PERFORMANCE IMPROVEMENT OF ROUTING PROTOCOL FOR VEHICULAR MOBILE AD HOC NETWORKS USING NS3

Harish Chandra Maurya[1], Dr. Pushpneel Verma[2]

[1] *Research Scholar, Department of CSE, Bhagwant University, Ajmer, Rajasthan*
[2] *Assistant Prof., Department of CSE, Bhagwant University, Ajmer, Rajasthan*

ABSTRACT

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. Because of the complexity and cost of practical evaluation of VANETs, researchers often rely on network simulations to evaluate their work. In this paper, we have developed a Network Simulator 3 (NS-3) based framework for VANET that analyzes network performance based on key performance indicators such as throughput, packet loss ratio, overhead, end-to-end delay, jitter, provides. Although the major challenge in VANET is to devise a more suitable routing protocol to efficiently route packets to their final destination, despite the high speed, frequent disconnections and highly variable topology of vehicles. This paper presents and analyzes the effect of vehicle density on the performance of the most well-known routing protocols. Quantitative metrics such as overhead, packet delivery ratio, average throughput and average end-to-end delay are evaluated using the network simulator NS-3 and SUMO. The comparison of a VANET routing protocol with the available routing pro- tools 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.

Keyword: - *AODV, OLSR, DSDV, DSR, NS-3 and VANET simulations etc.*

1.1 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 way- points. No mobility models speci_c 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 pro- tools 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.

1.2 AODV ROUTING PROTOCOL

1.2.1 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.

1.2.2 AODV Routing Overview

The source code for the AODV model it can be found in the directory "sr-c/aodv" and the class hierarchy is showing into the figure 1.1.

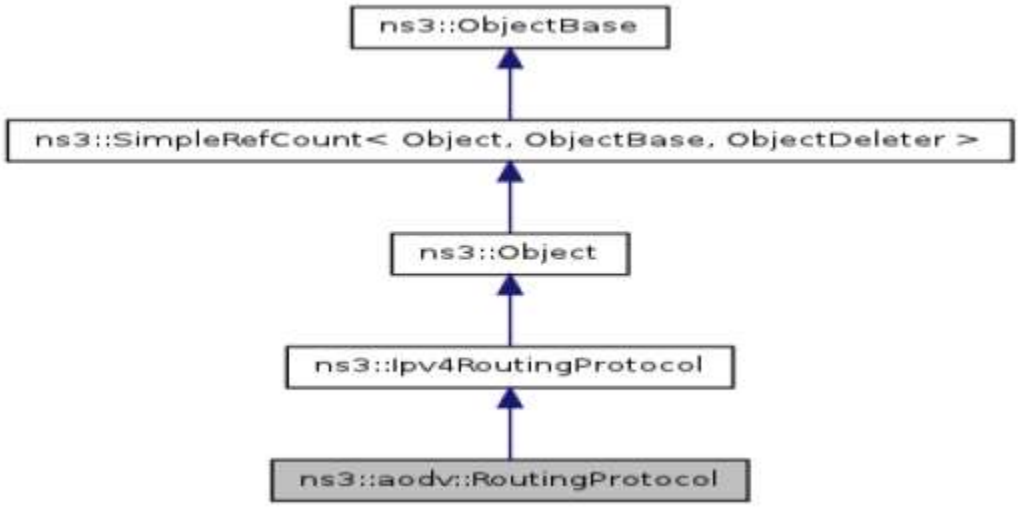

Figure 1.1: AODV Hierarchy Diagram

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:: Route Output, is used for locally originated packets and the second one, ns3::aodv:: Route Input, 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::Routing Protocol. 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::Ipv4Routing Protocol::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). There- fore, one may experience bad performance when running over wireless. There are several reasons for this: 1) HELLO messages are broadcasted. In 802.11, broad- casting 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 trans- mission 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 Adhoc Wifi Mac 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.

