Some elements of protocol operation are not described in the RFC. These elements generally concern cooperation of different OSI model layers. If the node receives an RREQ is a neighbour, the cause may be a unidirectional link. This AODV implementation can detect the presence of unidirectional links and avoid them if necessary. Protocol operation strongly depends on broken link detection mechanism.

The model implements two such heuristics. First, this implementation support HELLO messages. However HELLO messages are not a good way to perform

neighbour sensing in a wireless environment (at least not over 802.11). Therefore, one may experience bad performance when running over wireless. There are several reasons for this: 1) HELLO messages are broadcasted. In 802.11, broadcasting is often done at a lower bit rate than unicasting, thus HELLO messages can travel further than unicast data, 2) HELLO messages are small, thus less prone to bit errors than data transmissions and 3) Broadcast transmissions are not guaranteed to be bidirectional, unlike unicast transmissions. Second, we use layer 2 feedback when possible. Link are considered to be broken if frame transmission results in a transmission failure for all retries. This mechanism is meant for active links and works faster than the first method. The layer 2 feedback implementation relies on the TxErrHeader trace source, currently supported in AdhocWifiMac only.

The model is for IPv4 only. The following optional protocol optimizations are not implemented: 1)Expanding ring search. 2)Local link repair. 3) RREP, RREQ and HELLO message extensions. These techniques require direct access to IP header, which contradicts the assertion from the AODV RFC that AODV works over UDP. This model uses UDP for simplicity, hindering the ability to implement certain protocol optimizations. The model doesn't use low layer raw sockets because they are not portable.

4.2.3 AODV Helper

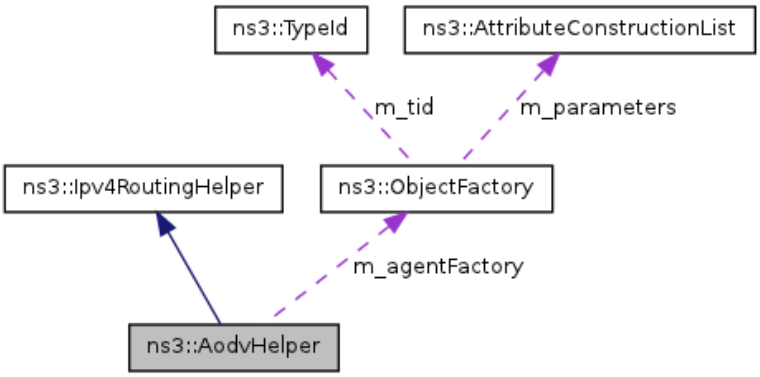

Figure 4.2: AODV Helper Diagram

A helper class for AODV has been written (figure 4.2). After an IPv4 topology has been created and unique IP addresses assigned to each node, the simulation script writer can call the functions to enable AODV (listing 4.1).

Listing 4.1: NS3 AODV Install

```
//Enable aodv
AodvHelper aodv;
//you can configure AODV attributes
//here using aodv.Set(name, value)
InternetStackHelper internet;
internet.SetRoutingHelper (aodv);
internet.Install (c);
//where c is a node container
```

In addition, the behaviour of AODV can be modified by changing certain attributes with the follow functions:

- void SetMaxQueueTime(Time t)
  - MaxQueueTime: maximum time packets can be queued (in seconds),
  - set with class: TimeValue,
  - underlying type: Time,
  - initial value: +30000000000.0ns,
  - flags: construct write read.
- void SetMaxQueueLen(uint32-t len)
  - MaxQueueLen: maximum number of packets that we allow a routing protocol to buffer,
  - set with class: ns3::UintegerValue,
  - underlying type: uint32-t 0:4294967295,

- initial value: 64,
- flags: construct write read.

- void SetDesinationOnlyFlag(bool f)
  - DestinationOnly: indicates only the destination may respond to this RREQ,
  - set with class: BooleanValue,
  - underlying type: bool,
  - initial value: false,
  - flags: construct write read.

- void SetGratuitousReplyFlag(bool f)
  - GratuitousReply: indicates whether a gratuitous RREP should be unicast to the node originated route discovery,
  - set with class: BooleanValue,
  - underlying type: bool,
  - initial value: true,
  - flags: construct write read.

- void SetHelloEnable(bool f)
  - EnableHello: indicates whether a hello messages enable,
  - set with class: BooleanValue,
  - underlying type: bool,
  - initial value: true,
  - flags: construct write read.

- void SetBroadcastEnable(bool f)
  - EnableBroadcast: indicates whether a broadcast data packets forwarding enable,

– set with class: BooleanValue,

– underlying type: bool,

– initial value: true,

– flags: construct write read.

4.3 OLSR Routing Protocol

4.3.1 OLSR Routing

NS-3 has an already implemented and available model for OLSR. This model implements the base specification of RFC-3626 [6]. OLSR has been developed at the University of Murcia (Spain) by Francisco J. Ros for NS-2, and was ported to NS-3 by Gustavo Carneiro at INESC Porto (Portugal).

4.3.2 OLSR Routing Overview

The source code for the OLSR model it can be found in the directory src/olsr and the class hierarchy is shown in figure 4.3.

The model is for IPv4 only. Mostly compliant with OLSR as documented in [6] about the use of multiple interfaces that was not supported by the NS-2 version, but is supported in NS-3; OLSR does not respond to the routing event notifications corresponding to dynamic interface up and down (ns3::RoutingProtocol:: NotifyInterfaceUp and ns3::RoutingProtocol::NotifyInterfaceDown) or address insertion/removal ns3::RoutingProtocol::NotifyAddAddress and ns3:: RoutingProtocol::NotifyRemoveAddress). Unlike the NS-2 version, does not yet support MAC layer feedback as described in [6]; Host Network Association (HNA) is supported in this implementation of OLSR. Refer to examples/olsr-hna.cc to see how the API is used.

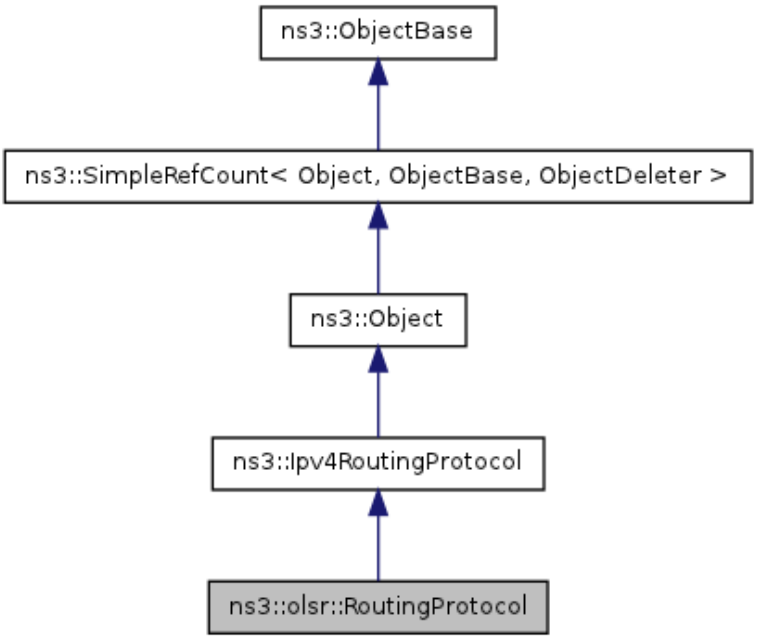

Figure 4.3: OLSR Hierarchy Diagram

4.3.3 OLSR Helper

A helper class for OLSR has been written. After an IPv4 topology has been created and unique IP addresses assigned to each node, the simulation script writer can call one of three overloaded functions with different scope to enable OLSR: ns3::OlsrHelper::Install (NodeContainer container), ns3::OlsrHelper::Install(node) or ns3::OlsrHelper::InstallAll (void)

Listing 4.2: NS3 OLSR Install in Node Container

```
// Enable olsr
OlsrHelper olsr;
Ipv4StaticRoutingHelper staticRouting;
Ipv4ListRoutingHelper list;
list.Add(staticRouting, 0);
list.Add(olsr, 10);
InternetStackHelper internet;
internet.SetRoutingHelper(list);
internet.Install(c);
```

```
    //where c is a node container
```

In addition, the behavior of OLSR can be modified by changing certain attributes. The method ns3::OlsrHelper::Set () can be used to set OLSR attributes. These include HelloInterval, TcInterval, MidInterval, Willingness. Other parameters are defined as macros in olsr-routing-protocol.cc. Attributes defined for this type:

- HelloInterval: HELLO messages emission interval
  - set with class: TimeValue,
  - underlying type: Time,
  - initial value: +2000000000.0ns,
  - flags: construct write read.
- TcInterval: TC messages emission interval
  - set with class: TimeValue,
  - underlying type: Time,
  - initial value: +5000000000.0ns,
  - flags: construct write read.
- MidInterval: MID messages emission interval. Normally it is equal to TcInterval
  - set with class: TimeValue,
  - underlying type: Time,
  - initial value: +5000000000.0ns,
  - flags: construct write read.
- Willingness: Willingness of a node to carry and forward traffic for other nodes
  - set with class: ns3::EnumValue,

– underlying type: never—low—default—high—always,

– initial value: defaultm

– flags: construct write read.

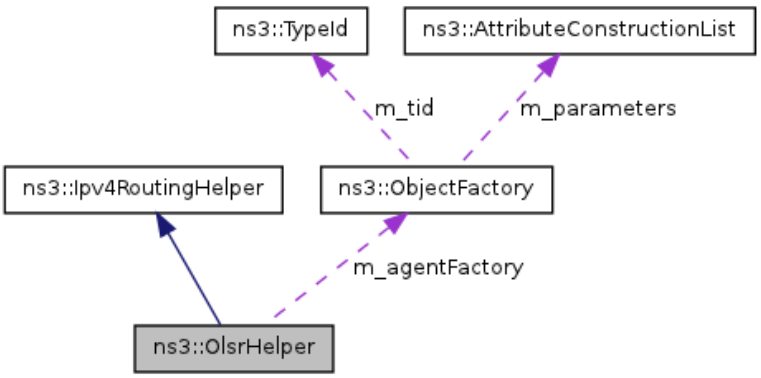

Figure 4.4: OLSR Helper Diagram

4.4 DSDV

4.4.1 DSDV Routing

DSDV routing protocol is a pro-active, table-driven routing protocol for MANETs developed by Charles E. Perkins and Pravin Bhagwat in 1994 [29]. It uses the hop count as metric in route-selection. This model was developed by the ResiliNets research group at the University of Kansas [33]. A paper on this model exists at [24].

4.4.2 DSDV Routing Overview

Every node will maintain a table listing all the other nodes it has known either directly or through some neighbours. Every node has a single entry in the routing table. The entry will have information about the node's IP address, last known sequence number and the hop count to reach that node. Along with these details the table also keeps track of the next-hop neighbour to reach the destination node, the time-stamp of the last update received for that node.

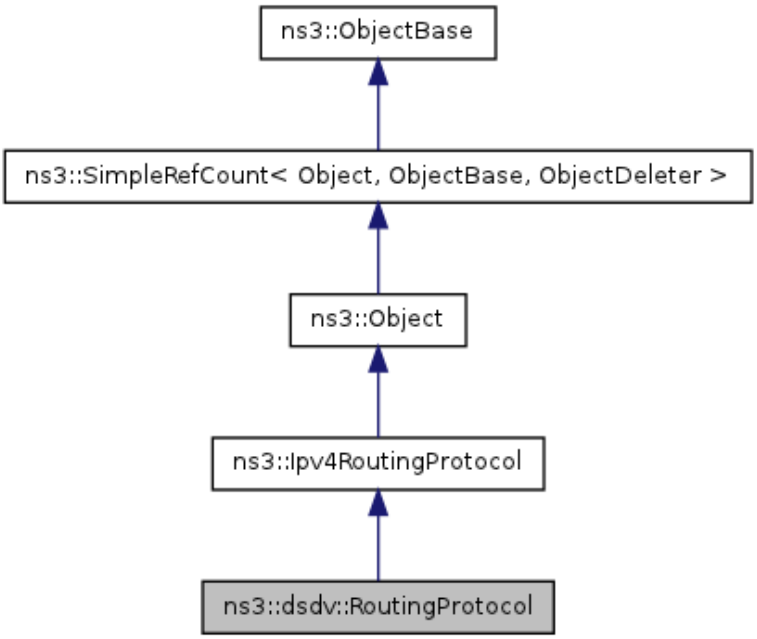

Figure 4.5: DSDV Hierarchy Diagram

The DSDV update message consists of three fields, Destination Address, Sequence Number and Hop Count. Each node uses 2 mechanisms to send out the DSDV updates. They are:

1. Periodic updates which are sent out after every m-periodicUpdateInterval (default:15s). In this update the node broadcasts out its entire routing table.

2. Trigger updates which are small updates in-between the periodic updates. These updates are sent out whenever a node receives a DSDV packet that caused a change in its routing table. The original paper did not clearly mention, when for what change in the table should a DSDV update be sent out. The current implementation sends out an update irrespective of the change in the routing table.

The updates are accepted based on the metric for a particular node. The first factor determining the acceptance of an update is the sequence number. It has to accept the update if the sequence number of the update message is higher

irrespective of the metric. If the update with same sequence number is received, then the update with least metric (hopCount) is given precedence.

