Project report
Multipath DSR protocol for ad hoc network
by

Alexandre Boursier boursier@kom.aau.dk
Stéphane Dahlen sdahlen@kom.aau.dk
Julien Marie-Françoise mariefra@kom.aau.dk
Thior Santander Marin thior@kom.aau.dk
Shekar Nethi snet03@kom.aau.dk

MSc Mobile Communications
Aalborg University
Center for PersonKommuniKation

Supervisors:
Tatiana K. Madsen tatiana@kom.aau.dk
Yaoda Liu yl@kom.aau.dk
Hans Peter Schwefel hps@kom.aau.dk

Submission date: 17th December 2004
ABSTRACT

Ad hoc network is a network without centralised administration in which different users can communicate and exchange information. In such a structure, all the nodes participate in order to achieve the network and ensure the travel of the information. Hence, multihopping techniques are used to achieve this task.

The communication reliability within an ad hoc network and how the different nodes act are managed by routing protocols. Nowadays, different types of protocols exist. Nevertheless, the source routing ones, based on information known at the source of the communication, seem to attract more studies. Source routing protocols had shown interesting results in realistic scenarios in areas such as military battlefields or airport stations. Our study deals with one of those protocols: DSR.

This project is focused on multipath with routing protocol. Indeed, it is necessary to understand that multipath techniques enhance reliability and can ensure security. We decided to go further and therefore we designed a new multipath algorithm.

The solution had been implemented and evaluated with the network Simulator 2. Since we want to know how our protocol reacts in different mobility cases, two mobility patterns had been used. The first one is the random waypoint model which allows us to present relevant results, due to the fact that all situations are taken into account. The second one using group mobility of nodes is more realistic because its applications could be, for example, an ad hoc network on a battlefield. Simulation results show that the created protocol behaves better than DSR and AODV, the two main actual reactive protocols. Our protocol performs well in high mobility by using much less overhead than the two others mentioned before. Additionally, it is interesting to see that DSR without any modifications manage poorly in high mobility situation.
## Contents

1 Introduction .......................................................... 8  
   1.1 Ad Hoc Network .................................................. 8  
   1.2 Multihopping .................................................... 9  
   1.3 Challenges of ad hoc networks ................................ 9  
   1.4 Applications ................................................... 10  
   1.5 Goal of the study ............................................... 10  
   1.6 Outline of the report .......................................... 11  

2 General Aspect: Routing in an ad hoc network .......... 12  
   2.1 The OSI layer ................................................... 12  
   2.2 Data encapsulation and layer distribution ............... 13  
   2.3 Routed and routing protocols ................................. 14  
   2.4 Problems encountered while routing in an ad hoc network .............................................. 14  
   2.5 Existing routing protocols ................................... 15  
   2.6 Other ad hoc networks protocols ........................... 17  
   2.7 Multipath .......................................................... 18  
      2.7.1 Principle .................................................... 18  
      2.7.2 Different aspects of multipath ......................... 19  
      2.7.3 Advantages and Drawbacks ............................. 21  

3 Current DSR mechanisms ........................................ 23  
   3.1 Overview .......................................................... 23  
   3.2 DSR Headers [6] .................................................. 26  
   3.3 DSR data structures [6] ........................................ 36  

4 Algorithms .......................................................... 39  
   4.1 Algorithm 1: DSRM ............................................. 39  
      4.1.1 Notations .................................................. 39  
      4.1.2 Description ............................................... 40  
   4.2 Algorithm 2: DSRMP ............................................ 44  
      4.2.1 About the AOMDV algorithm ............................ 44  
      4.2.2 About our new algorithm ............................... 45
## 5 Implementation of the DSR extensions

5.1 NS2 : a GPL simulator ............................................. 48  
5.1.1 Overview of NS2 ................................................. 48  
5.1.2 TCL : the scenario interface .................................. 49  
5.1.3 The C++ code .................................................. 50  
5.1.4 General Overview .............................................. 50  
5.1.5 How to add a new protocol ................................. 51  
5.1.6 DSR in NS2 .................................................... 52  
5.2 Implementation of DSRM ....................................... 53  
5.2.1 First version .................................................. 53  
5.2.2 Second version ............................................... 56  
5.3 Implementation of DSRMP ..................................... 59  
5.3.1 First version .................................................. 59  
5.3.2 Second version ............................................... 61  
5.3.3 Third version ................................................ 64  
5.3.4 Fourth version ............................................... 65  
5.3.5 Fifth version ................................................ 67  
5.4 Installation of the DSRMP protocol ......................... 68  
5.4.1 Installation of NS2 .......................................... 69  
5.4.2 Installation of DSRMP ....................................... 69

## 6 Performance evaluation of DSR-MP

6.1 Performance evaluation ....................................... 71  
6.1.1 Area and node density .................................... 71  
6.1.2 Velocity .................................................... 72  
6.1.3 Random waypoint movement model ..................... 72  
6.1.4 Constant Bit Rate (CBR) traffic model ............... 73  
6.1.5 Routing Metrics ............................................ 73  
6.1.6 Simulation environment .................................. 74  
6.2 Analysis of various DSRMP versions ....................... 75  
6.3 Assumption used for DSRMPv5 ............................. 78  
6.3.1 Low mobility (1m/sec) ................................... 79  
6.3.2 Medium mobility (10m/sec) ............................... 82  
6.3.3 High mobility (20m/sec) ................................... 85  
6.4 Discussion about the results ................................ 88  
6.5 Group Mobility ............................................... 89  
6.5.1 Low mobility (1m/sec) ................................... 89  
6.5.2 High mobility (20m/sec) ................................... 90  
6.6 Discussion about the results ................................ 91

## 7 Conclusion and Future Work

A Tcl file

A.1 Aim of the .tcl file ........................................ 93  
A.2 The .tcl file ................................................ 93
CONTENTS

B Traffic scenarii
   B.1 1 source and 1 destination ........................................... 97
   B.2 10 sources 10 destinations version .................................. 98

C Random model file for scenarii
   C.1 Overview and usage ..................................................... 101
   C.2 File generation & name structure .................................. 102
   C.3 Scenarii organization ................................................. 103

D Java parser
   D.1 Purpose of ns-2 files ................................................ 104
   D.2 Java parser listing ................................................... 105

E Bash scripts
   E.1 gen.sh ........................................................................ 108
   E.2 runall.sh .................................................................... 109
   E.3 1src-2-10src.sh ......................................................... 110

F Matlab fonction to plot the results
   F.1 Aim of the Matlab script ............................................. 111
   F.2 The Matlab script ....................................................... 111

G CD ..................................... 113
List of Figures

2.1 The OSI Model .................................................. 12
2.2 Data encapsulation ............................................. 13
2.3 Node Disjoint .................................................... 19
2.4 Link Disjoint .................................................... 19
2.5 Performance ...................................................... 20
2.6 Security .......................................................... 21
2.7 Eavesdropping ................................................... 21

3.1 route discovery - RREQ ........................................ 24
3.2 route discovery - RREP ......................................... 24
3.3 Route Maintenance ............................................... 26
3.4 DSR Headers in an IP packet .................................. 26
3.5 DSR Headers - Fixed part ...................................... 27
3.6 Route Request Option .......................................... 28
3.7 Route Reply Option ............................................. 29
3.8 Route Error Option ............................................. 31
3.9 Acknowledgement Request Option ........................ 33
3.10 Acknowledgement Option ..................................... 33
3.11 Source Route Option ........................................... 34

4.1 Network Topology ............................................... 40
4.2 Route Research 1 ................................................. 41
4.3 Route Research 2 ................................................. 41
4.4 Results ........................................................... 43
4.5 AOMDV topology ............................................... 44
4.6 Unoptimized topology ......................................... 45
4.7 AOMDV RREP ................................................... 46

5.1 NS2 overview ..................................................... 51
5.2 Algorithm Mechanism 1 ....................................... 54
5.3 Algorithm Mechanism 2 ....................................... 54
5.4 Paths ............................................................. 55
5.5 Route Error ....................................................... 56
LIST OF FIGURES

5.6 New Route Request .................................................. 57
5.7 Error and Data Packet ................................................ 58
5.8 New Data Packet ........................................................ 58
5.9 New path used ........................................................... 59
5.10 Broadcast 1 ............................................................... 60
5.11 Broadcast 2 ............................................................... 60
5.12 Two Routes .............................................................. 61
5.13 Movement 1 .............................................................. 62
5.14 Movement 2 .............................................................. 63
5.15 Movement 3 .............................................................. 63
5.16 Three Paths v3 .......................................................... 65
5.17 Three Paths Node Disjoint v4 ....................................... 66
5.18 Three Paths Link Disjoint v4 ....................................... 67