1.2.3 AODV Helper

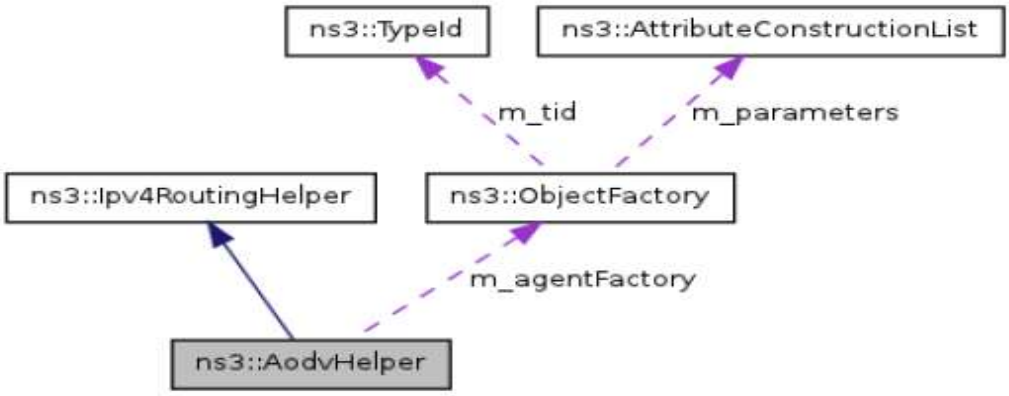

Figure 1.2: AODV Helper Diagram

A helper class for AODV has been written (figure 1.2). 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 (table 1.1).

Table 1.1: 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
```

1.3 OLSR ROUTING PROTOCOL

1.3.1 OLSR Routing

NS-3 has an already implemented and available model for OLSR. This model implements the base speci_cation 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).

1.3.2 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 1.3. 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 in-sertion/removal ns3::RoutingProtocol::NotifyAddAddress and ns3:: RoutingPro-tocol::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.

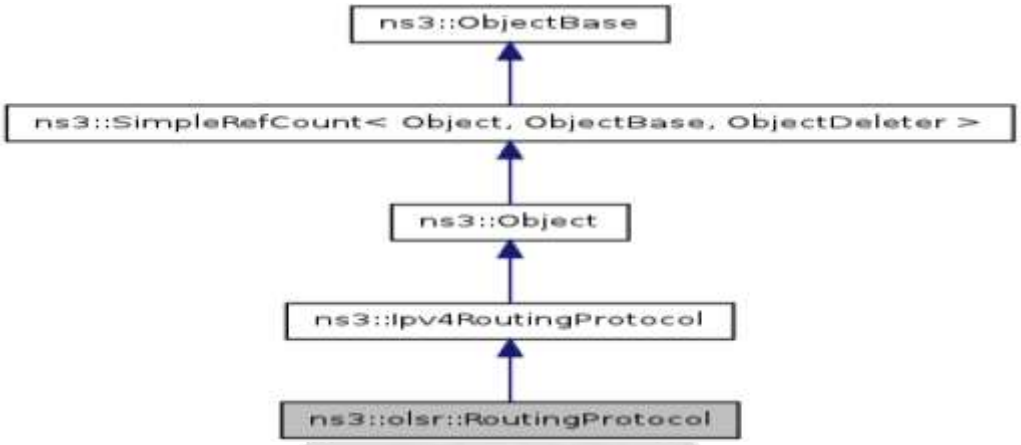

Figure 1.3: OLSR Hierarchy Diagram

1.3.3 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).

Table 1.2: 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
```

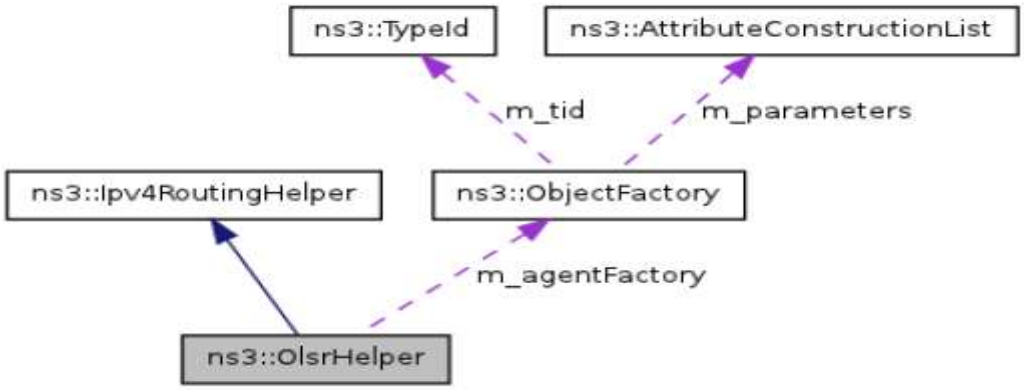

Figure 1.4: OLSR Helper Diagram

1.4 DSDV

1.4.1 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].

1.4.2 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.