In highly mobile scenarios, there is a high chance of route fluctuations, thus it has the concept of weighted settling time where an update with change in metric will not be advertised to neighbours. The node waits for the settling time to make sure that it did not receive the update from its old neighbour before sending out that update.

The current implementation covers all the above features of DSDV. The current implementation also has a request queue to buffer packets that have no routes to destination. The default is set to buffer up to 5 packets per destination.

4.4.3 DSDV Helper

A helper class for DSDV has been written (figure 4.6). After an IPv4 topology has been created and unique IP addresses assigned to each node, the simulation script writer can call the functions to enable DSDV (listing 4.3).

Listing 4.3: NS3 DSDV Install

```
//Enalbe DSDV
DsdvHelper dsdv;
dsdv.Set ("PeriodicUpdateInterval",
  TimeValue (Seconds (periodicUpdateInterval)));
dsdv.Set ("SettlingTime",
  TimeValue (Seconds (settlingTime)));
InternetStackHelper internet;
 internet.SetRoutingHelper (dsdv);
 internet.Install (c);
```

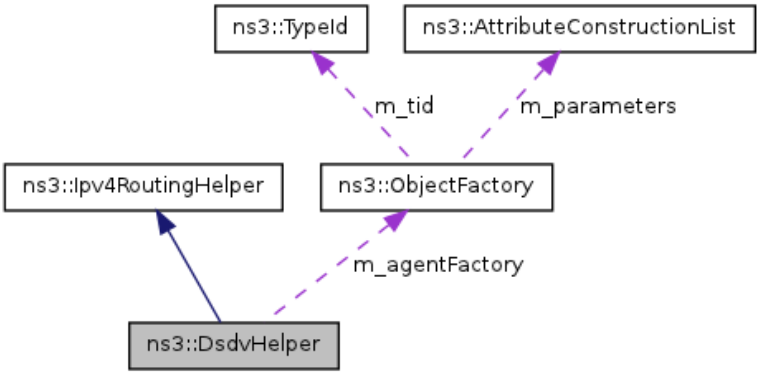

Figure 4.6: DSDV Helper Collaboration Diagram

4.5 DSR

4.5.1 DSR Routing

DSR protocol is a reactive routing protocol designed specifically for use in multi-hop wireless ad hoc networks of mobile nodes. This model was developed by the ResiliNets research group at the University of Kansas [33].

4.5.2 DSR Routing Overview

This model implements the base specification RFC-4728 [8]. Class dsr:: Dsr-Routing implements all functionality of service packet exchange and inherits Ipl4Protocol. Class dsr::DsrOptions implements functionality of packet processing and talk to DsrRouting to send/receive packets. Class dsr::DsrFsHeader defines the fixed-size header and identify the up-layer protocol. Class dsr::DsrOption Header takes care of different DSR options and process different header according to specification from [8]. Class dsr::DsrSendBuffer is a buffer to save data packet as well as route error packet when there is no route to forward the packet. Class dsr::DsrMaintainBuffer is a buffer to save data packet for next-hop notification when the data packet has already sent out of send buffer Class dsr::RouteCache is the essential part to save routes found for data packet, DSR responds to several routes for a single destination Class dsr::RreqTable implements the functionalities to avoid duplicate route requests and control route request rate for a single

destination.

Protocol operation depends on the many adjustable parameters. It support parameters, with their default values, from [8] and parameters that enable/disable protocol features or tune for specific simulation scenarios, such as the max size of send buffer and its timeout value. The full parameter list is in DsrRouting.cc file.

DSR discovers routes totally on demand. Therefore, DSR model buffers all packets, while a RREQ is disseminated. It has implemented a packet buffer in dsr-rsendbuff.cc. The packet queue implements garbage collection of old packets and a queue size limit. When the packet is sent out from the send buffer, it will be queued in maintenance buffer for next hop acknowledgement.

Route cache implementation support garbage collection of old entries and state machine ans it has a STL map container. The key is the destination IP address, where protocol operation strongly depends on broken link detection mechanism. It has implemented all the three heuristics. First, uses layer 2 feedback when possible. Link considered to be broken, if frame transmission results in a transmission failure for all retries. This mechanism meant for active links and work much more faster, than first method. Layer 2 feedback implementation relies on TxErrHeader trace source, currently it is supported in AdhocWifiMac only. Second, passive acknowledgement should be used whenever possible. The node turns on "promiscuous" receive mode, in which it can receive packet not destined for itself, and when the node assures the delivery of that data packet to its destination, it cancels the passive acknowledgement timer. Last, we use network layer acknowledge scheme to notify the receipt of a packet. Route request packet will not be acknowledge or retransmitted.

Following optional protocol optimizations aren't implemented:

- Flow state.
- First Hop External (F), Last Hop External (L) flags.

- Handling unknown DSR options.
- Two types of error headers:
  1. flow state not supported option,
  2. unsupported option (not going to happen in simulation).

DSR operates with direct access to IP header, and operates between network and transport layer. It has also some implementation changes:

- the DsrFsHeader has added 3 fields: message type, source id, destination id, and these changes only for post-processing,
- message type is used to identify the data packet from control packet,
- source id is used to identify the real source of the data packet since we have to deliver the packet hop-by-hop and the ipv4header is not carrying the real source and destination ip address as needed,
- destination id is for same reason of above.

4.5.3 DSR Helper

It has not implemented a helper class yet. In order to create a scenario, the following should be kept in mind when running DSR as routing protocol:

- NodeTraversalTime is the time it takes to traverse two neighboring nodes and should be chosen to fit the transmission range.
- PassiveAckTimeout is the time a packet in maintenance buffer wait for passive acknowledgment, normally set as two times of NodeTraversalTime.
- RouteCacheTimeout should be set smaller value when the nodes' velocity become higher. The default value is 300s.

In order to make a node run DSR, the easiest way would be to use the ClickInternetStackHelper class in your simulation script (listing 4.4).

Listing 4.4: NS3 DSR Install

```
        //Enalbe DSR
InternetStackHelper internet;
DsrMainHelper dsrMain;
DsrHelper dsr;
internet.Install (c);
dsrMain.Install (dsr, c);
```

The example scripts inside "src/dsr/examples/" demonstrate the use of DSR based nodes in different scenarios. The helper source can be found inside "src/dsr/ helper/dsr-main-helper.h,cc" and "src/dsr/helper/dsr-helper.h,cc". The script dsr.cc use DSR as routing protocol within a traditional MANETs environment. DSR is also built in the routing comparison case in "examples/routing/" where manet-routing-compare.cc is a comparison case with built in MANET routing protocols and can generate its own results.

The model has been tested as follows:

- Unit tests have been written to verify the internals of DSR. This can be found in "src/dsr/test/dsr-test-suite.cc". These tests verify whether the methods inside DSR module which deal with packet buffer, headers work correctly.
- Simulation cases have been tested and have comparable results.
- The manet-routing-compare.cc has been used to compare DSR with three of other routing protocols.

4.6 E-GPSR

4.6.1 Theoretical Description of Protocol

E-GPSR is an extension of GPSR, as have been mentioned in the first chapter greedy forward is a strategy that do not create a path from source to the des-

tination, they find the next hop considering some parameters about position of other nodes (i.e the closest neighbour to the destination) and forward the packet to the next-hop.

With greedy forwarding a path could be created very quickly but its has the disadvantage of a possible failure to find a path [4], for this reason the routing protocol has to be guided to find a correct direction. A good way to avoid this failure is take place some over information except the distance from destination to decide the next hop of the path, so you can increase the possibility to find a path. Taking into that if a node has a lot of neighbours is more possible to communicate with a node that has connection with the destination, this could be a good information to take care.

For the E-GPSR of this thesis a function has been made to measure the quality of next-hop, which is:

$$NextHopMetric = d(j, dest) - (w * N_j)$$

- j identifies a neighbour and dest is the destination node,
- d (i, dest) is the distance between the j-node and the destination,
- Nj is the number of neighbour of the j device,
- w represents a weight that expresses the importance of the Neighbours.

The neighbour with the lowest result of this function, is the next hop for the packet.

Several simulations have been created with different weights to find the most efficient weight, in this case weight equal to zero means the classical GPSR. After checking weights from 0 to 15, as showing to figure 4.7, it have been found that a weight equal to 2 has the best performance and is also better than zero (which is the GPSR). The simulation scenarios had the follow attributes :

- density of nodes within 1 to 7,

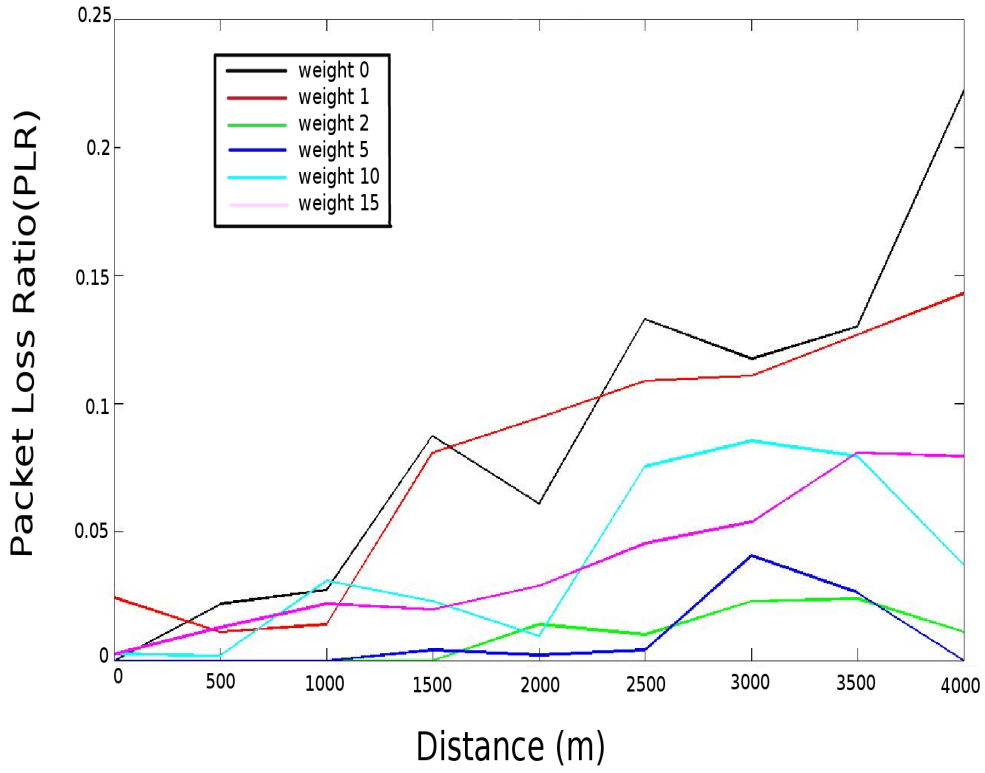

Figure 4.7: PLR with Different Weights for Neighbours

- number of UDP client-server applications that start transmit in same time from 1 to 10,
- borders of topology from 1000m to 4001m,
- 150 different seeds which means 150 different sequences of random place UDP client-servers setting up.

The number of nodes had been taken from this formula:

$$N = density * ((border^2)/(distancebetweenNeighbours^2)$$

with this formula a consistency number of nodes are taken place in order to have logical topology. A topology example of simulation could be seen in figure 2.5, in which also showing two path (with red and blue) of UDP client server application coloured by red.

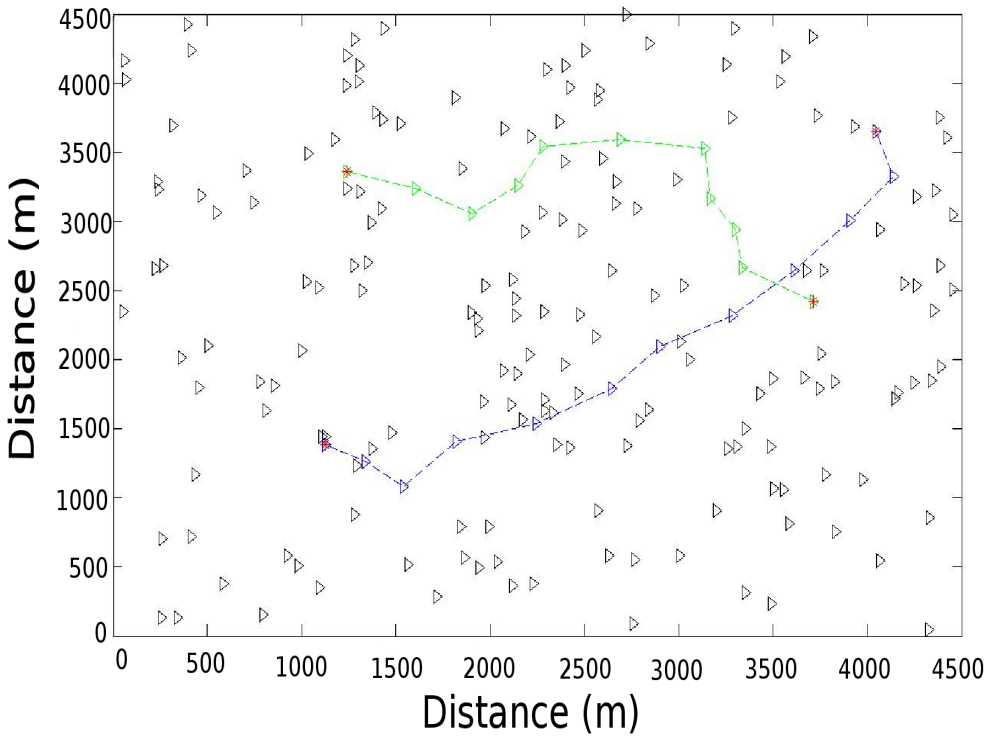

Figure 4.8: Topology in Simulation about Different Weights

4.6.2 Implementation

E-GPSR has been design into NS-3 with the follow operation:

- calculate the function of quality for every neighbour with help of GPS,
- finds the next hop that could communicate with destination according the results of the above function with help of static routing protocol,
- set a Time To Live (TTL) for each next hop,
- remove the next hop when TTL expired.