6.1 Packet delivery ratio (20m/sec 1src-dst) .......................... 75
6.2 Normalised routing overhead (20m/sec 1src-dst) ................... 75
6.3 Packet delivery ratio (1m/sec 1src-dst) .............................. 76
6.4 Average end to end delay (1m/sec 1src-dst) ......................... 76
6.5 Packet delivery ratio (1m/sec 1src-dst) .............................. 76
6.6 Normalised routing overhead (1m/sec 1src-dst) ..................... 76
6.7 Average end to end delay (1m/sec 1src-dst pair) .................... 77
6.8 Packet delivery ratio (20m/sec 1src-dst) .............................. 77
6.9 Normalised routing overhead (20m/sec 1src-dst) ..................... 77
6.10 Improvement from Version 1 to Version 5 (20m/sec 1src-dst) .... 77
6.11 Improvement from Version 1 to Version 5 (20m/sec 1src-dst) .... 77
6.12 Packet Delivery fraction 1m/s - 1src-1dst .......................... 79
6.13 Average end to end Delay 1m/s - 1src-1dst ........................ 79
6.14 Normalized Routing Load 1m/s - 1src-1dst ........................ 79
6.15 Packet Delivery fraction 1m/s - 10src-10dst ........................ 81
6.16 Average end to end Delay 1m/s - 10src-10dst ...................... 81
6.17 Normalized Routing Load 1m/s - 10src-10dst ........................ 81
6.18 Packet Delivery fraction 10m/s - 1src-1dst .......................... 82
6.19 Average end to end Delay 10m/s - 1src-1dst ........................ 82
6.20 Normalized Routing Load 10m/s - 1src-1dst ........................ 82
6.21 Packet Delivery fraction 10m/s - 10src-10dst ........................ 84
6.22 Average end to end Delay 10m/s - 10src-10dst ...................... 84
6.23 Normalized Routing Load 10m/s - 10src-10dst ........................ 84
6.24 Packet Delivery fraction 20m/s - 1src-1dst .......................... 85
6.25 Average end to end Delay 20m/s - 1src-1dst ........................ 85
6.26 Normalized Routing Load 20m/s - 1src-1dst ........................ 85
6.27 Packet Delivery fraction 20m/s - 10src-10dst ........................ 87
6.28 Average end to end Delay 20m/s - 10src-10dst ...................... 87
6.29 Normalized Routing Load 20m/s - 10src-10dst ........................ 87
6.30 Packet delivery ratio (1m/sec 1src-dst) .............................. 89
6.31 Normalised routing overhead (1m/sec 1src-dst) ...................... 89
LIST OF FIGURES

6.32 Packet delivery ratio (20m/sec 1src-dst) .......................... 90
6.33 Normalised routing overhead (20m/sec 1src-dst) .................. 90
6.34 Average end to end delay (1m/sec 1src-dst) ......................... 90

B.1 1 source 1 destination scenario. ....................................... 98
B.2 10 sources 10 destinations scenario. .................................. 99

C.1 The Random Waypoint Algorithm ..................................... 101
C.2 Scenarii organization ...................................................... 103
List of Tables

2.1 Reactive versus Proactive protocols ............................. 17
6.1 Common packet size in bytes .................................... 73
6.2 Scenarii summary .................................................. 78
Chapter 1

Introduction

Nowadays, there is a huge increase of handled devices. Indeed, laptops, mobile phones and PDAs take an important place in the everyday life. Hence, the challenge is now to make all these devices communicate together in order to build a network. Obviously, this kind of networks has to be wireless. Indeed, the wireless topology allows flexibility and mobility. In this context, the idea of ad hoc networks was developed.

1.1 Ad Hoc Network

In 1968, engineers tried to develop a network in Hawaii which was responsible for the interconnection between educational facilities. This project has been named ALOHA (Areal Location Of Hazardous Atmospheres) and created by the University of Hawaii under the leadership of Norman Abramson. The main drawback is that ALOHA is based on a single-hop protocol. Hence, the couple source-destination has to be always within the same transmission range.\[16\], \[12\] and \[15\].

In 1973, the Defense Advanced Research Projects Agency (DARPA) developed the Packet Radio Network (PRnet) which was inspired by ALOHA. It was expected that a node could be able to communicate with all the other nodes. Nevertheless, those nodes had a narrow radio range. Hence, two distant nodes could not exchange data. The engineers took this problem into consideration and implemented the multihopping technique. Thus, a node can deliver data on behalf of another one to a determined destination.\[15\]

It is considered that those two projects lead to the known definition of ad hoc networks. Indeed, the IEEE (Institute of Electrical and Electronics Engineers) decided to replace the term Packet Radio Network by the term of Mobile Ad hoc NETwork (MANET).

Before the definition of an ad hoc network, nodes should be defined. A node can be a laptop, a PDA or another communication device. A node is characterized by its limited CPU capacity, storage capacity, bandwidth and battery power. Hence, the chosen protocol has to take into consideration that those resources are scarce.\[15\] and \[10\]

An ad hoc network is a wireless network formed by wireless nodes without any help of infrastructure. In such a network, the nodes are mobile and can communicate dynamically in an arbitrary manner. The network is characterised by the absence of central administration devices such as base stations or access points. Furthermore, nodes should
be able to enter or to leave the network easily. In these networks, the nodes act as routers. They play an important role in the discovery and the maintenance of the routes from the source to the destination or from a node to another one. This is the principal challenge to such a network. If link breakages occur, the network has to stay operational by building new routes. The main technique used is the multi-hopping which increase the overall network capacity and performances. By using multihopping, one node can deliver data on behalf of another one to a determined destination. Thus, the problem of range radio is solved.

1.2 Multihopping

Multihopping is a technique which allows the virtual extension of the transmission range for each node. Indeed, a node can send a packet to another one even if this destination node is not in the radio range from the source. A sended packet crosses several wireless links to reach the destination. That allows to save energy since a node needed to reach the destination through a single hop. But, to be efficient, it is necessary that the different nodes which participate to the transmitting process are well distributed in space. According to [15], the multihopping technique:

- Conserves transmit energy resources.
- Reduces interferences.
- Increases the overall network throughput.

1.3 Challenges of ad hoc networks

Ad hoc engineering has to solve two main problems:

- Routing: the mobility of the nodes leads to a more complex task for routing. Indeed, in most of the cases, nodes will move frequently. The changes of topology imply that a dynamic routing protocol needs to maintain routes between source and destination.

- Mobility Management: The routing algorithm has to manage the mobility of the nodes which move randomly and unpredictably in the network. In order to achieve this task, nodes might store informations concerning the topology of the network in their routing table. Hence, the success of the ad hoc network depends of both the quality of the informations collected by the node and the efficiency of the routing protocol.

There are other aspects of the ad hoc networking which could be highlighted like security where trust relationships have to be set up (use of cryptography). Indeed the use of multiple hops could be a problem because, on one hand, it facilitates the interception of the data by an unauthorized person and, on the other hand, the intentional interference (jamming) or unintentional interference due to the fact that several nodes share the same air interface domain.
1.4 Applications

In many cases, an ad hoc network could be deployed advantageously. An ad hoc network should be useful for military troops which want to set up a network in an hostile area where wired devices are not adequate. In a conference, the audience would be able to share informations. We can also imagine the case of an airport where associates want to exchange data and, in the same time, are informed on the weather, the schedule of the departures and so on. They can also choose to establish a connection with a backbone network like Internet. Ad Hoc Network will also take an important role in domains such as domotic where several robots will be able to communicate in order to achieve the different tasks they are implemented for.[15], [10] and [12]

The future for this kind of networks seems to be very clear, due to the fact that the market of wireless communications grows up every years. Ad hoc will be needed everywhere, people will want to communicate by deploying a cheap or temporary network. Efforts are also made to implement such a network in PAN (Personal Area Network) to interconnect, for instance, domestic appliances. This would lead to lower prices which is one of the main reason while mobile computing growth rapidly.[15]

1.5 Goal of the study

The objective of this work is to implement a multipath protocol based on an existing one. We have chosen DSR for his reliability and because it seems to impose itself as the best in the ad hoc network field. To build this new algorithm, multipath concept should be clearly assimilated. This new algorithm has been evaluated and compared with other algorithms which had shown a qualitative behaviour.

The goal of our study is to:

- Understand the concept of ad hoc networks
- Get an overview of the main protocols in ad hoc networks
- Study in detail the mechanism of an ad hoc protocol (DSR)
- Work on a new multipath protocol
- Find solutions to create multipath algorithms
- Build this protocol
- Evaluate this protocol in comparison with other protocols tested in different studies
- Interpret the results obtained
1.6 Outline of the report

Since the subject of this study is "Cross Layer approach in Ad Hoc Network", it is necessary to become familiar with some notions linked with Network Engineering. Our contributions per chapter are as follows:

In chapter 2, we define some essential concepts as the OSI Layer Model or the definition of multipath protocols. Indeed, nodes are devices for communication and so, should obey a certain number of rules according to the standards defined by the OSI Model. That allows us to highlights concepts linked with the Network layer. Additionally, the multipath concept is defined in this chapter in order to ensure that the reader understand our choices for the implementation work.

In chapter 3, the protocol DSR is studied. Its main characteristics are viewed in details. It was necessary to become familiar with this protocol since the protocol we achieved is based upon DSR. This chapter highlights the parameters which can be changed in the C++ code of DSR.

In chapter 4, our two multipath algorithms are introduced theoretically. To achieve a relevant protocol, an algorithm is a critical set of rules since it defines how nodes, and globally the network, should manage and exchange information. For each of them, the reader is helped to understand their mechanisms. Indeed, simple schemes were used for a better understanding.

In chapter 5, the both multipath solutions are implemented and tested. The results show that our second algorithm behaves very well. Hence severals improvments were added and then, several versions were achieved. This chapter describes how the results of the tests done have guided our choices to achieve a better protocol.

In chapter 6, a comparison between DSR, AODV and DSRMP (our protocol) is described. Indeed by this way, we can show how our protocol react in comparison with two well-known protocols for ad hoc network. Two mobility models had been used: the random waypoint model and the group mobility model. DSRMP results show a reliable multipath protocol which is able to outperform AODV and original DSR.
Chapter 2

General Aspect: Routing in an ad hoc network

2.1 The OSI layer

Network protocols links to the OSI (Open Systems Interconnection) reference model, and represents the totality of protocol definitions and provide international standardization of many aspects of computer-to-computer communication. From [4] Figure 2.1 is a brief explanation of the purpose of each layer in that model.