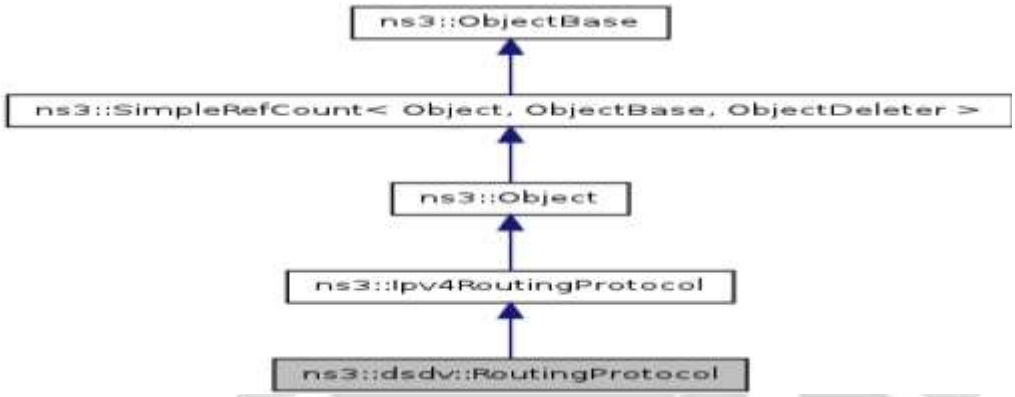

Figure 1.5: DSDV Hierarchy Diagram

The DSDV update message consists of three _elds, 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 uctuations, 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.

1.4.3 DSDV Helper

A helper class for DSDV has been written (figure 1.6). 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 (table 1.3). table 4.3: NS3 DSDV Install