Also the ns-3::ipv4-L3 and ns-3::Internet Stack has been Configured, in order to make the follow operations:

- install topology based routing protocol into each node,

- checks if topology-based routing protocol has been set:
  - get the Ip address of the source from the packet,
  - get the Ip address of the destination from the packet,
- checks if the current node has not a next hop:
  - find a new next hop,
  - set TTL for next hop,
- checks if TTL is equal to zero:
  - find a new next hop,
  - set TTL for next hop,
- sends the packet to next hop and reduce by one the TTL.

An overview of the classes included in the implementation can be seen into figure 4.9.

To take position information of nodes for the E-GPSR it needs to provide a GPS device that has this informations at any time. This device it could be an unit on board of car or a location service base, so in this way every node has to communicate with an infrastructure base.

Consequently in NS-3, GPS is an ideal device that knows the position of each node and have the follow operations:

- measure the distance between two nodes,
- could give the number of neighbours of a given node.

GPS module uses the function SetNetDeviceContainer() to insert into a vector pointers to net devices from a NetDeviceContainer, so GPS could have information about theirs position and speed at any time, furthermore it have SetNetDevice() function to put into vector of a pointer of just one device.

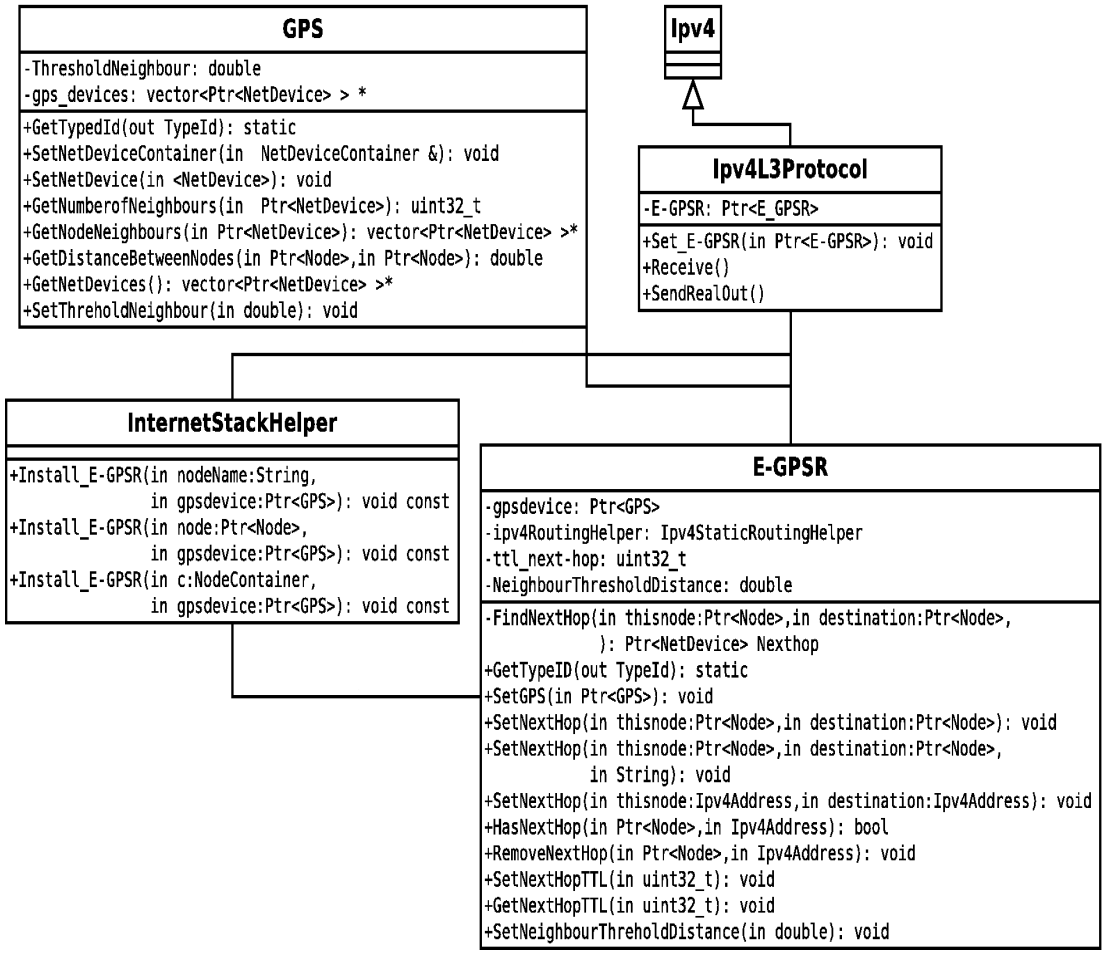

Figure 4.9: Position-Based UML Diagram

The E-GPSR needs to know the the number of neighbours about a given node to calculate the efficiency function. So GPS class provides the GetNumberOfNeighbours() function which returns a unsighted integer with the number of neighbours, GPS also can give a vector with the Neighbours of given node using the GetNodeNeighbours() function, data that the routing protocol needs to find the next-hop. GPS in order to find if a node is neighbour with another needs to know the distance between two nodes so the function GetDistanceBetweenNodes() provides this information. The GPS has a threshold within its decides whether or not a node neighbouring with others. To set this threshold SetThresholdNeighbour() function has to be used, where the threshold is the maximum distance in

which two nodes can communicate so they are neighbours.

Internet Stack Helper has been configure importing the InstallE-GPSR() function to create an E-GPSR routing protocol into every node. This could be done by giving a node container, a node itself or a string that includes the name of the nodes (if this has been set up to ns3).

Ipv4-l3 module has been configured to use the E-GPSR routing protocol. First it has added a pointer which shows into E-GPSR object, so it could be use for routing packets, for this reason SetE-GPSR() function has been insert to the ipv4-l3-protocol class. Also the Receive() function has been configured that is the first function called when a packet is being received from the up or down layer. The configuration makes this function to check if E-GPSR has been setting up, so its forward the packet within this routing protocol. Also its checks if there is a next hop that could communicate with the destination, using the HasNextHop(), function of E-GPSR which returns true or false about the existence of a next hop node that could communicate with the destination of the packet. In case that there are no next hop, the SetNextHop() function is used which invokes the FindNextHop() function to find the next hop that it could communicate with the destination. After this the SetNextHopTTL() function is called to set a TTL to the next hop when this TTL expired the next hop is deleted (so a new next-hop has to be found).

Another function that have been modified is SendRealOut() this function is being called then the Ipv4-l3 sends the packet. It have been modified so it checks if E-GPSR has been set, in this case using the GetNextHopTTL() function checks if the TTL of the next hop has been expired. If so it looking for a new next hop and a TTL for this hop using the SetNextHop() and SetNextHopTTL() functions of E-GPSR, if next hop TTL has not been expired its forwards the packet to next hop reducing the TTL.

In figure 4.10 is showing how the algorithm finds a path by next-hoping the packet, with green is noticed the neighbours.

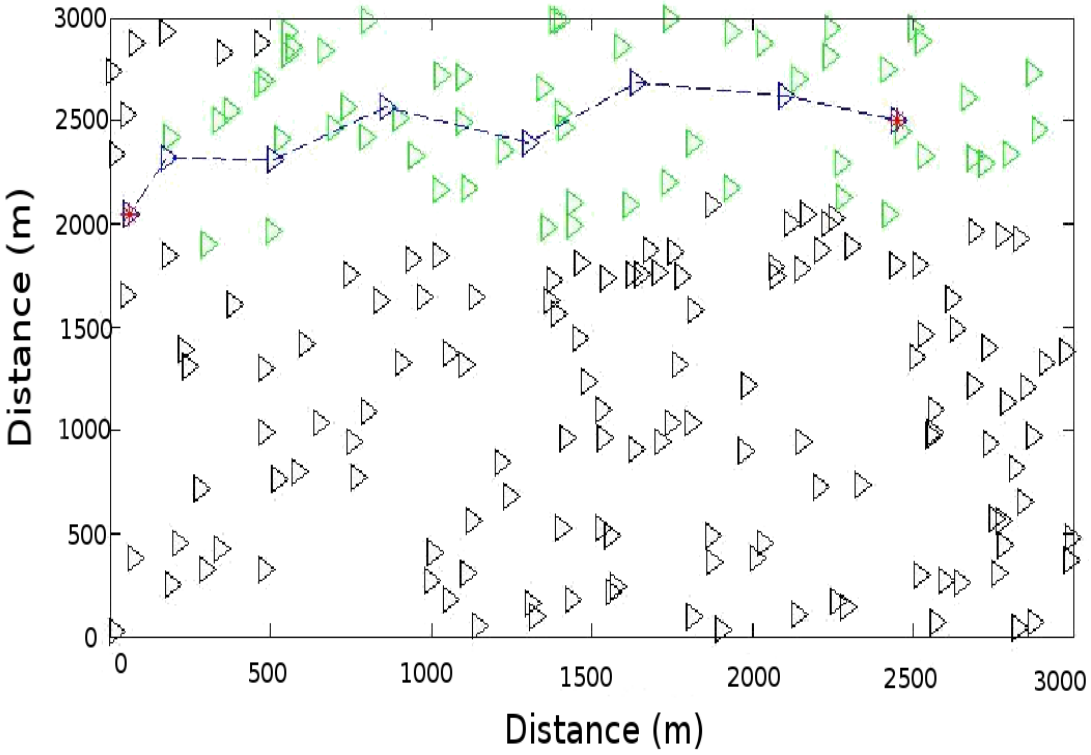

Figure 4.10: Routing Path

4.6.3 Configuration

In the E-GPSR it could be changed the follow parameters:

- Neighbour Distance, which is the maximum distance where two neighbour nodes can communicate. The function about that is SetThresholdNeighbour() of class GPS.
- Next Hop TTL, which is the time that a next hop consider reliable. The function about that is SetNextHopTTL() of class Position Based Routing Protocol.

In listing 4.5 is showing how to install E-GPSR into a NS-3 scenario.

Listing 4.5: Install E-GPSR into NS-3

```
// Enable position-based routing protocol
```

```
//Set a GPS Device to the wifi Devices
Ptr<GPS> gpsdevice = CreateObject<GPS > () ;
gpsdevice->SetNetDeviceContainer(fdevices);
//where fdevices is a Net-Device Container
Ptr<EGPSR>
EGPSR =
CreateObject<EGPSR> ();
EGPSR->SetGps(gpsdevice);
InternetStackHelper internet;
internet.SetTopologyBasedRoutingProtocol
(EGPSR);
internet.EGPSR (c);
//where c is a node container
```

Chapter 5

Performance Evaluation

5.1 Transmission Range

5.1.1 MAC and PHY Configuration

In this section is being described how a scenario could be configure in NS-3 to establish MAC and PHY layer. In next section is showing simulation results about transmission range using two nodes which use UDP client-server.

The IEEE 802.11p specification has introduced only few amendments to the common IEEE 802.11 standard. For this reason, MAC and PHY layers for VANETs can be easily configured and installed by using dedicated helper classes of NS-3 already available for the WiFi module:

- WifiMacHelper,
- YansWifiPhyHelper,
- WiFiHelper.

So the QosWiFiMacHelper, which inherits directly from the Wifi-MacHelper, is being model the MAC layer based on the EDCA. With respect to the original EDCA specification, described in [2], the IEEE 802.11p suggests a different set of CWAIFS parameters, as reported in Tab.1 [3]. YansWifiPhyHelper, instead, creates and configures the PHY layer.

In order to enabling IEEE 802.11p amendments for both MAC and PHY layers, it is necessary to set:

- the IEEE 802 specification,

- the communication channel CCH or SCH.

This could be done by calling the SetStarndard() function of WifiMacHelper. Finally, the method Install() of the aforementioned class, receiving as input parameters the WiFi MAC helper, the PHY helper, and the node container created into the scenario, handles the effective set up of 802.11p nodes by creating network devices where MAC and PHY models, before configured are installed.

5.1.2 Simulation Details

It is important to discover the maximum transmission range of a WiFi device which is strictly influenced by the physical transmission rate. The reason is that Routing protocols is able to make routing paths only when nodes is able to communicate each other, and this could be happen only inside of the nodes' transmission range. In order to investigate this aspect, it have been deployed two main cases of scenarios composed by only two nodes on which an UDP client-server application that generates packets of 640 Bytes every 0.04 s have been set up. In the first case it have used different physical modulation schemes and in the second different wifi control managers.