![Figure 2.1: The OSI Model.](image-url)
2.2 Data encapsulation and layer distribution

- Physical layer: Physical medium control, bit transmission and timing.
- Data link layer: Divided into MAC and Logical sub layer: Link control and data transmission.
- Network layer: Network addressing and routing.
- Transport layer: Last chance to correct network errors.
- Session layer: Applications synchronization and connection management.
- Presentation layer: Conversion between different machine representations.
- Application layer: High level application support tools.

In this project, only a part of a network function will be studied: the third layer of the OSI model, the network layer, specifically the routing process.

2.2 Data encapsulation and layer distribution

Data obtain from application layer, e.g. data from a soft phone need to be encapsulated in order to be send across the network, figure 2.2 is schematics of how this process works.[4]

![Figure 2.2: Data encapsulation.](image)
2.3 Routed and routing protocols

Before explaining the specificity of routing in an ad hoc network a definition of a routed and a routing protocol should be done.\cite{7} and \cite{17} (1)

- **Routed protocols:** Any network protocol that provides enough information in its network layer address to allow a packet to be forwarded from host to host based on the addressing scheme. Routed protocols are nothing more than data being transported across the networks. They define the format and use of the fields within a packet. Packets generally are conveyed from end system to end system. IP (Internet Protocol), Telnet, RPC (Remote Procedure Call), SNMP, SMTP are examples of routed protocols.

- **Routing protocols:** facilitate the exchange of routing information between networks, allowing routers to build routing tables dynamically. Routing protocols are the software that allow routers to dynamically advertise and learn routes, determine which routes are available and which are the most efficient routes to a destination. RIP and RIP II (Routing Information Protocol), OSPF (Open Shortest Path First), BGP (Border Gateway Protocol) are examples of routing protocols.

In this project the way a mobile node determines the link with its joint node will be improved, using an existing ad hoc routing protocol: DSR (Dynamic Source Routing Protocol)

2.4 Problems encountered while routing in an ad hoc network

Routing in a MANETs is an important aspect of a protocol, and will influence on the overall end to end ”quality” of the link. Each routing protocol has a specific domain and purpose in which it can be used. Below are some aspects that should take into account before defining in which way the protocol should be used.

In reference number \cite{13} and \cite{6} (2) some of the following parameters are given like:

- Network size/scalability: as the number of nodes increases, the greater the burdens on a flat routing scheme is, and hence hierarchical routing protocols may be needed for large number of nodes. In addition, some networks (e.g. mobile military networks or highway networks) may be relatively large (e.g. tens or hundreds of nodes per routing area), therefore a scalable protocol should be necessary.

- Geographical area: how spread out the nodes are. E.g in a in an amusement park, where a high density of nodes are available or in an automotive situation where density can be from high during rush hours to low during afternoons.

- The topology rate of change: this is a major performance-impacting parameters for an ad hoc routing protocols. For example, the proactive routing protocols are very inefficient with a highly changing topology, since a lot of routes finding is wasted.
2.5 Existing routing protocols

- **QoS:** Wireless links will continue to have significantly lower capacity than their hardwired counterparts. In addition, the realized throughput of wireless communications is often much lower than a radio’s maximum transmission rate. The effects of multiple access, fading, noise, and interference conditions, should be taken into account. The route acquisition time is a particular aspect of end-to-end delay and is a special aspect of ”on demand” routing algorithms as the time required to establish route(s) is longer than only searching in a existing lookup table.

- **Energy-constrained operation:** Some or all of the nodes in a MANET may rely on batteries. For these nodes, the most important system design criteria for optimization may be energy conservation.

- **Limited physical security:** Mobile wireless networks are generally more prone to physical security threats than are fixed cable nets. The increased possibility of eavesdropping, spoofing, and denial-of-service attacks should be carefully considered. One strength of the multi hop networks, is the decentralized nature of network control in MANETs which provides additional robustness against the single points of failure of more centralized approaches.

### 2.5.1 Existing routing protocols

Several types of routing protocol exists. Indeed, since certain protocols are designed for specific cases, two mains categories were created:

1. **proactive (table-driven) protocol.** Periodically floods the network to reconstruct the routing table. E.g. CGSR (Clusterhead Gateway Switch Routing protocol), DSDV (Destination-Sequenced Distance Vector routing protocol), LCA (Linked Cluster Architecture)... Their main characteristics are:
   - low route latency
   - high overhead (periodic table updates)
   - route repair depends on update frequency

The main proactive protocols are:

**DSDV** (Destination Sequenced Distance Vector): It is a hop-by-hop distance vector. Each node periodically broadcasts routing updates. By using its sequence number, DSDV is able to avoid loops. [10]

**OLSR** (Optimized Link State Routing): (October 2003) OLSR is an optimization of the classical link state algorithm tailored to the requirements of a mobile wireless LAN. The key concept used in the protocol is that of multipoint relays (MPRs). MPRs are selected nodes which forward broadcast messages during the flooding process. This technique substantially reduces the message overhead as compared to
2.5 Existing routing protocols

a classical flooding mechanism, where every node retransmits each message when it receives the first copy of the message. [6] (4)

2. reactive (on-demand) protocol. As long as the route is working no attempt to search for another one. E.g. ARA (Ant-based Routing Algorithm for Mobile Ad-Hoc Networks), BSR (Backup Source Routing protocol)... Their main characteristics are:

- high route latency
- no overhead from periodic update
- route caching can reduce latency

The main reactive protocols are:

AODV (Ad hoc On Demand Distance Vector): (July 2003) This protocol offers quick adaptation to dynamic link conditions, low processing and memory overhead, low network utilization, and determines unicast routes to destinations within the ad hoc network. [6] (5)

DSR (Dynamic Source Routing) This protocol uses packets header which contains the complete ordered list of nodes through which the packet must pass. It is also characterized by the fact of using two particular mechanism: route discovery and route maintenance. They both are described later in Chapter 3. [10]

TORA (Temporally-Ordered Routing Algorithm routing protocol)(20th July 2001) In this document it is mentioned that a key concept in the protocol’s design is an attempt to de-couple the generation of far-reaching control message propagation from the dynamics of the network topology. This protocol has expired but is implemented into NS-2. [6] (7)

A third category could be derived from the two previous ones, using some quality of one type and enhancing it with the participation of the other one. For instance, using a reactive protocol, where the route discovery is done only when a communication is requested, and caching the available routes in a case of a link failure, which is an aspect of proactive routing. These protocols are called hybrid protocols.

In the table 2.1, protocols characteristics are highlighted according to their type:
Table 2.1: Reactive versus Proactive protocols

<table>
<thead>
<tr>
<th></th>
<th>Reactive - on-demand</th>
<th>Proactive - table-driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>overhead</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>memory requirement</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>cope with mobility</td>
<td>good</td>
<td>bad</td>
</tr>
<tr>
<td>sleep time</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>purpose</td>
<td>relatively high mobility</td>
<td>low mobility</td>
</tr>
</tbody>
</table>

2.6 Other ad hoc networks protocols

A seen on [17] (2) other categories of ad hoc networks exists like:

- **Hierarchical:**
  - CBRP Cluster Based Routing Protocol expired
  - CEDAR Core Extraction Distributed Ad hoc Routing expired
  - FSR Fisheye State Routing Protocol expired
  - LANMAR Landmark Routing Protocol expired
  - ZRP (Zone Routing Protocol protocol) expired
  - BRP (Bordercast Resolution Protocol) - work in progress, July 2002.
  - IARP (Intrazone Routing Protocol) expired
  - IERP (Interzone Routing Protocol) expired

- **Multicast:**
  - ABAM (On-Demand Associativity-Based Multicast)
  - MZR (Multicast Zone Routing)
  - DDM (Differential Destination Multicast) expired
  - DSR have multicast in current release
  - MAODV (Multicast Ad hoc On-Demand Distance Vector routing) expired

- **Geographical:** DREAM (Distance Routing Effect Algorithm for Mobility), LAR (Location-Aided Routing protocol)
2.7 Multipath

- Power aware: ISAIAH (Infra-Structure Aodv for Infrastructured Ad Hoc networks), PARO (Power-Aware Routing Optimization Protocol)...

- Geographical Multicast (Geocasting): LBM (Location Based Multicast), MRGR (Mesh-Based Geocast Routing)...

- other IMEP (Internet Manet Encapsulation Protocol) expired
  
  All are not supported by IETF (CGSR, DBF, SSR...), or have expired [6] (STAR, RDMAR, BRP...).

2.7 Multipath

Ad hoc network could be implemented on various mobile devices. But these devices could be split in different categories. In example a laptop device has more battery life, more processing power, and lower mobility as a handled device. On another hand a PDA had a higher probability to move and can be use far away from a power source. On this project we want to implement and enhanced a general protocol. Therefore the main aspects will be:

- lightweight
- low memory requirement
- low processing requirement

One widely used protocol which has such requirement is DSR. Therefore it will be studied and improved. In networks, the multipath technique is the establishment of several routes between a single source node and a single destination node.

2.7.1 Principle

Multipath routing consists of three components: route discovery, route maintenance and traffic allocation.

- Route discovery is the process of finding a route between two nodes.
- Route maintenance is the process finding broken routes, repairing them or finding a new route in the presence of a route failure.
- The main issues of traffic allocation strategy are to decide how the data will be sent and distributed in the network via available paths.
2.7 Multipath

Two kinds of links can be used:

- Node disjoint

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{node_disjoint.png}
\caption{Node Disjoint.}
\end{figure}

- Link Disjoint

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{link_disjoint.png}
\caption{Link Disjoint.}
\end{figure}

This type of link assumes that every path has no link in common; hence, a path can have common nodes, as long as paths keep being different.