Table 1.3: 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);
```

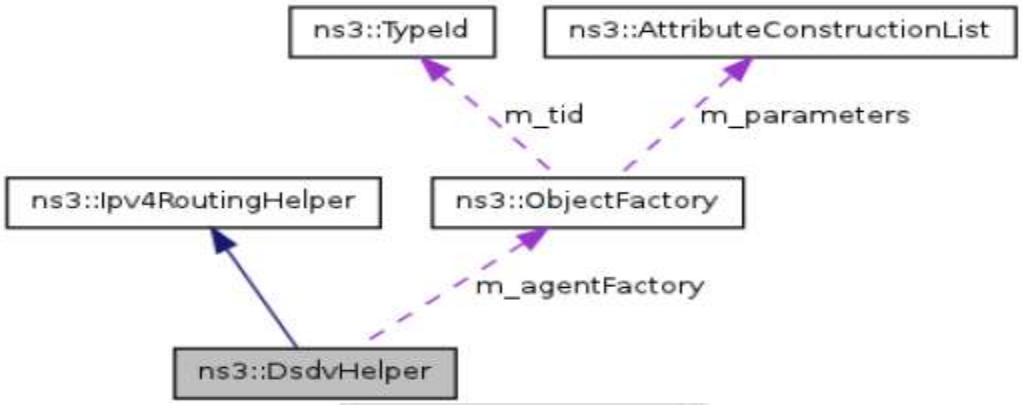

Figure 1.6: DSDV Helper Collaboration Diagram

1.5 DSR

1.5.1 DSR Routing

DSR protocol is a reactive routing protocol designed especially 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].

1.5.2 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 denes the xed-size header and identify the up-layer protocol. Class dsr::DsrOption

Header takes care of different DSR options and process different header according to speci_cation 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_le.

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 AdhocWifi Mac 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) ags.

Handling unknown DSR options.

Two types of error headers:

1. ow 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.

1.5.3 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 t 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 ClickIn- ternetStackHelper class in your simulation script (table 1.4).

Table 1.4: 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 "sr-c/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.

1.6 RESULT ANALYSIS AND DISCUSSION

1.6.1 E-GPSR: 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 destination, 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 1.7, 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,

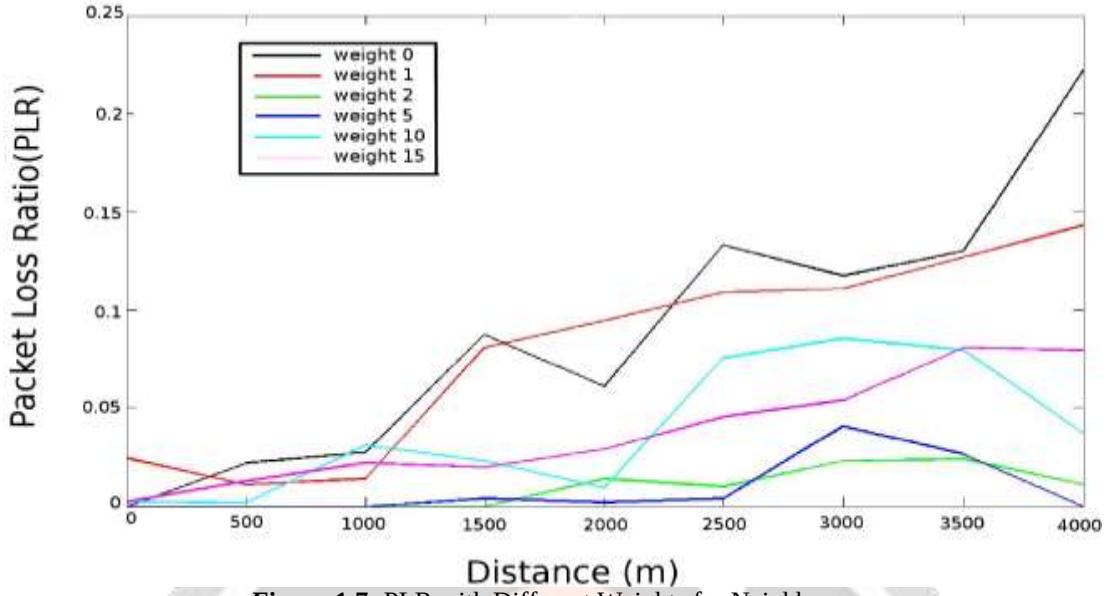

Figure 1.7: PLR with Different Weights for Neighbours