Packet Loss Rate (PLR) have been measured of UDP application by varying the distance between nodes and the transmission rates. Note that the PLR gives an idea on the transmission range, if the PLR is closer to 0, communicating nodes are into the same transmission range otherwise is out of range. The Figure 5.1 shows PLR of a UDP client-server application with different physical modulation scheme. The Figure 5.2 shows PLR of a UDP client-server application with different wifi control algorithms.

5.1.3 Simulation Results

As reported in figure 5.1, which is demonstrates that the PLR increases as both physical transmission rate and distance between devices increase. Despite smaller physical rates guarantee higher transmission ranges, they could limit significantly the user throughput. On the other hand, higher modulation schemes can reach

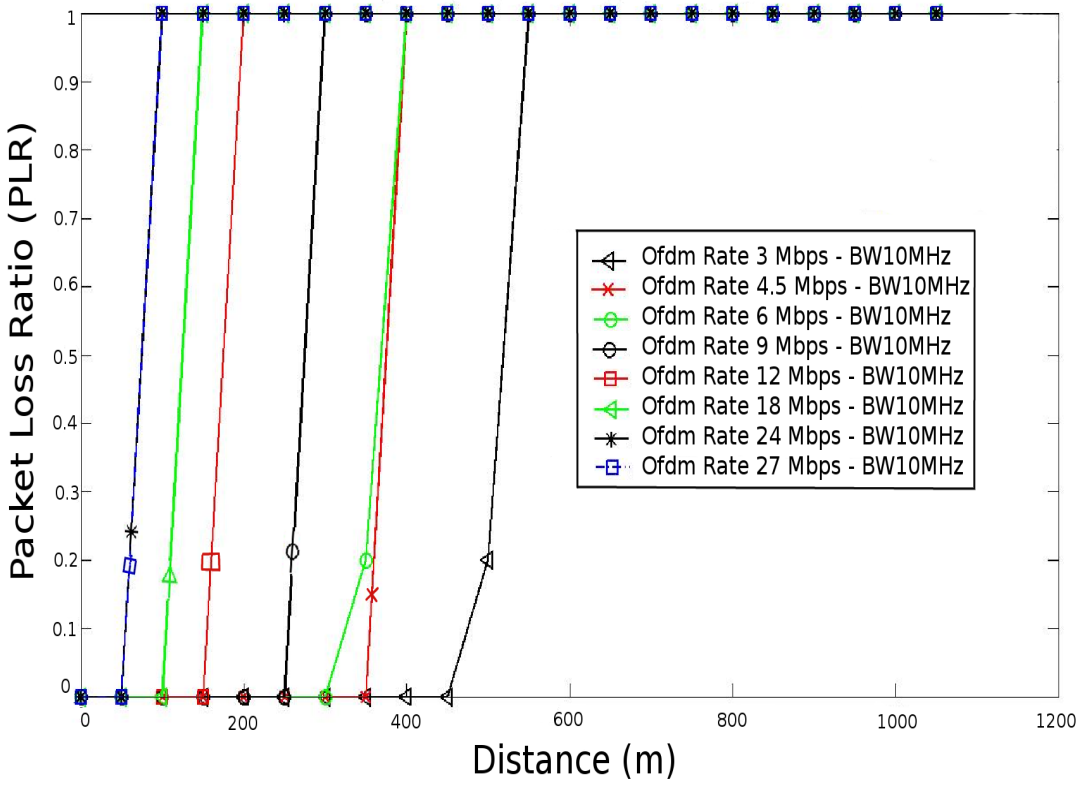

Figure 5.1: PLR with Different Modulation Schemes

higher throughput only for very short distances between vehicles. It emerges that the use of constant PHY settings could break down network performance, especially in vehicular scenarios where the density of nodes and network conditions vary quickly.

An algorithm that adapts the physical transmission rate during the time can be exploited for resolving the problem. Furthermore in figure 5.2 shows for different wifi control algorithms the PLR is quite the same. So we have follow the assumption: only when the distance between two nodes is bellow 500m could be in the same transmission range.

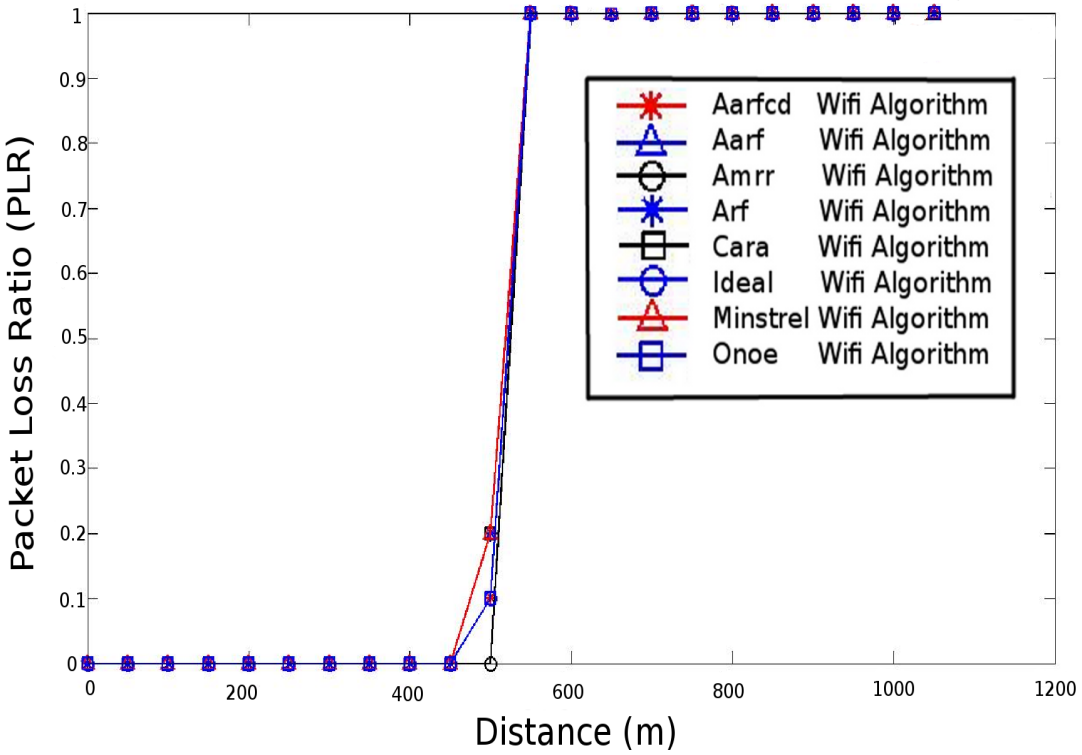

Figure 5.2: PLR with Different wifi Manager Algorithms

5.1.4 NS-3 Code

NS-3 supports several PHY rate control algorithms such as ARF (Auto Rate Fallback), AARF (Adaptive ARF), and AARF-CD (AARF with Collision Detection). They have been conceived for adjusting the physical rate according to the status of the transmission (i.e., the number of delivered/lost packets). The SetRemoteStationManager() function of the WiFiHelper class is used for selecting one of these strategies as showing into the listing 5.1 and 5.2

Listing 5.1: Set-up Wifi - NS-3 Code

```
NodeContainer c;
c.Create (numNodes);
// The below set of helpers will help us
// to put together the wifi NICs want
WifiHelper wifi;
```

```
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
// set it to zero; otherwise, gain will be added
wifiPhy.Set ("RxGain", DoubleValue (-10) );

wifiPhy.SetPcapDataLinkType
    (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

YansWifiChannelHelper wifiChannel ;
wifiChannel.SetPropagationDelay
    ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss
    ("ns3::FriisPropagationLossModel");
wifiPhy.SetChannel (wifiChannel.Create ());

NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
//set 802.11 p Service Channel
wifi.SetStandard (WIFI_PHY_STANDARD_80211p_SCH);
//set wifi manager algorithm
wifi.SetRemoteStationManager (algorithm);
/*
available wifi managers:
ns3::AarfWifiManager, ns3::AarfcdWifiManager,
ns3::AmrrWifiManager, ns3::ArfWifiManager,
ns3::CaraWifiManager, ns3::ConstantRateWifiManager,
ns3::IdealWifiManager, ns3:i:MinstrelWifiManager,
ns3::OnoeWifiManager and ns3::RraaWifiManager.
*/
NetDeviceContainer fdevices = wifi.Install
    (wifiPhy, wifiMac, c);
```

Listing 5.2: NS-3 Install Wifi

```
//node container helps to organize the nodes
```

```
NodeContainer c;
c.Create (numNodes); //Number of nodes

// The below set of helpers will help us
// to put together the wifi NICs we want
WifiHelper wifi;

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
// set it to zero; otherwise, gain will be added
wifiPhy.Set ("RxGain", DoubleValue (-10) );

wifiPhy.SetPcapDataLinkType
     (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

YansWifiChannelHelper wifiChannel ;
wifiChannel.SetPropagationDelay
    ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss
    ("ns3::FriisPropagationLossModel");
wifiPhy.SetChannel (wifiChannel.Create ());

NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager"
,"DataMode",StringValue(Modulation));

/* available  Modulations
   OfdmRate3MbpsBW10MHz
   OfdmRate4_5MbpsBW10MHz
   OfdmRate6MbpsBW10MHz
   OfdmRate9MbpsBW10MHz
   OfdmRate12MbpsBW10MHz
   OfdmRate18MbpsBW10MHz
   OfdmRate24MbpsBW10MHz
```

```
  OfdmRate27MbpsBW10MHz
*/
NetDeviceContainer fdevices = wifi.Install
    (wifiPhy, wifiMac, c);
```

5.2 Static-grid Simulation Scenario

5.2.1 Simulation Details

In order to observer the behaviour of the routing protocols OLSR, AODV and E-GPSR, a good idea is to start with a static and full controlled scenario, with nodes that is not moving.

Several simulations have been made with a grid topology scenario, in order the behaviour of OLSR, AODV and E-GPSR could be compared. In all considered scenarios, UDP client-servers have been set up into the corners of topology (see figure 3.3) where UDP client sends 100 packets of 600 bytes with time interval at 0.04 seconds to UDP server. For the equation of simulations 25 seeds have been measures that means 25 different sequences of random place UDP client-servers setting up. Furthermore in all simulation wifi Ideal Manager have been used. Into the simulations the grid topology have been made with five different grids:

- 4x4 and number of nodes = 16,
- 6x6 and number of nodes = 36,
- 8x8 and number of nodes = 64,
- 10x10 and number of nodes = 100,
- 15x15 and number of nodes = 225.

Also an example of the grid-topology is showing in figure 5.3.

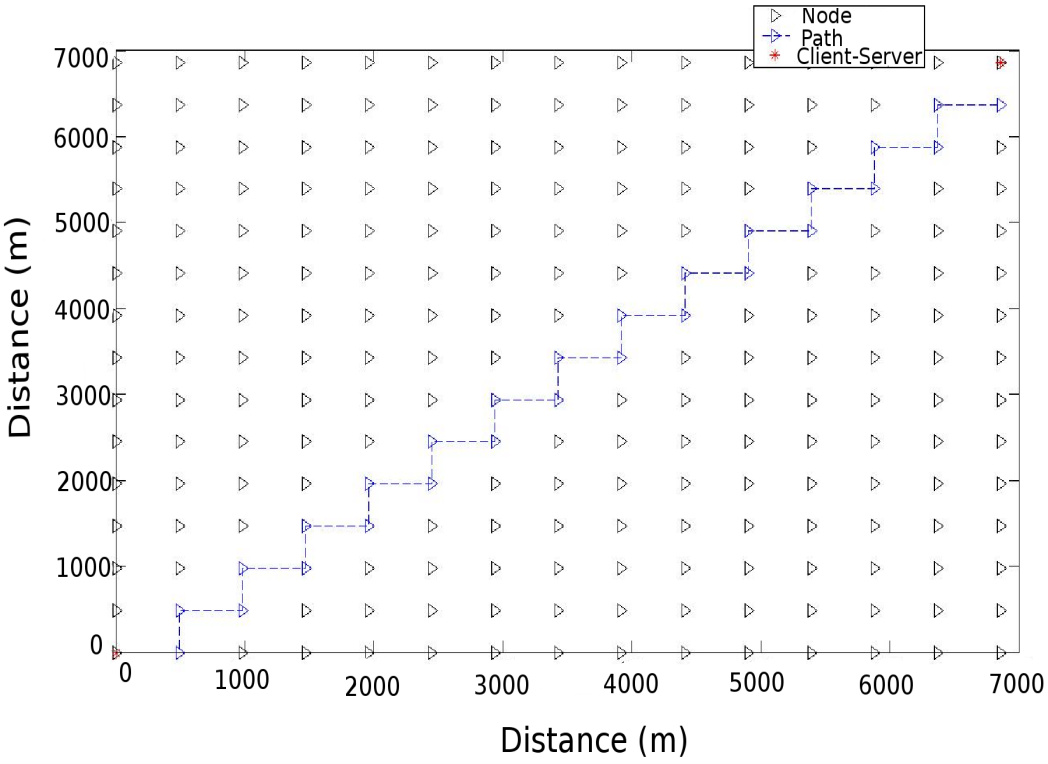

Figure 5.3: Grid Topology

5.2.2 Simulation Results

The PLR is quite the same for all simulations and is showing into the figure 5.3. The distance is the space between each node separately.