2.7.2 Different aspects of multipath

Performances

Performances cover different aspects. One of those is the data rate. Splitting the message and using different paths at the same time to send different packets can increase the data rate: it would be similar to connect source and destination more than once. To achieve this, a link disjoint topology should be used:
2.7 Multipath

Indeed, we can assume that the processing time for one node is greater than the time for transmission, meaning that one node can handle more than one packet at a time. However, the problem of collision is still to be handled, but it is the same in a node disjoint or in a link disjoint configuration.

Energy consumption

Two ways to limit energy consumption:

(Flooding) First of all, the path discovery request can be limited: a node keeps in its memory all the possible paths (even with common links), and as soon as a link is broken, this node chooses another path from its memory and erases the old one.

(Sending) The overall performance can be improved by choosing the most cost efficient path, with different criteria (power consumption, distance between the nodes, number of errors, etc.). It is also possible to share the load between the nodes by choosing alternate paths, i.e. using the full capacity of the network. When a link is highly used, it alleviates congestion by using less used links. Hence, the topology would be node disjoint.

Fault-tolerance

Multipath can provide route resilience, and therefore the network can recover quickly from any change.

Quality of Service (QoS)

This is also a consequence of a higher aggregate bandwidth and end to end delay as the data could be delivered in a fixed delay.

Reliability

To ensure the transmission of a packet from a source to a destination, multipath can provide different ways. Two of them are flooding and error correction. Flooding a packet to every path known to reach a destination is a way to bring more reliability about a transmission. Sending different packet by different paths, it is possible to use a code error correction to allow the destination to rebuild a missing packet. This would increase the probability of a successful transmission.
2.7 Multipath

Security/Privacy

Multipath can be used to provide a little bit of security. Indeed, splitting a message and spreading the packets using different paths for one another, no intermediate node would be able to rebuild the message. Indeed, no node would receive the entire message, except the destination.

![Security](image)

**Figure 2.6: Security.**

However, it could still be possible for a node to listen to another node’s packets, hence, to rebuild the message.

![Eavesdropping](image)

**Figure 2.7: Eavesdropping.**

In this example (figure 2.7), one path goes through A, another one goes through B. If A is near B, A can eavesdrop B’s packets and rebuild the entire message.

Multicast

This procedure permits to send the same packet to multiple users, avoiding the retransmission of the same packet several times. Whereas multipath implies the use of several routes between one source and one destination, multicast is about sending one packet from one source to multiple destinations.

2.7.3 Advantages and Drawbacks

As it is discussed above, multipath can bring different kinds of advantages to an Ad Hoc Network, be it by increasing the data rate, optimizing the energy consumption, robustness of the network, etc. Nevertheless, as in many systems, those improvements are based on a trade off that has to be taken into account in order to really optimize a protocol.

Hence, several drawbacks can be pinpointed in the use of such a technique. As it is said before, the cost of storing extra route could be very important, especially if a huge amount of mobile nodes is present in the network.
2.7 Multipath

The transmission interference is also a problem. Transmissions from a node along a path may interfere with transmissions from a node along another path, thereby limiting the achievable throughput. The different paths should be therefore as independent as possible. But it is also necessary to take care of the fact that the different routes are not in the same collision domain. All those things make multipath routing much more complex than unipath routing.

Packets reordering, if the data are split into several packets sent through several paths, adds difficulties to such a technique.
Chapter 3

Current DSR mechanisms

In this chapter, DSR mechanisms will be discussed. Shortly introduced in a first section where some of the main characteristics will be highlighted, then a detailed description of the headers used in the current protocol, and finally, a description of the conceptual structures used to support DSR operation.

3.1 Overview

DSR protocol is composed by two “on-demand” mechanisms ([6] (6) page V), which are requested only when two nodes want to communicate with each other. Route Discovery and Route Maintenance are built to behave according to changes in the routes in use, adjusting themselves when needed. Along with those mechanisms, DSR allows multiple routes to any destination, thus can lead easily to load balancing or increase robustness.

- Route Discovery

  Route Discovery is the mechanism by which a node S (Source) wishing to send a packet to a Destination node D obtains a source route to D. Route Discovery is used only when S attempts to send a packet to D and does not already know a route to D.

  - The source first has to check in its ”Route Cache” if it knows a suitable route for the destination. If no route is found, it will have to start a route discovery protocol to find a route to the destination.

  - The route discovery itself consists on a chain of locally broadcasted Route Request (RREQ). The broadcasting occurs until one of the broadcasted RREQ reaches either the destination node or a node who knows a route to that destination.

  When a node receives a RREQ, it checks in its Route Request Table if the same RREQ has already been received. In that case, the packet is discarded. Hence, if an intermediate node knows a route to the destination, the source
3.1 Overview

may receive more than one route to the destination. Otherwise, the Route Discovery allows only one path to one destination.

A Route Request contains a unique request identification (2, in the example figure 3.1), determined by the initiator of the Request, and a target destination (E).

If a node receives a RREQ and is not the target it appends its own address to the route record in the RREQ and propagates it by transmitting it as a local broadcast.

If a node receiving the RREQ has recently seen another RREQ message from this initiator bearing this same request identification and target address, or if this node’s own address is already listed in the route record in the RREQ, this node discards the Request.

**Figure 3.1: route discovery - RREQ.**

- The Route Reply (RREP, figure 3.2) is then emitted. It is assumed here that links are symetrical, hence, the RREP uses the reversed route from the RREQ. When the source receives this RREP, it caches this route in its Route Cache.

**Figure 3.2: route discovery - RREP.**

- The source node in particular, and any node in general, cache routing informations received from packets or simply overheard. This use of explicit source
3.1 Overview

routing allows the sender to select and control the routes used for its own packets, and supports the use of multiple routes to any destination. Moreover, the Route Cache should support storing more than one route to each destination [6].

- Route Maintenance

*Route Maintenance* is the mechanism by which node S is able to detect, while using a source route to D, if the network topology has changed such that it can no longer use its route to D because a link along the route no longer works. When Route Maintenance indicates a source route is broken, S can attempt to use any other route it happens to know to D, or can invoke Route Discovery again to find a new route for subsequent packets to D. Route Maintenance for this route is used only when S is actually sending packets to D.

Each node is responsible for its own links, i.e. each node must confirm that data can flow over the link from that node to the next hop. This information can be provided by an acknowledgement from different sources.

- As an existing standard part of the MAC protocol in use.
- By a passive acknowledgement (a node overhears a transmission of the packet it has just forwarded)
- If a built-in acknowledgment mechanism is not available, the node transmitting the packet can explicitly request a DSR-specific software acknowledgment be returned by the next node along the route; this software acknowledgment will normally be transmitted directly to the sending node, but if the link between these two nodes is unidirectional, this software acknowledgment could travel over a different, multi-hop path.

A link is considered to be broken when a node has been unable to verify the reachability of a next-hop node after reaching a maximum number of retransmission attempts [6]. The broken link is then removed from the cache of the former node, and a Route Error is sent to every node which use the broken link and which is in its Route Request Table.

A node may salvage a packet if it knows another route to the packet’s destination. However, a count is maintained in the packet of the number of times that it has been salvaged, to prevent a single packet from being salvaged endlessly.
3.2 DSR Headers [6]

The DSR protocol uses specific headers to carry control informations. Those headers can be included in any IP packet.

Those headers are created from two different parts:

- A fixed-sized part of 4 bytes
- Zero or more DSR options

Generally, DSR headers follow the IP header in the packet. In fact, the Protocol field in the IP header is used to indicate that a DSR Options header follows.

the DSR Options header must be a multiple of 4 bytes in the case other headers follow the DSR Options header (This is to preserve the alignment of these following headers in the packet).
Fixed portion of DSR Option header

The fixed portion of the DSR Options header is used to carry information that must be present in any DSR Options header. This fixed portion of the DSR Options header has the format shown in the figure 3.5.

![Figure 3.5: DSR Headers - Fixed part](image)

- **Next Header**

  8-bit selector. Identifies the type of header immediately following the DSR Options header. Uses the same values as the IPv4 Protocol field.

- **Flow State Header (F)**

  Flag bit. Must be set to 0. This bit is set in a DSR Flow State header and clear in a DSR Options header.

- **Reserved**

  Must be sent as 0 and ignored on reception.

- **Payload Length**

  The length of the DSR Options header, excluding the 4-octet fixed portion. The value of the Payload Length field defines the total length of all options carried in the DSR Options header.

- **Options**

  Variable-length field; the length of the Options field is specified by the Payload Length field in this DSR Options header. Contains one or more pieces of optional information (DSR options), encoded in type-length-value (TLV) format (with the exception of the Pad1 option).
3.2 DSR Headers [6]

![Figure 3.6: Route Request Option](image)

**Route Request Option**

**IP fields:**

- **Source Address**

  Must be set to the address of the node originating this packet. Intermediate nodes that retransmit the packet to propagate the Route Request must not change this field.

- **Destination Address**

  Must be set to the IP limited broadcast address (255.255.255.255).

- **Hop Limit (TTL)**

  May be varied from 1 to 255, for example to implement non-propagating Route Requests and Route Request expanding-ring searches.

**Route Request fields:**

- **Option Type**

  Nodes not understanding this option will ignore this option.

- **Opt Data Len**

  8-bit unsigned integer. Length of the option, in octets, excluding the Option Type and Opt Data Len fields.
3.2 DSR Headers [6]

- Identification

A unique value generated by the initiator (original sender) of the Route Request. Nodes initiating a Route Request generate a new Identification value for each Route Request, for example based on a sequence number counter of all Route Requests initiated by the node.