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 1.5, in which also showing two path (with red and blue) of UDP client server application coloured by red.

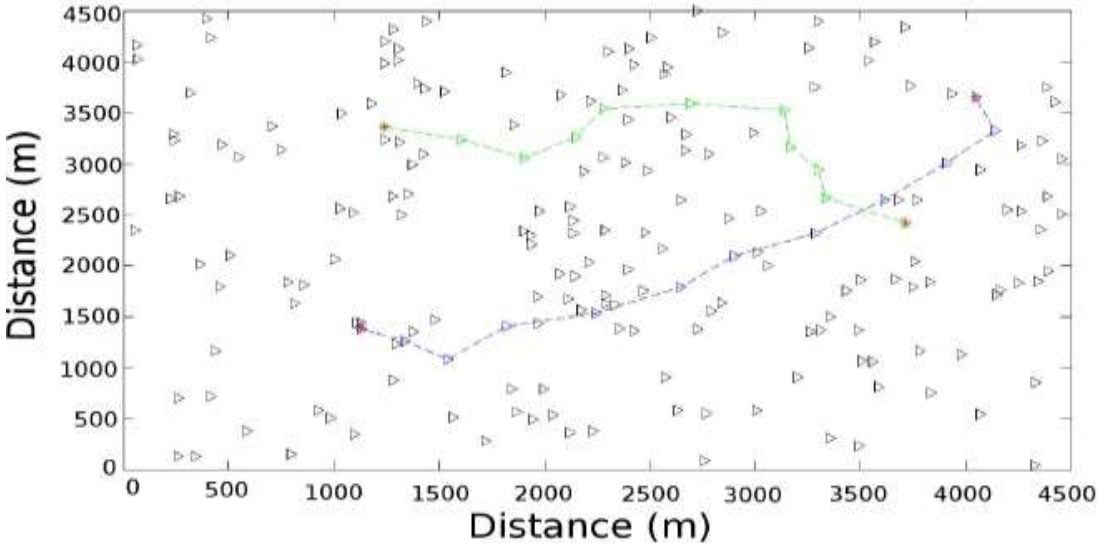

Figure 1.8: Topology in Simulation about Different Weights

1.6.2 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 1.9.
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 SetNetDe-vice() function to put into vector of a pointer of just one device.

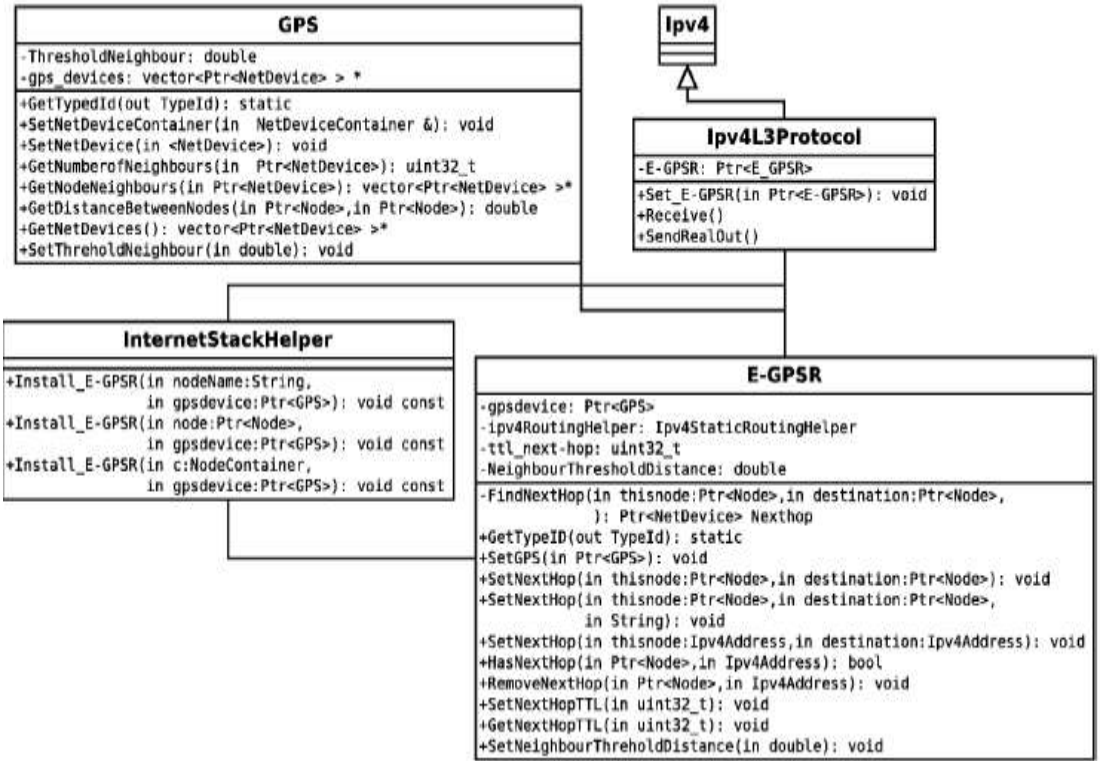

Figure 1.9: Position-Based UML Diagram

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 GetNumberOfNeigh- bours() 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 modi_ed 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 1.10 is showing how the algorithm finds a path by next-hoping the packet, with green is noticed the neighbours.

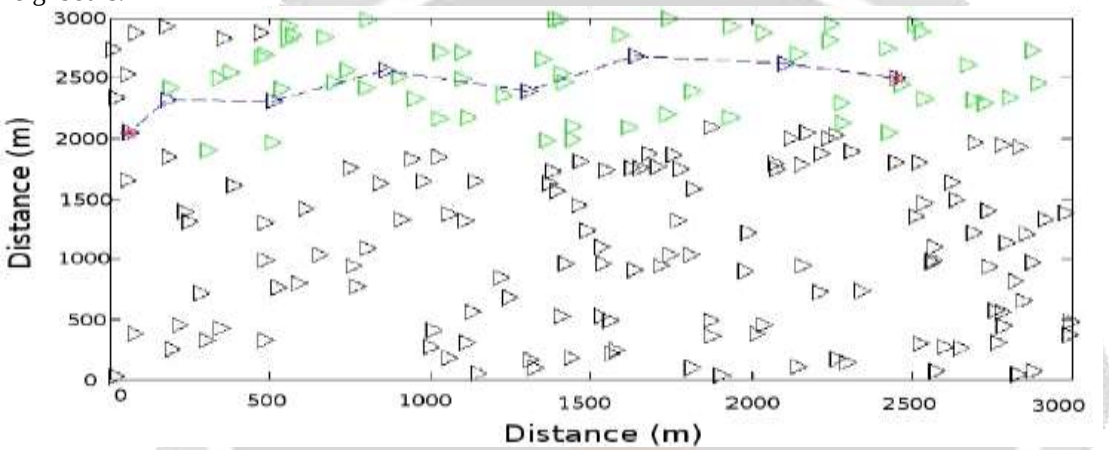

Figure 1.10: Routing Path

1.6.3 Configuration

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

Neighbour Distance, which is the maximum distance where two neighbor 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 Table 1.5 is showing how to install E-GPSR into a NS-3 scenario.

Table 1.5: 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
```

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) Speci_cations. 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 - Speci_c 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 14, 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 16, 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. C_ etinkaya, 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.