According the graph, all routing protocols (AODV, OLSR, E-GPSR) could make the connection path so nodes can send packets within an UDP client-server application, when distance between each node is bellow than 500m, this is a logical situation if we have in our minds the previous result about transmission range. In which has analysed that the maximum distance between two node where is able to send packets is 500m. The point here is that routing protocols is able to create a path among nodes that connect each other.

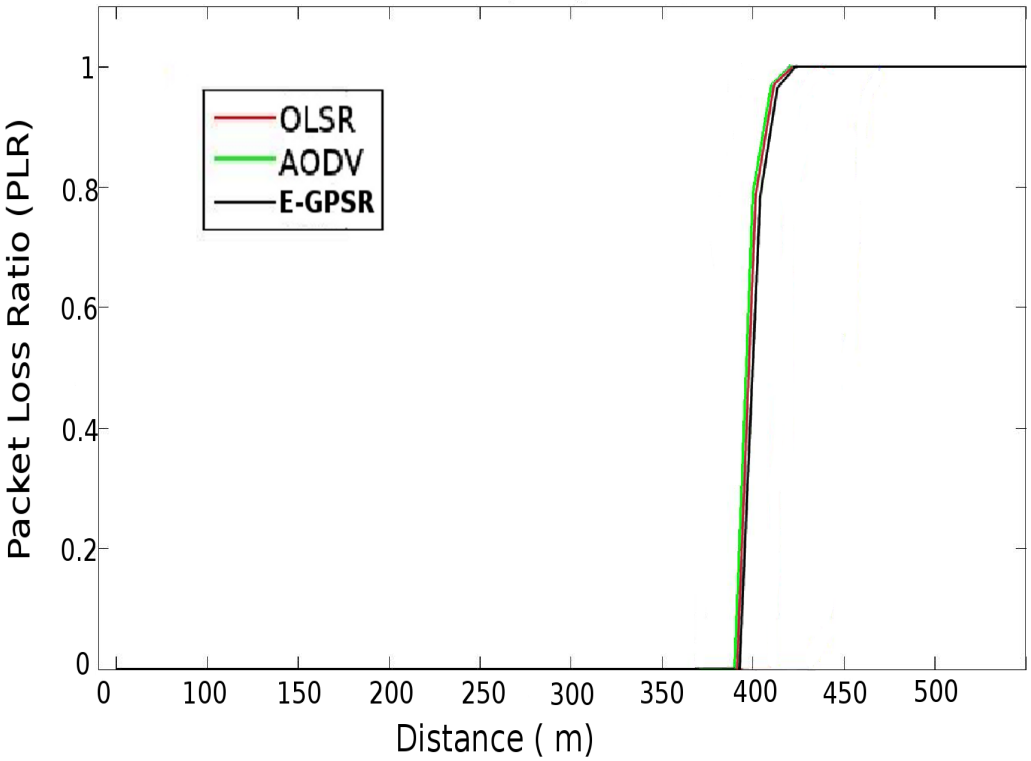

Figure 5.4: PLR with 802.11p and Different Routing Protocols

5.2.3 NS-3 Code

To set up a grid topology the code is showing to listing 5.3.

Listing 5.3: Set-up a Grid Topology - NS-3 Code

```
//make a grid topology
MobilityHelper mobility;
  mobility.SetPositionAllocator(
    "ns3::GridPositionAllocator",
    "MinX", DoubleValue(0.0),
    "MinY", DoubleValue(0.0),
    "DeltaX", DoubleValue(distance),//distance for X in m
    "DeltaY", DoubleValue(distance),//distance for Y in m
    "GridWidth", UintegerValue(gridWidth),
    //where grid Width a number that
```

```
    // makes the [number X number] topology
    "LayoutType", StringValue("RowFirst"));
  mobility.SetMobilityModel
    ("ns3::ConstantPositionMobilityModel");
  mobility.Install(c);
  //where c is a node container
```

To set up a UDP client server the code is showing in listing 5.4.

Listing 5.4: Set-up UDP Client-Server - NS3 Code

```
//assign IP addresses
Ipv4AddressHelper ipv4;
NS_LOG_INFO("Assign IP Addresses.");
ipv4.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer i = ipv4.Assign(fdevices);
//where fdevices a NetDeviceContainer
ApplicationContainer apps;
//Create one udpServer applications on node one.
  UdpServerHelper server(srv_port);
  //where srv_port is the port of the server
  apps = server.Install(c.Get(serverNode));
  //c is a node container
  apps.Start(Seconds(105.0+time_step));
  apps.Stop(Seconds(250.0+time_step));
//Create one UdpClient application
//to send UDP datagrams from node to node
  UdpClientHelper client(i.GetAddress(serverNode),
    srv_port);
  client.SetAttribute("MaxPackets",
    UintegerValue(numPackets));
  client.SetAttribute("Interval",
    TimeValue(interPacketInterval));
  client.SetAttribute("PacketSize",
    UintegerValue(packetSize));
```

```
apps = client.Install(c.Get(clientNode));
apps.Start(Seconds(106.0+time_step));
apps.Stop(Seconds(240.0+time_step));
```

5.3 Dynamic Mobility Simulation Scenario

5.3.1 Simulation Details

It have been already checked the transmission range and the ability of routing protocols to create path in a static scenario, but in VANETs the nodes is moving constantly so the next level is to check the ability of routing protocols (AODV, OLSR, E-GPSR) to establish connection paths among random-moving nodes.

For this kind of scenarios the nodes have to be placed randomly, NS-3 provides a helper class which called ns3::MobilityHelper. Also the nodes is moving, this could be configured with the function SetMobilityModel(), using the follow parameters:

- speed: A random variable used to pick the speed of a random waypoint-model,
  set with class: RandomVariableValue,
  underlying type: RandomVariable,
  initial value: Uniform:0.3:0.7,
  flags: construct write read.

- Pause: A random variable used to pick the pause of a random waypoint-model,
  set with class: RandomVariableValue,
  underlying type: RandomVariable,
  initial value: Constant:2,
  flags: construct write read.

- PositionAllocator: The position model used to pick a destination point,
  set with class: ns3::PointerValue,
  underlying type: ns3::Ptr $< ns3 :: PositionAllocator >$,

initial value: 0,
flags: construct write read.

The topology start in a border about 1000m and is being increased with 500m until 5000m. The total number of Nodes is being calculated with this form:

$$N = density * ((border^2)/(distancebetweenNeighbours^2)$$

where density is varying from 1 to 6 and the distance between Neighbours is the distance that we consider that two nodes can transmit each other and varying from 300m to 450m. An example of the topology is showing to figure 5.5.

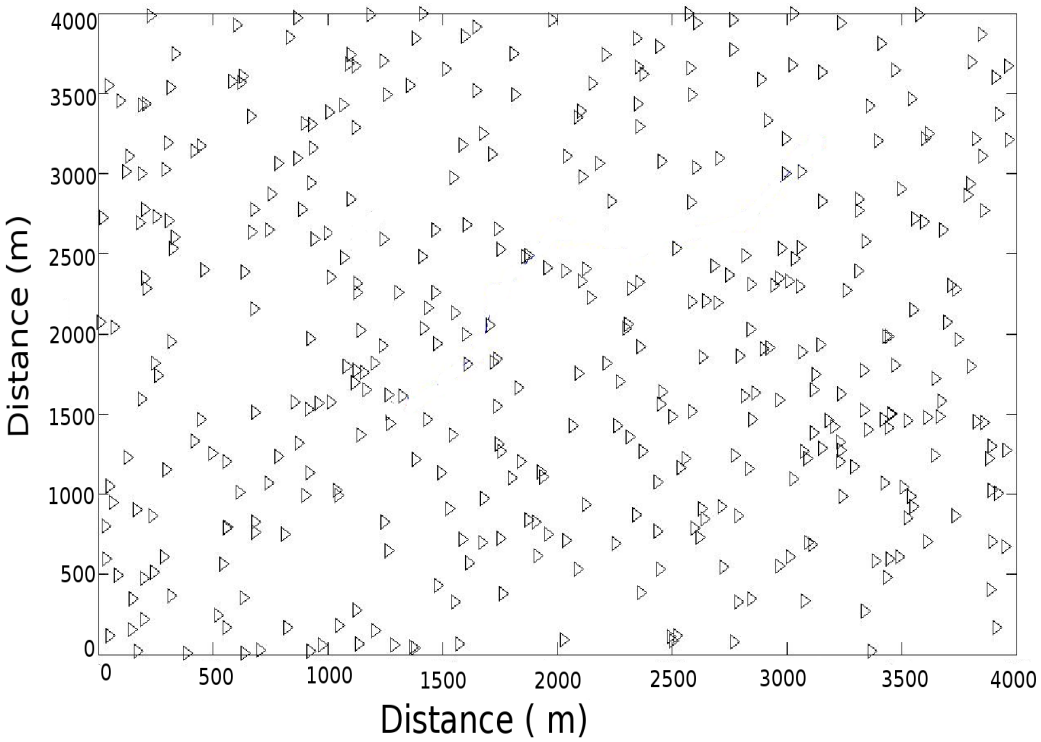

Figure 5.5: Random Mobility Topology with Density 3

5.3.2 Simulation Results

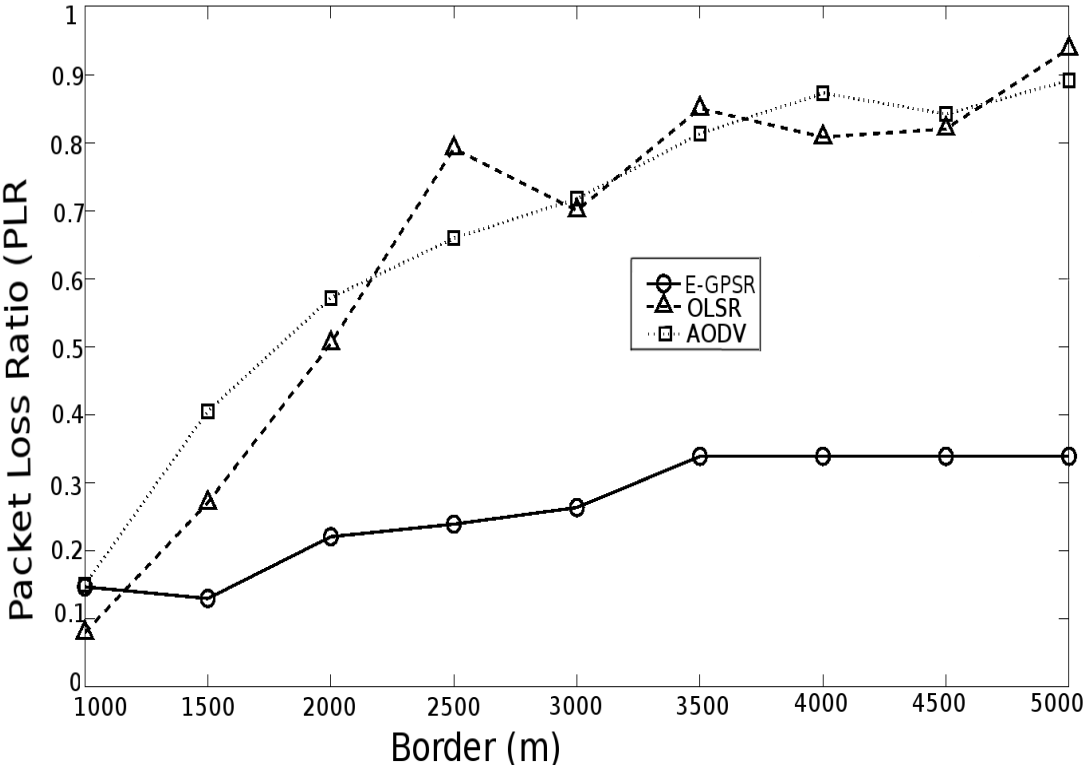

Figure 5.6: PLR into Random Topology

In the figure 5.6 it can be seen the PLR for all routing protocols, where distance refers to distance between source and destination in meters. As the distance is being increase the OLSR and AODV is not able to establish the connection unlike with the E-GPSR which has lower PLR and after 3500 meters is being stabilized. So here we can see that AODV and OLSR in long distances MANETs where VANETs is being involved they can not make routing paths, in contrast E-GPSR is shows a well promising performance for VANETs.

5.3.3 NS-3 Code

Here is the code to set up a random way point into NS-3:

Listing 5.5: Set-up Mobility Random Topology - NS-3 Code

```
//Instantiate mobility helper
MobilityHelper mobility;
```

```
//Set up an object for position
ObjectFactory pos;
//Type = Rectangle
pos.SetTypeId ("ns3::RandomRectanglePositionAllocator");
//Setup postion of X and Y
pos.Set ("X", RandomVariableValue
  (UniformVariable (0.0, border))); //where border
                        //is the size of rectangle

pos.Set ("Y", RandomVariableValue
   (UniformVariable (0.0, border))); //where border
                         //is the size of rectangle
//get the random position
Ptr<PositionAllocator> taPositionAlloc =
   pos.Create()->GetObject<PositionAllocator> ();

//Set up the mobility model
mobility.SetMobilityModel
("ns3::RandomWaypointMobilityModel",
 "Speed", RandomVariableValue(ConstantVariable (2)),
 "Pause", RandomVariableValue(ConstantVariable (5)),
 "PositionAllocator", PointerValue (taPositionAlloc));
//set the random positions
mobility.SetPositionAllocator (taPositionAlloc);
//Install
mobility.Install (c);
//where c is a node container
```

5.4 Realistic with SUMO Simulation Scenario

5.4.1 Simulation Details

A realistic VANET scenario includes nodes that is vehicles and moving with high speeds into long distances in a highway. NS-3 does not provide modules for VANET scenarios, in this way external simulation tools have been used.