This value allows a receiving node to determine whether it has recently seen a copy of this Route Request: if this Identification value is found by this receiving node in its Route Request Table (in the cache of Identification values in the entry there for this initiating node), this receiving node must discard the Route Request. When propagating a Route Request, this field must be copied from the received copy of the Route Request being propagated.

- Target Address

The address of the node that is the target of the Route Request.

- Address[1..n]

Address[i] is the address of the i-th node recorded in the Route Request option. The address given in the Source Address field in the IP header is the address of the initiator of the Route Discovery and must be listed in the Address[i] fields; the address given in Address[1] is thus the address of the first node on the path after the initiator. The number of addresses present in this field is indicated by the Opt Data Len field in the option \( n = \frac{\text{Opt Data Len} - 6}{4} \). Each node propagating the Route Request adds its own address to this list, increasing the Opt Data Len value by 4 octets.

The Route Request option must appear more than once within a DSR Options header.

Route Reply Option

The Route Reply option in a DSR Options header is encoded as shown in figure 3.7.

![Route Reply Option](image)

**Figure 3.7:** Route Reply Option
3.2 DSR Headers [6]

*IP fields:*

- **Source Address**

  Set to the address of the node sending the Route Reply. In the case of a node sending a reply from its Route Cache or sending a gratuitous Route Reply, this address can differ from the address that was the target of the Route Discovery.

- **Destination Address**

  Must be set to the address of the source node of the route being returned. Copied from the Source Address field of the Route Request generating the Route Reply, or in the case of a gratuitous Route Reply, copied from the Source Address field of the data packet triggering the gratuitous Reply.

*Route Reply fields:*

- **Option Type**

  Nodes not understanding this option will ignore this option.

- **Opt Data Len**

  8-bit unsigned integer. Length of the option, in octets, excluding the Option Type and Opt Data Len fields.

- **Last Hop External (L)**

  Set to indicate that the last hop given by the Route Reply (the link from Address[n-1] to Address[n]) is actually an arbitrary path in a network external to the DSR network; the exact route outside the DSR network is not represented in the Route Reply. Nodes caching this hop in their Route Cache must flag the cached hop with the External flag. Such hops must not be returned in a cached Route Reply generated from this Route Cache entry, and selection of routes from the Route Cache to route a packet being sent must prefer routes that contain no hops flagged as External.

- **Reserved**

  Must be sent as 0 and ignored on reception.

- **Address[1..n]**

  The source route being returned by the Route Reply. The route indicates a sequence of hops, originating at the source node specified in the Destination Address field of
3.2 DSR Headers [6]

the IP header of the packet carrying the Route Reply, through each of the Address[i] nodes in the order listed in the Route Reply, ending with the destination node indicated by Address[n]. The number of addresses present in the Address[1..n] field is indicated by the Opt Data Len field in the option (n = (Opt Data Len - 1) / 4).

A Route Reply option MAY appear one or more times within a DSR Options header.

Route Error Option

The Route Error option in a DSR Options header is encoded as shown in figure 3.8

![Route Error Option Diagram](image)

**Figure 3.8: Route Error Option**

- **Option Type**
  
  Nodes not understanding this option will ignore this option.

- **Opt Data Len**
  
  8-bit unsigned integer. Length of the option, in octets, excluding the Option Type and Opt Data Len fields.

  For the current definition of the Route Error option, this field must be set to 10, plus the size of any Type-Specific Information present in the Route Error. Further extensions to the Route Error option format may also be included after the Type-Specific Information portion of the Route Error option specified above. The presence of such extensions will be indicated by the Opt Data Len field. When the Opt Data Len is greater than that required for the fixed portion of the Route Error plus the necessary Type-Specific Information as indicated by the Option Type value in the option, the remaining octets are interpreted as extensions. Currently, no such further extensions have been defined.

- **Error Type**
  
  The type of error encountered. Currently, the following type values are defined:
- 1 = NODE_UNREACHABLE
- 2 = FLOW_STATE_NOT_SUPPORTED
- 3 = OPTION_NOT_SUPPORTED

Other values of the Error Type field are reserved for future use.

- **Rsv**
  
  Reserved. Must be sent as 0 and ignored on reception.

- **Salvage**
  
  A 4-bit unsigned integer. Copied from the Salvage field in the DSR Source Route option of the packet triggering the Route Error.

  The “total salvage count” of the Route Error option is derived from the value in the Salvage field of this Route Error option and all preceding Route Error options in the packet as follows: the total salvage count is the sum of, for each such Route Error option, one plus the value in the Salvage field of that Route Error option.

- **Error Source Address**
  
  The address of the node originating the Route Error (e.g., the node that attempted to forward a packet and discovered the link failure).

- **Error Destination Address**
  
  The address of the node to which the Route Error must be delivered. For example, when the Error Type field is set to NODE_UNREACHABLE, this field will be set to the address of the node that generated the routing information claiming that the hop from the Error Source Address to Unreachable Node Address (specified in the Type-Specific Information) was a valid hop.

- **Type-Specific Information**
  
  Information specific to the Error Type of this Route Error message.

A Route Error option may appear one or more times within a DSR Options header.

**Acknowledgement Request Option**

The Acknowledgement Request option in a DSR Options header is encoded as shown in figure 3.9.
3.2 DSR Headers

Figure 3.9: Acknowledgement Request Option

- Option Type

Nodes not understanding this option will remove the option and return a Route Error.

- Opt Data Len

8-bit unsigned integer. Length of the option, in octets, excluding the Option Type and Opt Data Len fields.

- Identification

The Identification field is set to a unique value and is copied into the Identification field of the Acknowledgement option when returned by the node receiving the packet over this hop.

An Acknowledgement Request option must not appear more than once within a DSR Options header.

Acknowledgement Option

The Acknowledgement option in a DSR Options header is encoded as shown in figure 3.10.

Figure 3.10: Acknowledgement Option

- Option Type

Nodes not understanding this option will remove the option.
3.2 DSR Headers [6]

- **Opt Data Len**

  8-bit unsigned integer. Length of the option, in octets, excluding the Option Type and Opt Data Len fields.

- **Identification**

  Copied from the Identification field of the Acknowledgement Request option of the packet being acknowledged.

- **ACK Source Address**

  The address of the node originating the acknowledgement.

- **ACK Destination Address**

  The address of the node to which the acknowledgement is to be delivered.

An Acknowledgement option may appear one or more times within a DSR Options header.

**DSR Source Route Option**

The DSR Source Route option in a DSR Options header is encoded as shown in figure 3.11.

![Source Route Option Diagram](image)

**Figure 3.11: Source Route Option**

- **Option Type**

  Nodes not understanding this option will drop the packet.
3.2 DSR Headers [6]

- **Opt Data Len**

  8-bit unsigned integer. Length of the option, in octets, excluding the Option Type and Opt Data Len fields. For the format of the DSR Source Route option defined here, this field must be set to the value \((n \times 4) + 2\), where \(n\) is the number of addresses present in the Address\([i]\) fields.

- **First Hop External (F)**

  Set to indicate that the first hop indicated by the DSR Source Route option is actually an arbitrary path in a network external to the DSR network; the exact route outside the DSR network is not represented in the DSR Source Route option. Nodes caching this hop in their Route Cache must flag the cached hop with the External flag. Such hops must not be returned in a Route Reply generated from this Route Cache entry, and selection of routes from the Route Cache to route a packet being sent must prefer routes that contain no hops flagged as External.

- **Last Hop External (L)**

  Set to indicate that the last hop indicated by the DSR Source Route option is actually an arbitrary path in a network external to the DSR network; the exact route outside the DSR network is not represented in the DSR Source Route option.

  Nodes caching this hop in their Route Cache must flag the cached hop with the External flag. Such hops must not be returned in a Route Reply generated from this Route Cache entry, and selection of routes from the Route Cache to route a packet being sent must prefer routes that contain no hops flagged as External.

- **Reserved**

  Must be sent as 0 and ignored on reception.

- **Salvage**

  A 4-bit unsigned integer. Count of number of times that this packet has been salvaged as a part of DSR routing.

- **Segments Left (Segs Left)**

  Number of route segments remaining, i.e., number of explicitly listed intermediate nodes still to be visited before reaching the final destination.
3.3 DSR data structures [6]

- Address[1..n]

The sequence of addresses of the source route. In routing and forwarding the packet, the source route is processed. The number of addresses present in the Address[1..n] field is indicated by the Opt Data Len field in the option \( n = (\text{Opt Data Len} - 2) / 4 \).

When forwarding a packet along a DSR source route using a DSR Source Route option in the packet’s DSR Options header, the Destination Address field in the packet’s IP header is always set to the address of the packet’s ultimate destination. A node receiving a packet containing a DSR Options header with a DSR Source Route option must examine the indicated source route to determine if it is the intended next-hop node for the packet and determine how to forward the packet.

3.3 DSR data structures [6]

Different conceptual data structures are important to support DSR operation. The most important ones for this topic are described in this section.

Route Cache

The Route Cache contains every routing information collected by and useful for the node. A node can collect these routing information by different means:

- Route Request
- Route Reply
- DSR Source Route

A node can also remove those informations according to different criteria:

- Route Error
- Link layer retransmission mechanism

The Route Cache allows to store more that one route to each destination. The mechanism to select a route from the Route Cache, as well as the management of space within a node’s cache or the data structure organisation is mostly dependent on the implementation. Each entry in the Route Cache should have a timeout associated with it, to allow that entry to be deleted if not used within some time.
Send Buffer

The Send Buffer of a node implementing DSR is a queue of packets that cannot be sent by that node because it does not yet have a source route to each such packet’s destination. As for the Route Cache, every packet in the Send Buffer is associated with a timeout. A FIFO strategy should be used to prevent the buffer from overflowing.

According to the rate limitation, a Route Discovery should be initiated as often as possible for the destination address of any packets residing in the Send Buffer.

Route Request Table

This Route Request Table of a node stores information about Route Requests that have been recently originated or forwarded by this node:

- Time-to-Live (TTL): Allows the node to implement a variety of algorithms for controlling the spread of its Route Request on each Route Discovery initiated on a target
- The time that this node last originated a Route Request for that target node
- The number of consecutive Route Discovery initiated for this target since receiving a valid Route Reply giving a route to that target node
- The remaining amount of time before the next Route Discovery
- The identification value and target address from the most recent Route Requests received by this node from that initiator node

Gratuitous Route Reply Table

The Gratuitous Route Reply Table is used to limit the rate at which it originates gratuitous Route Replies to the same original sender for the same node from which it overheard a packet to trigger the gratuitous Route Reply.

This table contains the following fields:

- The address of the node to which this node originated a gratuitous Route Reply
- The address of the node from which this node overheard the packet triggering that gratuitous Route Reply
- The remaining time before expiration of this entry
3.3 DSR data structures [6]

Network Interface Queue and Maintenance Buffer

The Network Interface Queue of a node implementing DSR is an output queue of packets from the network protocol stack waiting to be transmitted by the network interface (represented sometimes by a \texttt{struct ifqueue}). This queue is used to hold packets while the network interface is in the process of transmitting another packet.

The Maintenance Buffer of a node implementing DSR is a queue of packets sent by this node that are awaiting next-hop reachability confirmation as part of Route Maintenance. To prevent overflowing, a LIFO strategy should be used.

Blacklist

This structure is necessary when the interface requires physically bidirectional links for unicast transmission (for instance, 802.11). This table indicates that the link between a node and the specified neighbor node may not be bidirectional.
Chapter 4

Algorithms

Since the goal of this study is to implement a DSR protocol which manages multiple routes, an algorithm is necessary. This chapter introduces two algorithms built for the project: DSRM and DSRMP. These algorithms are the result of a new strategy to find routes. Hence, the route discovery process is modified. We describe these two algorithms by giving for each one a concrete and simple example. Since DSRMP performs better than DSRM, it has been chosen for the implementation.

What does a node know?
An ad hoc network is composed by several nodes which interact in order to communicate together. The knowledge of a node is directly linked with the kind of protocol used. In source routing protocol such as DSR, a node can learn little information about the network, contained in a sending packet. Thus, the node knows the source node, the destination node, the route through which the packet travelled. From this last data, the node knows the number of hops necessary to reach it. This information is critical for the below algorithm.

4.1 Algorithm 1: DSRM

4.1.1 Notations

The DSRM algorithm pursues principally the goal to find node-disjoint routes. In order to allow the reader to better understand its mechanism, we will use the following notation:

- Node Source: S
- Node Destination: D
- Intermediate Node: Z

In the below figure, the link between two nodes means that they are in the communication range of each others. In order to describe the algorithm, the following ad hoc network topology is chosen. It is obvious that our algorithms were tested in randomized cases but the figure ?? will help the reader to follow our explanations.
4.1 Algorithm 1: DSRM

4.1.2 Description
In this section, the behavior of the different nodes of the ad hoc network is described.

**Node S:** This node generates the RREP packet and sends it as in the normal DSR protocol.

**Node Z:** In this algorithm, when a node receives a RREQ packet, there are two possibilities.

- It is the first packet received, generated by the Node S. So, this node considers that this packet went through the shortest route to reach it. Hence, the node Z counts the number \( X \) of hops and puts it in its cache.

- It is not the first packet received, generated by the Node S. Another RREQ packet arrived before. Nevertheless, Node Z receives it and counts the number \( P \) of hops. By this way, it is able to compare with the number \( X \).

  - If \( P > X \), the packet is discarded
  - If \( P < \) or = to \( X \), node Z takes into account this packet.

Hence, the number \( X \) is determinant in the process of finding routes.

All the intermediate nodes between the node S and the node D react in the same way as the above mechanism description (figures 4.2 and 4.3). After those operations, each node broadcasts the RREQ packet after having added in it its own identifier.

**Node D:** The destination node is configured to receive a maximum number \( R \) of routes and is able to launch up to \( R \) unicasted RREP packets.
4.1 Algorithm 1: DSRM

Node D considers that the best and the shortest route is the first one which arrives in its cache. Hence, the destination refers itself to this route to make the following decisions.

Node D gives a number for each route according to their order of arrival i.e. the number 1 for the first and the number 2 for the second.

Node D takes the first number as the reference route and compares it with the others. All the routes which have one or several common nodes with the reference route will be discarded. The remaining routes stay available.

Node D takes the smallest following number in its cache (not the second because it could have been discarded). Node D considers this route as the reference route and compares it with the others. All the routes which have one or several common nodes with
4.1 Algorithm 1: DSRM

the reference route will be discarded.

Node D continues until the remaining routes are all node-disjoint. After this, it can unicast the RREP packet to the node S. Hence several paths are established.

In the figure number , the route which will arrive to the destination are:

Number 1: S-2-5-D
Number 2: S-2-4-D
Number 3: S-2-6-D
Number 4: S-1-4-D
Number 5: S-3-6-D
Number 6: S-3-5-D
Number 7: S-1-5-D

According to the algorithm, the destination node takes the route number 1 as reference. Then, it discarded all the routes which contain one or several common nodes between the source and the destination.

Number 1: **S-2-5-D**  Reference
Number 2: S-2-4-D  Discarded
Number 3: S-2-6-D  Discarded
Number 4: S-1-4-D
Number 5: S-3-6-D
Number 6: S-3-5-D  Discarded
Number 7: S-1-5-D

In this example, only four routes are still available:

Number 1: S-2-5-D
Number 4: S-1-4-D
Number 5: S-3-6-D
Number 7: S-1-5-D

Now, the destination node takes another route as reference: the route with the smallest number except the number 1 which has already been considered as reference. The process of discarding routes continues.

Number 1: S-2-5-D
Number 4: **S-1-4-D**  Reference
Number 5: S-3-6-D
Number 7: S-1-5-D  Discarded
4.1 Algorithm 1: DSRM

The destination node considers the last one as reference:

Number 1: S-2-5-D
Number 4: S-1-4-D
Number 5: S-3-6-D Reference

Finally, the destination node has three node disjoint paths in its cache. It will be able to unicast an RREP through each of those paths (figure 4.4). This example shows a way to find a simple and reliable algorithm.

![Diagram](image)

**Figure 4.4: Results**

**Drawbacks:**
The main drawback of such an algorithm is that it can broadcast a huge amount of RREQ packet since a node broadcasts a packet for each received routes. Indeed a same node can broadcast a huge amount of RREQ packet if the network is composed by many nodes. Moreover we can predict that in an area with a high density of nodes, the algorithm will not work in a desired manner since the routing overhead increases significantly. Nevertheless, in a well distributed network, without areas where nodes are concentrated, this kind of algorithms ensure the finding of node disjoint route. This ability could be particularly useful for studies concerning reliable packet splitting in ad hoc networks.
4.2 Algorithm 2: DSRMP

4.2.1 About the AOMDV algorithm

We have based our second algorithm on the extension of the AODV to the AOMDV protocol (http://www.cs.sunysb.edu/~mahesh/aomdv/). The AOMDV protocol improves the route discovery. It tries to find different paths with a link disjoint configuration, as in the figure 4.5.

![AOMDV topology](image)

**Figure 4.5: AOMDV topology**

The node 1 wants to discover routes to the nodes 7. So it sends a broadcast. This broadcast is received by the node 2 and 3. The node 2 replies first and sends another broadcast. The nodes 4 and 3 receive the broadcast from 2. In the cache of the node 3, there are now 2 broadcasts, one from the node 1 and one from the node 2. The node 3 sends only one broadcast: it is received by the node 2 and 4, which keep it in their cache. Then the node 4 sends one broadcast, and its the same scenarios for the nodes 5, 6, 7.

At the end of the route discovery, we have several nodes, which contain different broadcasts in their cache. The cache of every node is describes in the follow table:

<table>
<thead>
<tr>
<th>nodes</th>
<th>1st Broadcast(from)</th>
<th>2nd Broadcast(from)</th>
<th>3rd Broadcast(from)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>6</td>
<td>-</td>
</tr>
</tbody>
</table>

The destination node receives 2 broadcast. It replies to the first. The reply goes to the node 5, which check is cache and then uses the node 4, then the node 2. The path 12457 is available.

The destination answers to the second broadcast. It sends the reply to the node 6, which sends to the nodes 4. The node 4 sends the reply to the node 3 and not the node 2 because it has already uses this node for the first reply. Then the route 134-67 is available.
4.2 Algorithm 2: DSRMP

<table>
<thead>
<tr>
<th>nodes</th>
<th>1st Broadcast(from)</th>
<th>2nd Broadcast(from)</th>
<th>3rd Broadcast(from)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>6</td>
<td>-</td>
</tr>
</tbody>
</table>

This algorithm looks to perform well. But in some configurations like in figure 4.6, it does not optimize the routes.

![Unoptimized topology](image)

**Figure 4.6: Unoptimized topology**

In the previous topology, the AOMDV algorithm will discover 2 routes: 1246 and 13256. Its due to the fact that the nodes 2 makes its broadcast before the node 3, and then the nodes 3 and 4 have the broadcast from the node 2 in the top of their cache.

It will be better if the two routes were 1246 and 1356.

### 4.2.2 About our new algorithm

Our protocol is based on the AOMDV algorithm. We want only to improve it to abort what happens in the previous example.