The first tool is SUMO, as mentioned in chapter 3, SUMO is a simulation tool which produce a vehicle traffic mobility topology. The second tool is MOVE which produce trace files for NS-2. Finally a helper of NS-3 called ns3::Ns2MobilityHelper provides an Application Programming Interface (API) for use NS-2 mobility trace files into NS-3.

In mobility scenario there three types of vehicles:

- Car with length equal to 5m and max speed equal to 100 km/h.
- Bus with length equal to 11m and max speed equal to 80 km/h.
- Truck with length equal to 18m and max speed equal to 60 km/h.

The vehicles is moving around a rectangular which has width 2000m and height 600m. For more details about the topology and implementation at SUMO see chapter 3. The topology is used with four instances about the number of vehicles:

- 50 vehicles,
- 100 vehicles,
- 150 vehicles,
- 200 vehicles.

Into this vehicles UDP client-server applications have been set up. The time interval of UDP server is 0.04 seconds and size of send packets is 640 bytes. The vehicles which is client or server is chosen randomly with the condition that the

distance between source and destination (client and server) is more than 1000 meters (see section about transmission range) in order to have next hoping of the packet and routing protocol is being used. The number of UDP applications is various from 10 to 50 with a step by 10. Final in the simulation 105 different seeds have been used, that means 105 different sequences of random UDP client-servers has been set up.

5.4.2 Simulation Results

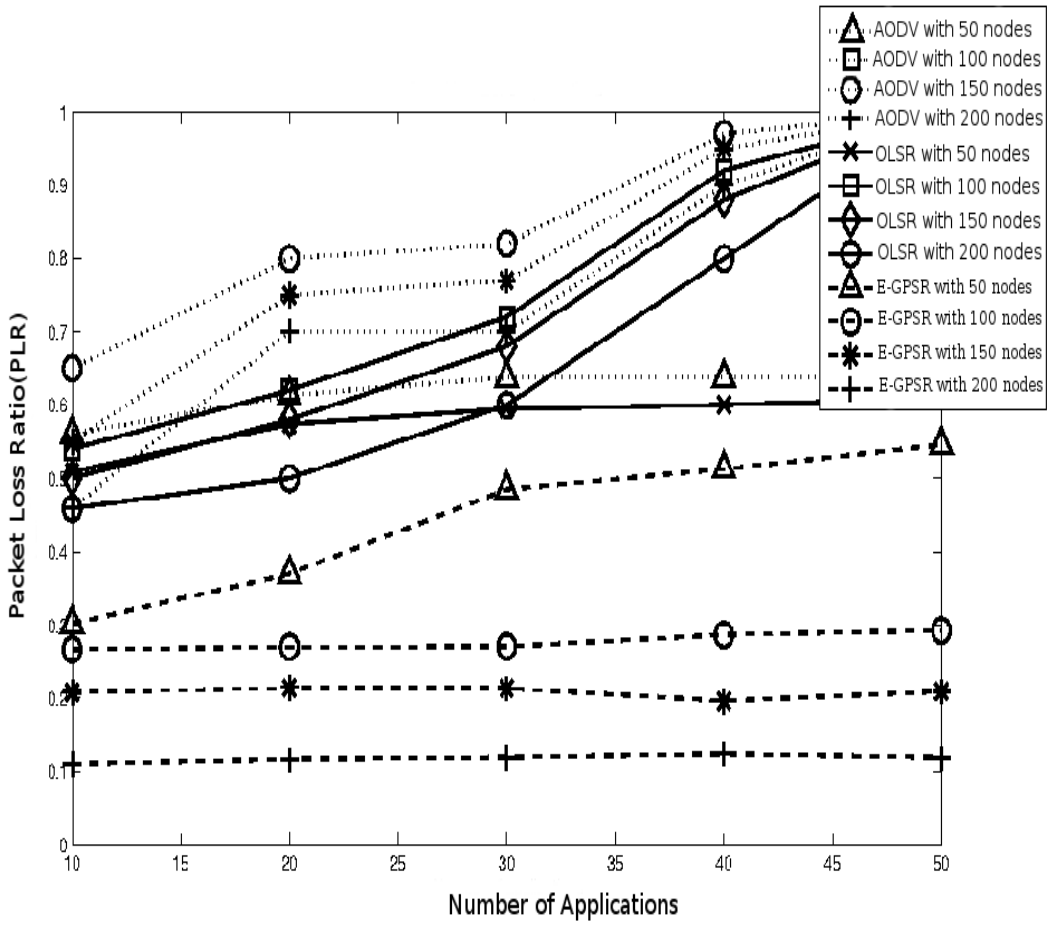

Figure 5.7: PLR into SUMO Topologies

The figure 5.7 shows the PLR in contrast the number of UDP application that have been set up. As it could be seen at this figure then the number of vehicles is low (50 nodes), the PLR is stabilized due to the small routing paths. The E-GPSR have better results than OLRS and AODV, but in compared of the other topologies of 100,150 and 200 nodes the PLR is higher. The reason is that the density of vehicle is lower, so some big gaps between vehicles appears.

OLSR and AODV starts with a medium PLR and seems is not working when a lot of applications is simultaneously transmitting. In the other side E-GPSR achieves a stable and lower PLR. So definitely, E-GPSR has better performance in all topologies (50, 100, 150 and 200 nodes) that the OLSR and AODV. Also is clear obvious that OLSR and AODV is not good enough to take over VANET topologies.

5.4.3 NS-3 Code

Here is the code for NS-3 using Ns2Mobility helper for input a scenario with moving vehicles:

Listing 5.6: Set-up Ns2Mobility Helper - NS-3 Code

```
//Enable logging from the ns2 helper
//LogComponentEnable ("Ns2MobilityHelper",LOG_LEVEL_DEBUG);

//Parse command line attribute
std::string traceFile="/FILE_PATH/nodes200.ns_movements";
std::string logFile="log1.log";
uint32_t numberOfnodes = 200 ;

// Create Ns2MobilityHelper with
//the specified trace log file as parameter
Ns2MobilityHelper ns2 = Ns2MobilityHelper (traceFile);

// open log file for output
std::ofstream os;
```

```
os.open (logFile.c_str ());

// Create all nodes.
NodeContainer c;
c.Create (numberOfnodes);

// The below set of helpers will
//help us to put together the wifi NICs we want
WifiHelper wifi;

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
// set it to zero; otherwise, gain will be added
wifiPhy.Set("RxGain", DoubleValue(-10));

YansWifiChannelHelper wifiChannel;
wifiChannel.SetPropagationDelay
  ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss
  ("ns3::FriisPropagationLossModel");
wifiPhy.SetChannel(wifiChannel.Create());
wifiPhy.SetPcapDataLinkType
  (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

// Add a non-QoS upper mac, and add a rate control
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default();
wifi.SetStandard(WIFI_PHY_STANDARD_80211p_SCH);
wifi.SetRemoteStationManager(algorithm);

NetDeviceContainer fdevices = wifi.Install
  (wifiPhy, wifiMac, c);

ns2.Install ();
```

Chapter 6

Conclusion & Future Work

This thesis has introduced an emergency field that is VANET. It has been described technologies and techniques that is take in place into VANET networks. Furthermore simulations tools such as NS-3 and SUMO have been presented which is mainly used from the researchers of this technology field. Also it is being shown in practice how to creating a complete VANET scenario in NS-3, including the setting up step by step of MAC and PHY layers, the routing protocol and the UDP client-server. In addition, it been presented a new Position-Based routing Protocol which is E-GPSR, developed for NS-3.

It has been checked the ability of AODV,OLSR and E-GPSR to create a routing path and has been measuring the maximum distance that is possible into VANET scenario using NS-3, which is 500m. It has been investigated the performance of AODV, OLSR and E-GPSR, in a simple static, dynamic and finally a realistic scenarios.

The performance analysis has been demonstrated that in simple Static Grid Topology all routing protocols (AODV, OLSR ,E-GPSR) have similar performance and could establish routing paths within an UDP client-server application.

In the Dynamic Mobility Topology, it has been shown that AODV and OLSR have very medium performance and weakness to make routing paths, in contrast E-GPSR showing a well promising performance and durability to find routing paths.

In last and finally realistic scenario includes nodes that is vehicles and moving with high speeds, the analysis gave the determinant results. In this case the E-GPSR has shown that can cope with high dynamic and frequently changing

VANET topologies, and achieves very promising results. In contrast OLSR and AODV with a very low performance shows that can not working in such environments such us VANET topologies, and amplifies the conviction that algorithms from MANET is not suitable for VANET case.

Simulation results have demonstrated that E-GPSR this new routing strategy is able to guarantee lower PLR than other routing algorithms already available in NS-3.

For future work is planing to develop more sophisticate techniques into E-GPSR mechanism and testing into NS-3 simulations. Furthermore bigger and more complex topologies is on the way in order to cover more realistic VANET scenarios. Further plan is going to investigate more Position-based Algorithms and implement into NS-3 as well as to testing into realistic VANET scenarios. Finally a comparison among Position-based protocols is being conceived.

Other future work includes the sector of Cluster-based routing protocols, which is also a very promising technique for VANET communication. In this directions is planning to developed basic Cluster-based algorithms and implemented to NS-3. Finally a comparison between Position-based protocols and Cluster-based is on procedure.

We believe that our work represents a contribution for all people that want to start working on VANETs.

Appendix A

Shell Script to automatically create entries for sumo XML-file

Listing 1: Shell Script to Create XML-Files for SUMO

```
#create the file
touch hello.rou.xml;

FILE="hello.rou.xml";

echo "<routes>" >> $FILE

echo "<vtype accel=\"1.0\" decel=\"3.0\"
id=\"Car\" length=\"5.0\" maxspeed=\"100.0\"
sigma=\"0.0\" />" >> $FILE

echo "<vtype accel=\"1.0\" decel=\"3.0\"
id=\"Bus\" length=\"11.0\" maxspeed=\"80.0\"
sigma=\"0.0\" />" >> $FILE

echo "<vtype accel=\"1.0\" decel=\"3.0\"
id=\"Truck\" length=\"18.0\" maxspeed=\"60.0\"
sigma=\"0.0\" />" >> $FILE

#insert newline to file
echo "" >> $FILE

echo "<route id=\"route1\" edges=\"start1 lr up rl down
lr up rl down \"/>" >> $FILE

echo "<route id=\"route2\" edges=\"start2 rl down lr up
rl down lr up \"/>" >> $FILE
```

```
echo "" >> $FILE


START=0;
NB_NODES=200;
ID_CAR_A=1;
ID_CAR_B=2;
DEPART_TIME_A=1;



until [ $START -gt $NB_NODES ]; do


        for j in `seq 1 2`;
        do
                #insert a function for choosing one
                #of vehicle types randomly
                NUMBER=$[ ( $RANDOM % 3 ) + 1 ]

                case $NUMBER in
                1)
                        TYPE="Car"
                        ;;
                2)
                        TYPE="Bus"
                        ;;
                3)
                        TYPE="Truck"
                        ;;
                esac
```

```
                    echo " <vehicle depart=\"${DEPART_TIME_A}\
                    "id=\"veh${ID_CAR_A}\" route=\"route${j}\
                    "type=\"${TYPE}\" />"      >> $FILE
                    let ID_CAR_A=$ID_CAR_A+1
                    let DEPART_TIME_A=$DEPART_TIME_A+2
          done

 let START=$START+1
done



echo "</routes>" >> $FILE
```

Appendix B

Instructions to use MOVE

Once you have run the command

Listing 2: Start MOVE

```
java -jar MOVE.jar
```

You will see the GUI of Fig 1.

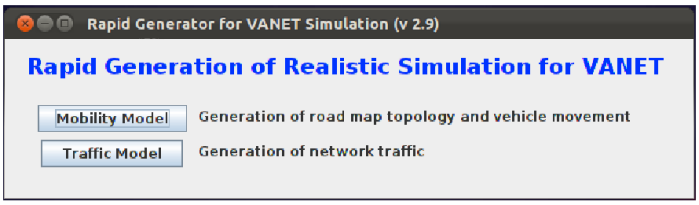

Figure 1: MOVE-GUI Fig. 1

Select the Mobility Model button to open the Mobility Model generator window as shown to Fig 2.

Then choose the Configuration bottom of Simulation part of the window of Fig 2.

As you do that , the window of Fig 3 will be appeared . Fill the fields (or press the three dots to appear the file manager) Map file with a .net.xml file and Routes file with a .rou.xml file . Fill the fields Begin End which is the starting and ending time of simulation check the box SetOutPut(Trace File) and fill (or use file manager) to set a output file . When you finish this and check that everything is alright At the menu on the top press:

```
File -> Save as -> output.sumo.out.tr
```

This file will be used later. So , now close this window and go to the window of Fig 2, when select the bottom Run Simulation of Simulation part of the window the window of Fig 4 will be appeared . In the field Sumo Configuration File put the file that you have just saved previously and press the bottom OK this will be take some time . When you see Simulation ended at time : "the time that you have put previously" this mean that the simulation has run successfully so close the window and go to the window of Fig 1 . Select Traffic Model bottom and the window of Fig 5 will appear . Choose Dynamic Mobility bottom and you will see the window of Fig 6. From the menu on the top press sequentially:

```
File->Import MOVe trace ->.rou.xml.sumo.tr (file)
```

After .net.xml (file) wait some time to load the files. When you noticed that a table of nodes has appear press sequentially

```
File->Save AS -> my-scenario.tcl
```

If you reach that point, you have successfully created the tcl file for NS-2 But the process doesnt stop here.

You have to delete some part of tcl file to be readable from NS-3 Delete the code that shown to the listing 1

Listing 3: Delete Code form tcl-File

```
---------- at the begenning of tcl file ----------


==========
# ============================================
# 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)      356       ;# number of mobilenodes
set val(rp)      AODV ;# routing protocol

set opt(x)       4252 ;# x coordinate of topology
set opt(y)       652      ;# y coordinate of topology
set stopTime       999.00

# ======================================================
# Main Program
# ======================================================
#
# Initialize Global Variables
#
set ns_ [new Simulator]
set tracefd [open  w]
$ns_ trace-all $tracefd

# set up topography object
set topo     [new Topography]
$topo load_flatgrid $opt(x) $opt(y)

#
```

```
# Create God
#
create-god $val(nn)

# ============================================================
#         TraCI Connection Setup
# ============================================================
set mobilityInterfaceClient [new TraCIClient]
$mobilityInterfaceClient set-remoteHost localhost
$mobilityInterfaceClient set-remotePort 8888
$mobilityInterfaceClient set-timeInterval 1.0
puts "Connect to TraCI server"
$mobilityInterfaceClient connect
$mobilityInterfaceClient startSimStepHandler

# Configure node

set chan_1_ [new $val(chan)]
$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 OFF \
                  -routerTrace OFF \
                  -macTrace OFF \
                  -movementTrace OFF \
                  -channel $chan_1_
```

```
for {set i 0} {$i < $val(nn)} {incr i} {
  set node_($i) [$ns_ node]
  $node_($i) random-motion 0 ;# disable random motion
  $mobilityInterfaceClient add-node $node_($i)
}
#


--------- at the end of the tcl file ----------

# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $stopTime "$node_($i) reset";
}
$ns_ at $stopTime "$mobilityInterfaceClient close"
$ns_ at $stopTime "stop"
$ns_ at $stopTime "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
    global ns_ tracefd
    $ns_ flush-trace
    close $tracefd
}

puts "Starting Simulation..."
$ns_ run
```

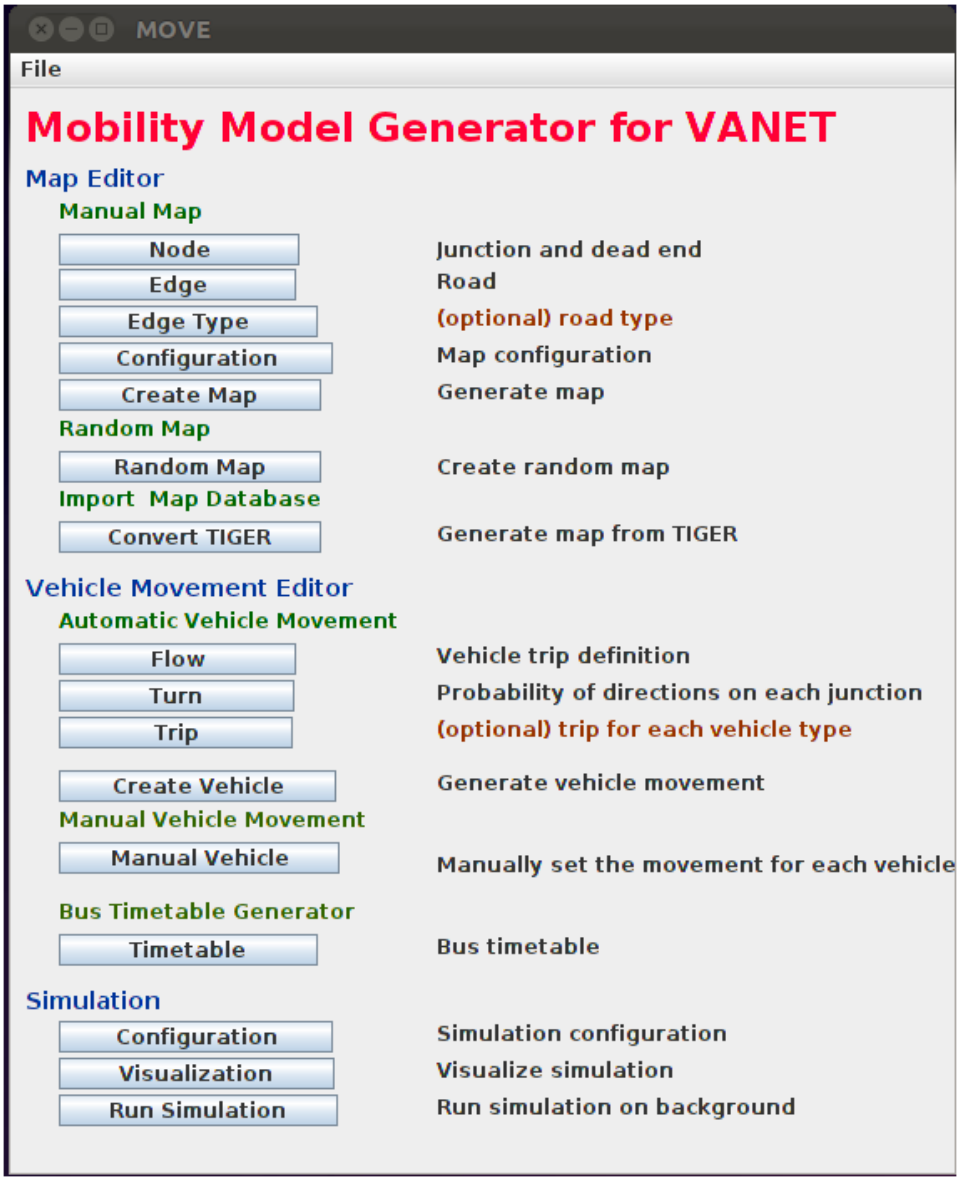

Figure 2: MOVE-GUI Fig. 2

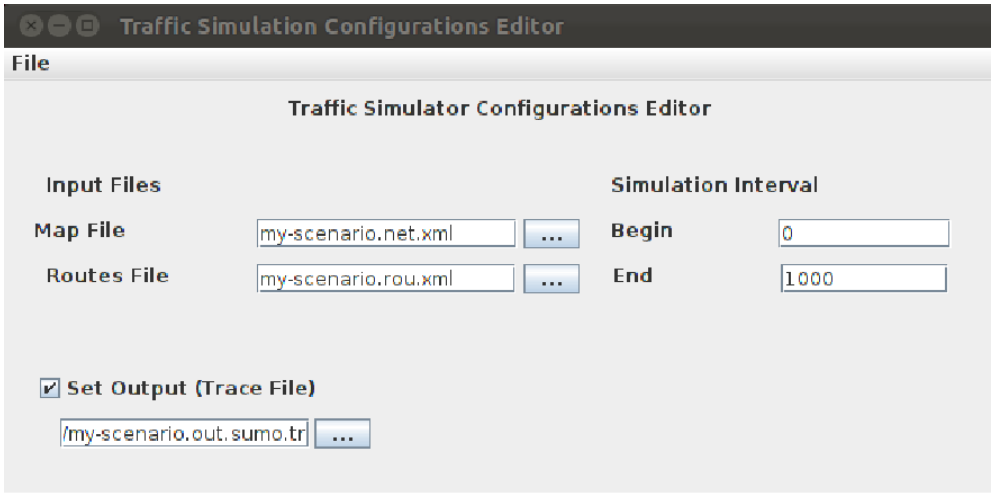

Figure 3: MOVE-GUI Fig. 3

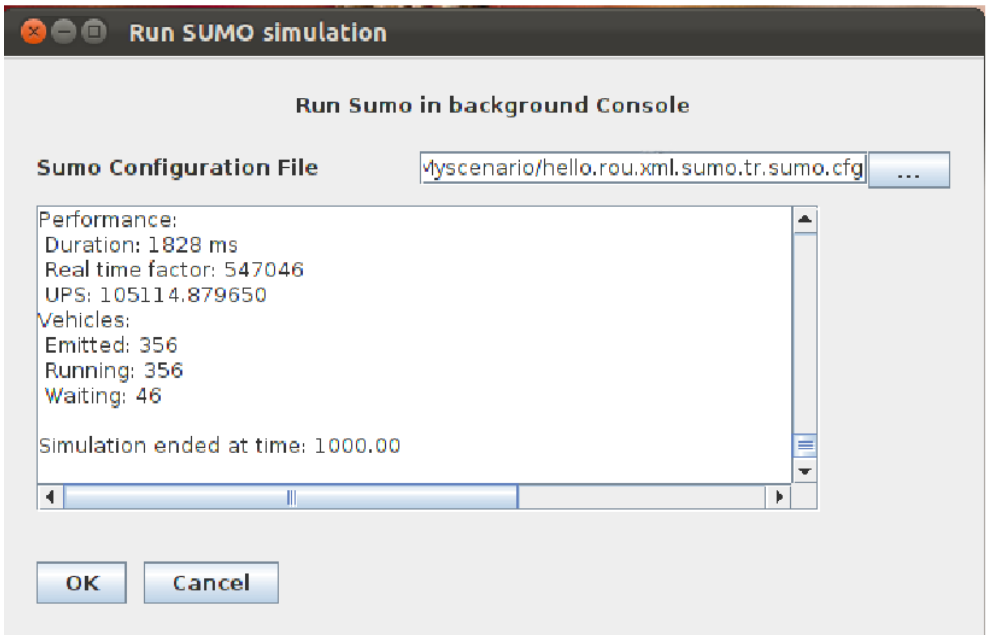

Figure 4: MOVE-GUI Fig. 4

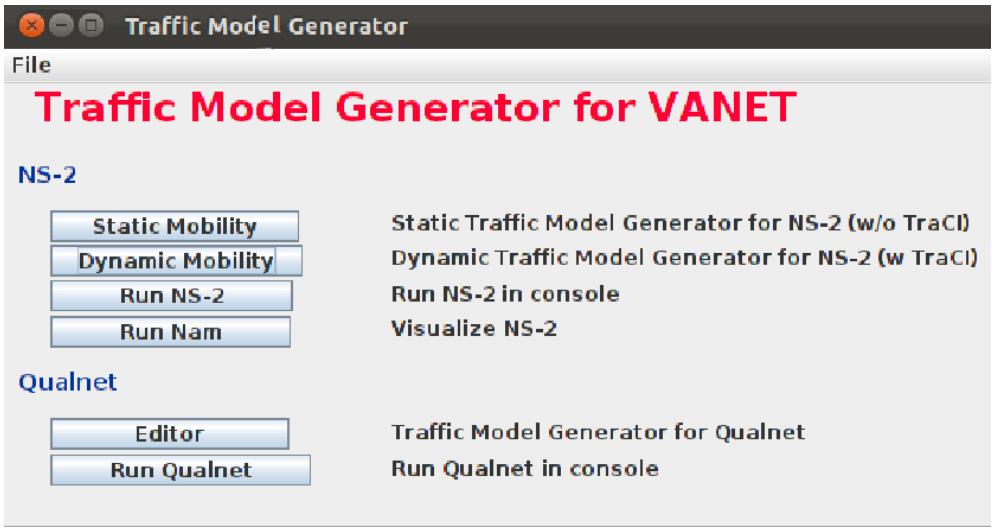

Figure 5: MOVE-GUI Fig. 5

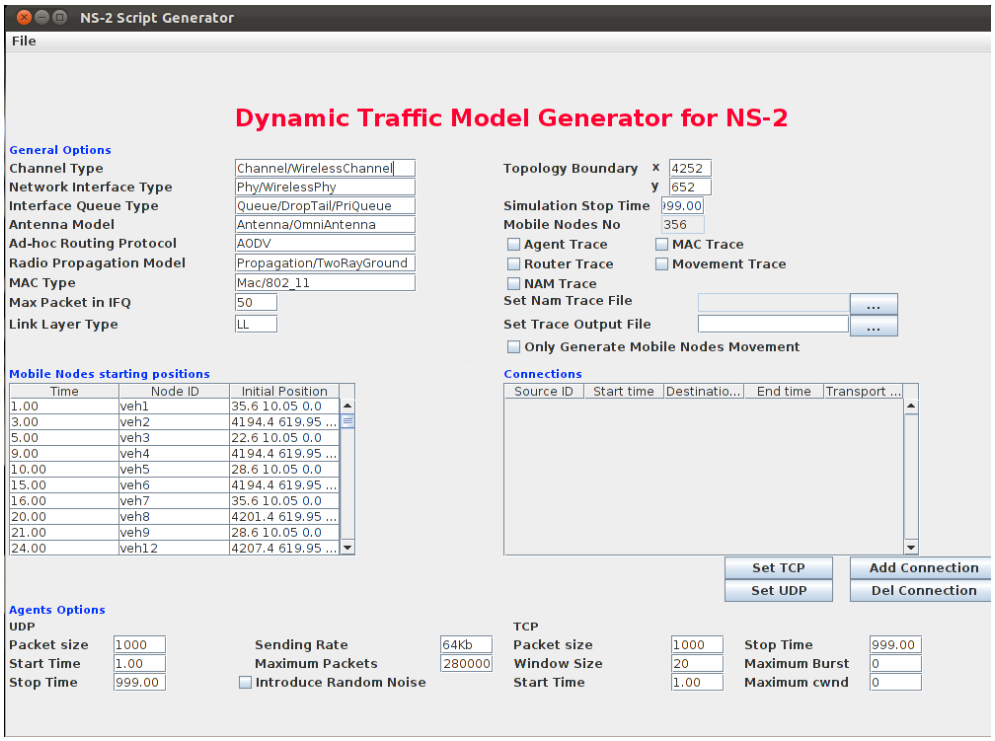

Figure 6: MOVE-GUI Fig. 6

Appendix C

The following list of packages should be accurate for Ubuntu 9.10 release; other releases or other Debian-based systems may slightly vary. minimal requirements for C++ (release): This is the minimal set of packages needed to run ns-3 from a released tarball.

```
sudo apt-get install gcc g++ python
```

minimal requirements for Python (release): This is the minimal set of packages needed to work with Python bindings from a released tarball.

```
sudo apt-get install gcc g++ python python-dev
```