The route discovery will work on the same way. We only need to add the length of the path in the cache of every node. Its not very difficult, because the header in DSR contains all nodes, and then we only have to count them.

The thing that we can change is the route reply.

First, we have to define the number of routes that we need. Usually, 3 routes is the best compromise: more routes will not improve the global performance. So the destination nodes has to reply to only 3 broadcast, and has to allocate 3 different numbers for each
4.2 Algorithm 2: DSRMP

reply.

From the previous example, the cache of the nodes will look like this:

<table>
<thead>
<tr>
<th>nodes</th>
<th>1\textsuperscript{st} Broadcast(from)</th>
<th>2\textsuperscript{nd} Broadcast(from)</th>
<th>3\textsuperscript{rd} Broadcast(from)</th>
<th>4\textsuperscript{th} Broadcast(from)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>1(1)</td>
<td>1-3(2)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1(1)</td>
<td>1-2(2)</td>
<td>1-2-4(3)</td>
<td>1-2-5(3)</td>
</tr>
<tr>
<td>4</td>
<td>1-2(2)</td>
<td>1-3(2)</td>
<td>1-2-5(3)</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>1-2(2)</td>
<td>1-3(2)</td>
<td>1-2-4(3)</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>1-2-4(3)</td>
<td>1-2-5(3)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

A node does not stock a path if it has already passed by itself, and a node will send only one broadcast, with the first path received.

The first reply will proceed well because any link will be already used (it is the first reply).

![Image of node connections](image)

**Figure 4.7: AOMDV RREP**

<table>
<thead>
<tr>
<th>nodes</th>
<th>1\textsuperscript{st} Broadcast(from)</th>
<th>2\textsuperscript{nd} Broadcast(from)</th>
<th>3\textsuperscript{rd} Broadcast(from)</th>
<th>4\textsuperscript{th} Broadcast(from)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>1(1)</td>
<td>1-3(2)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1(1)</td>
<td>1-2(2)</td>
<td>1-2-4(3)</td>
<td>1-2-5(3)</td>
</tr>
<tr>
<td>4</td>
<td>1-2(2)</td>
<td>1-3(2)</td>
<td>1-2-5(3)</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>1-2(2)</td>
<td>1-3(2)</td>
<td>1-2-4(3)</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>1-2-4(3)</td>
<td>1-2-5(3)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

The node 6 will send only two replies because it receives only two broadcasts.

The second reply will use the path 1-2-5. But the node 2 will reject it because the link between the nodes 1 and 2 is already use by the first reply.

The node 2 can check its cache and replace the link 1-2 by the link 1-3-2. By doing
4.2 Algorithm 2: DSRMP

this, the node 2 will add one hop: it is what it’s done in the AOMDV implementation.

We suggest sending a reply to the previous node (node 5) to notify that the node 2 will add 1 hop. So the previous node can check in its cache if it has a shorter route and if it has, use it or send a reply to the previous node again.

In the cache of every node, we can increment the length of the route for each path using the node 2. It can be node for each node receiving the reply.

Then the cache of every node will look like this.

<table>
<thead>
<tr>
<th>nodes</th>
<th>1st Broadcast(from)</th>
<th>2nd Broadcast(from)</th>
<th>3rd Broadcast(from)</th>
<th>4th Broadcast(from)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>1(1)</td>
<td>1-3(2)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1(1)</td>
<td>1-2(2)</td>
<td>1-2-4(3)</td>
<td>1-2-5(3)</td>
</tr>
<tr>
<td>4</td>
<td>1-2(2)</td>
<td>1-3(2)</td>
<td>1-2-5(3)</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>1-2(3)</td>
<td>1-3(2)</td>
<td>1-2-4(1)</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>1-2-4(3)</td>
<td>1-2-5(3)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Now, the node 5 can decide to use the link 1 - 3 instead of the link 1 - 2, because it contains the same number of hops than the previous path.

The second reply will use the path 1 - 3 - 5 - 7.
Implementation of the DSR extensions

The implementation part is an important time of the project. By implementing the different solutions, we can test them, find some improvements and understand why one works better than another.

We have chosen to work with NS2. NS2 is available under Linux, with a GPL license. Some standard algorithms are already implemented in this simulator, and DSR is one of these.

Hence, the two protocols have been built based on the existing DSR implementation. Then, they have been tested with this simulator. After the tests, we have improved the protocols according to some unexpected factors, or some cross layer problems. So, we have built two versions of the DSRM protocol and five versions of the DSRMP protocol.

5.1 NS2 : a GPL simulator

5.1.1 Overview of NS2

NS2 is a network simulator; built with C++ and TCL. As every simulator, the main purpose is to simulate different networks, to test different protocols, and to find the limitations of each. It has been developed in the California University, by LBL, Xerox PARC, UCB, and USC/ISI through the VINT project supported by DARPA.

First, this simulator was build for fixed network: all links among nodes were wired. That means that the neighbour had no direct neighbour: if two nodes were very close, they don’t communicate each other if they don’t have a cable between each other. So, later, an extension for wireless network was developed by UCB Daedelus, CMU Monarch projects and Sun Microsystems.

Nowadays, this simulator is used around the world, because of the GPL license, and
5.1 NS2 : a GPL simulator

because it is a powerful simulator. It can be downloaded from the Internet, at this URL: http://www.isi.edu/nsnam/ns/.

There are some tutorials to help beginners, and there is a lot of documentation (as the manual of ns2 - actually 400 pages).

The simulator is composed of two parts:

- The TCL code: it is used to communicate with the simulator, and permits to define different simulation parameters
- The C++ code: it is the main part of the project, because it defines how the simulator has to behave

5.1.2 TCL : the scenario interface

The main reason for using a TCL language in ns2 is because it is not useful to use only C++ code. In fact, by this mean, the user does not need to compile the simulator every time he wants to do a new simulation. The TCL language is interpreted by the C++ code in ns2, without being compiled.

To use a network simulator, you have to define two things:

- How does the protocol behave? It is done by the C++ code, because it does not change for every simulation
- What are the simulation parameters? It is done by the TCL code, because every simulation is different (number of nodes, positions, speeds, protocol used)

The TCL code allows the user to choose between fix or wireless networks, and among the different implemented protocols: TORA, AODV, DSR (for wireless networks).

The TCL file contains also information about nodes like position and speed, or informations about source and destination, the transmission rate, and a lot of other parameters.

The syntax of this language is defined in the ns2 manual.

Some tools have been developed to build these scenarios. For example, if we want to have a random model with several nodes, it is possible to use ‘setdest’. It is a tool that generates random positions and random speeds for a number of nodes. By doing this, it is easy to use different random models and to test a protocol.
5.1 NS2 : a GPL simulator

5.1.3 The C++ code

Like C++, TCL is an object oriented language. So, there are parallels between C++ objects, the TCL objects. A C++ object can be used in the TCL language.

If the user needs to share a C++ object with the TCL code, he needs to use the TclObject. This class is developed in the tcltl package, independently to ns2. It allows to define the TCL name of the C++ object. Then the C++ object is used in the TCL file, using this name.

The TCL object communicates with its corresponding C++ object by using some basic commands. These commands are defined in the commands() function of the C++ commands, which takes the known parameters argc and argv. Thus, the TCL object can initialise the C++ object.

The main purpose of the C++ code is to define how the simulator works, independently from the simulation parameters, and the results depend only on different initial conditions.

For instance, some C++ objects represent the different layers of the different nodes in the simulation. So, when a packet is sent from one node to another one, the packet goes through different C++ objects.

In the wireless layer, these packets are received in the recv function if the node is the destination, and in the tap function if the node is a neighbour and can listen the packet even if it is not the destination.

The simulation runs with a specific simulation time, not the real one. Then, by sending packets, C++ objects creates some events: they want that one C++ object receives one packet at a time, by introducing a delay for example. But because ns2 is a mono process program, there is a table, containing all the events and sorting them according to the time they occur.

5.1.4 General Overview

There are two kinds of interaction with the C++ code:

- The first one is with TCL files, which can describe the initial conditions. It can also describe some events, like the change of speed for example

- The second one is the generation of events by the C++ code itself

The C++ code produces some output files, which contain different types of results from the simulation. For example, one can be used to have a graphic display of the simulation using the tool NAM. Another one contains trace of all the packets, and can be analysed with other tools to have statistics about the simulation.
5.1 NS2 : a GPL simulator

5.1.5 How to add a new protocol

There are two ways to implement the new algorithms in ns2:

- Because the new algorithms are based on the DSR protocol, it is possible to change directly the DSR protocol and do the tests with the modified version
- Or to add a new protocol in ns2

We have chosen the second solution, because it is more practical to have both DSR and the new protocol in the same version of ns2 for testing purposes. Indeed, it is easier to compare their performances.

So, we copied the DSR folder and create two new one: DSRM and DSRMP for the two new algorithms. It is not as much a simple task as one can think at the beginning. By copying the folder, it is necessary to change all the class names and all the names of the static variables. If you do not do this, the simulator will not be able to compile, because of confusion in the object names.

Then, each protocol uses its own packet. So it is essential to define specific packets for the new protocol in the common/packet.h file.

Other different C++ files have to be changed:

- /queue/dsr-priqueue.cc : just to add the packet type of our protocol, declared in the packet.h file
- emph/trace/cmu-trace.cc(.h) : to have the correct trace file corresponding to our protocol
5.1 NS2 : a GPL simulator

- **Makefile** : to compile ns2 with the new protocol

To finish, the link between TCL objects and our protocol objects has to be built. So, it is necessary to create a TCL object corresponding to the C++ object in the `tcl/mobility` folder, and to change these files:

- `tcl/lib/ns-default.tcl`
- `tcl/lib/ns-lib.tcl`
- `tcl/lib/ns-mobilenode.tcl`

These three files create the correct TCL object corresponding to the protocol chosen in the simulation parameters. It is a kind of a switch among the wireless protocols.