Mercurial is needed to work with ns-3 development repositories.

```
sudo apt-get install mercurial
```

Running python bindings from the ns-3 development tree (ns-3-dev) requires bazaar

```
sudo apt-get install bzr
```

Debugging:

```
sudo apt-get install gdb valgrind
```

GNU Scientific Library (GSL) support for more accurate WiFi error models

```
sudo apt-get install gsl-bin libgsl0-dev libgsl0ldbl
```

The Network Simulation Cradle (nsc) requires the flex lexical analyzer and bison parser generator:

```
sudo apt-get install flex bison
```

To install gcc-3.4 for some Network Simulation Cradle (nsc) stacks:

```
sudo apt-get install g++-3.4 gcc-3.4
```

To read pcap packet traces

```
sudo apt-get install tcpdump
```

Database support for statistics framework

```
sudo apt-get install sqlite sqlite3 libsqlite3-dev
```

Xml-based version of the config store (requires libxml2 ¿= version 2.7)

```
sudo apt-get install libxml2 libxml2-dev
```

A GTK-based configuration system

```
sudo apt-get install libgtk2.0-0 libgtk2.0-dev
```

To experiment with virtual machines and ns-3

```
sudo apt-get install vtun lxc
```

Support for utils/check-style.py code style check program

```
sudo apt-get install uncrustify
```

Doxygen and related inline documentation:

```
sudo apt-get install doxygen graphviz imagemagick
sudo apt-get install texlive texlive-pdf
 texlive-latex-extra texlive-generic-extra
 texlive-generic-recommended
```

The ns-3 manual and tutorial are written in reStructuredText for Sphinx (doc/tutorial, doc/manual, doc/models), and figures typically in dia:

```
sudo apt-get install python-sphinx dia texlive
  texlive-pdf texlive-latex-extra
  texlive-extra-utils texlive-generic-recommended
```

Support for Gustavo Carneiro's ns-3-pyviz visualizer

```
sudo apt-get install python-pygraphviz python-kiwi
 python-pygoocanvas libgoocanvas-dev
```

Support for openflow module (requires some boost libraries)

```
sudo apt-get install libboost-signals-dev
  libboost-filesystem-dev
```

Glossary

ACK - ACKnowledgement
AIFS - Arbitration Inter-Frame Spaces
AODV - Ad-Hoc On Demand Distance Vector Routing Protocol
CCH - Control CHannel
CSMA/CD - Carrier Sense Multiple Access with Collision Detection
CW - Contention Window
DLL - Data Link Layer
DCF - Distributed coordination function
DSDV - Destination-Sequenced Distance-Vector
DSR - Dynamic Source Routing
EGPSR - Extended-GPSR
EDCA - Enhanced Distributed Channel Access
GPS - Global Positioning System
GPSR - Greedy Perimeter Stateless Routing
LLC - Logical Link Control
MANET - Mobility Ad-hoc NETwork
MOVE - MObility model generator for VEhicular networks
NAV - Network Allocation Vector
NS-2 - Network Simulator 2
NS-3 - Network Simulator 3
OLSR - Optimized Link State Routing Protocol
OSI - Open Systems Interconnection
PHY - PHYsical layer
PBRP - Position Based Routing Protocol
PLR - Packet Loss Ratio
PLCP - Physical Layer Convergence Protocol
PMD - Physical Medium Dependent
RTS/CTS - Request To Send frame/Clear To Send
RREQ - Route Request

SAP- Service Access Point
SAP - Subnetwork Access Protocol
SCH - Service CHannel
SUMO - Simulation of Urban MObility
TBRP - Topology Based Routing Protocol
TCP - Transmission Control Protocol
UDP - User Datagram Protocol
V2I - Vehicle to Infrastructure communication
V2V - Vehicle to Vehicle communication
VANET - Vehicular Ad-hoc NETwork
VII - Vehicle Infrastructure Integration
WAVE - Wireless Access in Vehicular Environments
WSMP - WAVE Short Messages Protocol

References

[1] IEEE 1609. *IEEE 1609 Family of Standards for for Wireless Access in Vehicular Environments (WAVE), available from IEEE standards.*

[2] IEEE 802.11. *Information Technology - Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless (LAN) Medium Access Control (MAC) and Physical Layer (PHY) Specifications.* ANSI/IEEE Std. 802.11, ISO/IEC 8802-11, 1999.

[3] IEEE 802.11p. *IEEE Draft Standard for Information Technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 6: Wireless Access in Vehicular Environments.* IEEE Std 802.11p, 2010.

[4] M. Abdoos, K. Faez, and M. Sabaei. Position based routing protocol with more reliability in mobile ad hoc network. In *Internet, 2009. AH-ICI 2009. First Asian Himalayas International Conference on*, pages 1 –4, Nov. 2009.

[5] Yuh-Shyan Chen Chung-Ming Huang. *Telematics Communication Technologies and Vehicular Networks: Wireless Architectures and Applications.* Aug. 2010.

[6] T. Clausen and P. Jacquet. *Optimized Link State Routing Protocol (OLSR)*, 2003.

[7] cygwin. Cygwin , linux feeling - on windows. available at `{http://www.cygwin.com/}`, 2011.

[8] D. Maltz D. Johnson, Y. Hu. *The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4*, 2007.

[9] Josh Broch David B. Johnson, David A. Maltz. Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks. 2001.

[10] S. Deering and R. Hinden. *RFC 2460 Internet Protocol, Version 6 (IPv6) Specification*. Internet Engineering Task Force, Dec. 1998.

[11] ETSI. *Intelligent transport systems (ITS), radiocommunications equipment operating in the 5 855 MHz to 5 925 MHz frequency band, harmonized EN covering essential requirements of article 3.2 of the RTTE directive.*, Dec. 2007.

[12] V. Govindaswamy, W.L. Blackstone, and G. Balasekara. Survey of recent position based routing mobile ad-hoc network protocols. In *Computer Modelling and Simulation (UKSim), 2011 UkSim 13th International Conference on*, pages 467 –471, Apr. 2011.

[13] Kenneth P Laberteaux Hannes Hartenstein. *VANET:Vehicular Applications and Inter-Networking Technologies*. 2010.

[14] IEEE 802.2. *Logical Link Control*. ANSI/IEEE Std 802.2-1985, 1984.

[15] ITS. *Commission Decision of 5 August 2008 on the harmonised use of radio spectrum in the 5875 - 5905 MHz frequency band for safety-related applications of Intelligent Transport Systems*, 2008.

[16] B Govinda Laxmi Ramesh Babu B Jagadeesh Kakarla, S Siva Sathya. A survey on routing protocols and its issues in vanet. *International Journal of Computer Applications (0975 8887)*, 28(4):38 –44, Aug. 2011.

[17] Ding Junxia. Simulation and evaluation of the performance of fsr routing protocols based on group mobility model in mobile ad hoc. In *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*, pages 1 –4, Dec. 2010.

[18] Fan Li and Yu Wang. Routing in vehicular ad hoc networks: A survey. *Vehicular Technology Magazine, IEEE*, 2(2):12–22, Jun. 2007.

[19] Mohammad Moustafa Qabajeh Liana Khamis Qabajeh, Laiha Mat Kiah. A qualitative comparison of position-based routing protocols for ad-hoc networks. *IJCSNS Interantional Journal of Computer Science and Network Security*, 9(2):131 – 140, Feb. 2009.

[20] Qiang Liu, Hua Wang, Jingming Kuang, Zheng Wang, and Zhiming Bi. Wsnp1-1: M-tora: a tora-based multi-path routing algorithm for mobile ad hoc networks. In *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pages 1 –5, Dec. 2006.

[21] Tom H. Luan, Xinhua Ling, and Xuemin Shen. Mac performance analysis for vehicle-to-infrastructure communication. In *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*, pages 1 –6, Apr. 2010.

[22] Wu Ming, Yang Lin-tao, Li Cheng-yi, and Jiang Hao. Capacity, collision and interference of vanet with ieee 802.11 mac. In *Intelligent Networks and Intelligent Systems, 2008. ICINIS '08. First International Conference on*, Nov. 2008.

[23] MOVE. Mobility model generator for vehicular networks. available at `http://sourceforge.net/apps/mediawiki/move`, 2011.

[24] Hemanth Narra, Yufei Cheng, Egemen K. Çetinkaya, Justin P. Rohrer, and James P.G. Sterbenz. Destination-sequenced distance vector (DSDV) routing protocol implementation in ns-3. pages 439–446, March 2011.

[25] NS-3. Network simulator. available at `http://www.nsnam.org/`, 2012.

[26] NS2. Network simulator - ns-2. available at `{http://nsnam.isi.edu/nsnam/index.php/Main_Page}`, 2011.

[27] Yanlin Peng, Z. Abichar, and J.M. Chang. Roadside-aided routing (rar) in vehicular networks. In *Communications, 2006. ICC '06. IEEE International Conference on*, volume 8, Jun. 2006.

[28] C. Perkins, E. Belding-Royer, and S. Das. *Ad hoc On-Demand Distance Vector (AODV) Routing*, 2003.

[29] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, 24(4):234–244, Oct. 1994.

[30] J. Postel and J. K. Reynolds. *RFC 1042: Standard for the transmission of IP datagrams over IEEE 802 networks*, Feb. 1988.

[31] M. Rajput, P. Khatri, A. Shastri, and K. Solanki. Comparison of ad-hoc reactive routing protocols using opnet modeler. In *Computer Information Systems and Industrial Management Applications (CISIM), 2010 International Conference on*, pages 530 –534, Oct. 2010.

[32] Sanjoy Das Ram Shringar Raw. Perfomance comparison of position-based routing protocols in vehicle-to-vehicle (v2v) communication. *International Journal of Engineering Science and Technology (IJEST)*, 3(1):435 – 443, Jan. 2011.

[33] David Hutchison ResiliNets, James P.G. Sterbenz. Resilinets research group. available at `http://www.ittc.ku.edu/resilinets`, 2011.

[34] H. Somnuk and M. Lerwatechakul. Multi-hop aodv-2t. In *Intelligent Ubiquitous Computing and Education, 2009 International Symposium on*, pages 214 –217, may 2009.

[35] Tao Song, Weiwei Xia, Tiecheng Song, and Lianfeng Shen. A cluster-based directional routing protocol in vanet. In *Communication Technology (ICCT), 2010 12th IEEE International Conference on*, pages 1172 –1175, Nov. 2010.

[36] M.N. SreeRangaRaju and J. Mungara. A unified approach to enhance the performance of zrp for manets on an urban terrain. In *Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on*, volume 1, pages 532 –536, Dec. 2010.

[37] SUMO. Simulation of urban mobility. available at `http://sumo.sourceforge.net/`, 2011.

[38] Daxin Tian, Yunpeng Wang, Guangquan Lu, and Guizhen Yu. A vanets routing algorithm based on euclidean distance clustering. In *Future Computer and Communication (ICFCC), 2010 2nd International Conference on*, volume 1, pages V1–183 –V1–187, May 2010.

[39] O. Tonguz, N. Wisitpongphan, F. Bai, P. Mudalige, and V. Sadekar. Broadcasting in vanet. In *2007 Mobile Networking for Vehicular Environments*, pages 7 –12, May 2007.

[40] Y. Toor, P. Muhlethaler, and A. Laouiti. Vehicle ad hoc networks: applications and related technical issues. *IEEE Comm. Surveys & Tutorials*, 10(3):74 –88, Sep. 2008.