5.1.6 DSR in NS2

The DSR folder is inside the main ns2 folder. It contains a lot of C++ files. Some of them are not used and are just there because they were used before an updated version.

The main file is the `dsragent.cc`. It contains the `DSRAgent` class, which corresponds to TCL objects. All interactions with the simulator occur by using this class.

The other classes, defined in the other files, are some kind of tools, used in the `DSRAgent object`, to simplify the code:

- `hdr_sr.cc(.h)`: contains the class `hdr_sr`, that represents the header of the DSR packet: it defines if the packet is a request, a reply, a route error, or a data packet. It contains the path that the packet has to use and all different parameters.

- `path.cc(.h)`: contains two useful classes: `ID` and `Path`. `ID` is used to define the address of a node (MAC or IP), and `Path` contains an array of nodes and a lot of possible operations that can be done on them.

- `srpacket.h`: contains the class `SRPacket`. When a DSRAgent receives a packet, it is the common class `Packet`. Then the packet is transform in `SRPacket` class, which is more useful to manipulate; `hdr_sr` is contained in the `Packet` object.

- `requesttable.cc(.h)`: contains the class `RequestTable`. It contains all information about the different requests received by a node. It is useful, because an intermediate node have to make only one broadcast, even if it receives it more than once.

- `flowstruct.cc(.h)`: contains the class `FlowTable`. This object contains a table that references all paths overheared by the node from any source to any destination. Then this table can be used to shorten the paths or to make a faster reply by sending directly a reply if an intermediate node already knows the destination.
5.2 Implementation of DSRM

- `requesttable.cc(.h) / mobicache.cc` : these three files implement the class `RequestTable`. This object contains a list of destination, and different paths to reach every one of them.

The other files are not used in the dsr implementation, it includes : `add_sr.cc, cache_stats.h, dsr_proto.cc(.h), linkcache.cc, simplecache.cc, sr_forwarder.cc(.h)`.

The DSRAgent class uses all of those classes, to make the DSR protocol working, as described before.

### 5.2 Implementation of DSRM

Based on the DSR protocol, we have implemented two versions of DSRM, corresponding to the first algorithm: intermediate nodes can send more than one reply for one request. The second version of the DSRM protocol has been built to minimize the effect the amount of overhead due to a non expected effect from DSR implementation.

#### 5.2.1 First version

As described in the previous chapter, the algorithm is rather simple. If a node needs to find a path to a destination, it proceeds to a route discovery like in DSR. Then, every intermediate node answer to every received broadcast by doing another one only if the number of node in the path is inferior or equal to the last one recorded. Then the destination node can list all the found paths, and choose them in order to have a node disjoint topology.

To translate it in C++ code, it induces an addition of two new classes:

- One containing a table for the intermediate nodes
- One containing another table for the final node

The table of the class for the intermediate nodes contains the history of all requests received by a node. A request can be easily identified by the address of the source and by the request number. Then corresponding to each request, the table associates the minimum number of nodes used to reach the source. If the intermediate node receives a request containing more nodes in its path than the minimum indicated in the table, the node will discard this request.

If the path contains the same number of nodes, the intermediate node will send another broadcast, and if the path contains fewer nodes, the intermediate node will send another
5.2 Implementation of DSRM

broadcast too, and will update the minimum in its table.

The table of the class for the final node contains all paths accepted by the node. In fact, it is the final node that decides which paths are going to be used or not. When the final node receives the first request, it sends it back to the source and adds the path in its table. When the final node receives another request, it checks if the path of the request is node disjoint with all the paths in its table. If it is, then the final node sends a reply to the source and adds the path in its table. If it is not, the final node just discards the request.

By using this algorithm, the final node knows every path to the source. As they will not change during the replies, the final node can add these paths in its route cache.

The next example (figures 5.2 and ??) is a simple one, and is used to illustrate the mechanisms of this algorithm. The source is the node 0, and the destination the node 7.

Some intermediate nodes will make more than one broadcast. All broadcasts are listed in the next table, in the order they occur.
5.2 Implementation of DSRM

<table>
<thead>
<tr>
<th>Broadcast number</th>
<th>Occurs in the node</th>
<th>Comes from the node</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Only the nodes 4, 5 and 6 made more than one broadcast. It is normal because the other nodes (1, 2, 3) have a direct link with the source (node 0). Having a direct link means that the minimum size of the path from the source is 1 node (the source), and no other broadcast can have a shorter path.

Then, nodes 4 and 6 made two broadcasts, because 4 can not listen 3, and 6 can not listen 1. Node 5 made three broadcasts, because it listened nodes 1, 2 and 3.

There are 10 intermediate broadcasts, and one initiated by the source. So the total of broadcasts is 11, for a simulation with 8 nodes.

At the end of the replies, the source and the destination get three paths in their route caches (figure 5.4). These three paths are node disjoint.

![Figure 5.4: Paths](image-url)
5.2 Implementation of DSRM

5.2.2 Second version

The DSRM version 1 works, and provides 3 paths to the source (node 0), that are node disjoint.

Now, to send its packet, the source node will use one of these three paths. If one is broken, then the node will choose another one.

In the last scenario, node 0 will first use the path 0-2-4-7. Now, if the node 7 moves in the top right corner, the link 2-4 will be broken before the link 4-7.

Hence, the node 0 sends its packet to node 2. Node 2 sends the packet to node 4, which sends the packet to the destination node. But at a certain time the link 2-4 breaks. So, the node 2 will not be able to reach the node 4. The node 2 can detect that the link is broken, because we are using the 802.11 link layer protocol, and when it sends the packet, node 2 did not receive the acknowledgement.

So the node 2 is going to send a route error to the node 0 to specify that a link is broken (figure 5.5).

![Figure 5.5: Route Error](image)

However, the data packet is still in the node 2, and the node 2 has to send it to the destination node. The node 2 does not have any path in its route cache to reach the destination node. So the node 2 will initialise a route request (figure 5.6) to reach the destination node, since the source node sends its packet using a new path.

This procedure increases the number of requests, and then increases the number of broadcasts. The number of broadcasts is more important in DSRM than in DSR; the intermediate nodes can answer more than one time for the same broadcast. So, this
5.2 Implementation of DSRM

In DSR, this kind of scenario does not occur. In fact, DSR uses a tap system. It means that every node listen its neighbour. If the neighbour has used a route to the destination node, the node will know that it can reach the destination node by using its neighbour. For instance, in the last example, one reply is sent by the node 4, one by 5 and one by 6. The reply from the node 4 is sent to the node 2, but the node 1 can also listen this reply, and will know that it can reach the destination node by using the node 4.

In our case, if the tap system was used, the node 2 would know that it can use the node 5, 6, 1 and 3 to reach the destination node, and it will not initiate a route request, saving routing overhead.

The tap system has been disabled in the DSRM protocol. In fact, the main purpose of this protocol is to use node disjoint paths. If a node is allowed to listen to its neighbours and add some paths in its cache, it is not necessary to build a multipath protocol. Then all the paths in the route cache of the nodes will be less correlated, and the probability to find another path in the node’s cache if failed down is higher.

But there is no sense to have fewer requests in the source node if we have more in the intermediate node. So, we have built the DSRM version 2 to avoid as much as possible the request in the intermediate node.

The principle is very simple. The intermediate node will send an error packet to the source node. Instead of sending only an error packet, we send the data packet back mixed with the error packet. Then the source node will use its cache to send the data packet again, but by another way.

The link 2-4 breaks down, so the node 2 sends the data packet mixed with the error packet to the source node (figure 5.7).
5.2 Implementation of DSRM

Then the source node will use another path (0 - 1 - 5 - 6) to send the packet again (figure 5.8). The next packet sent by the node 0 will use the same path (figure 5.9).

Thus, the DSRM protocol works, but only for small scenarios. In fact, with simple scenarios, like the previous one, the number of nodes is not big: in our case, it corresponds to 8 nodes, which leads to a correct number of broadcasts.

By increasing the number of nodes, the number of broadcasts induced from one request will increase exponentially. With a scenario of 50 nodes for examples, the number of overhead packets is so big, that most of the data packets are dropped due to collisions or to full buffers.

That is the reason why we spent more time with the next algorithm.
5.3 Implementation of DSRMP

The DSRMP algorithm proposes an extension of the AOMDV algorithm. It is quite difficult to implement it in one time, so we did it step by step. First, we have only implemented the AOMDV algorithm, based on DSR. Then, according to the test, we have built some new versions. The last version is not exactly the algorithm described in the previous chapter, but it looks similar. In fact, to avoid overhead, we have used the tap system instead of sending back replies.

5.3.1 First version

In the DSRM algorithm, the main task was done by the final node: this node chooses the replies to send back. So, all information has to be send to the destination node: it is the reason of the large amount of overhead. In the DSRMP protocol, this task is shared with the intermediate nodes: fewer overhead. But the reply paths are built during the way back. So, only the source node will know the path to the destination node. If the destination node wants to reach the source, it has to proceed to a new request.

The implementation of the DSRMP protocol is a little more complicated than the DSRM, because the task has to be shared among the intermediate nodes.

But instead of adding two classes, it is necessary to build only one:

- One containing a table for the intermediate nodes

The functions of this class is completely different from those of the DSRM algorithm.

In the DSRMP protocol, every intermediate node has to make one and only one broadcast for one request. So, in the table of the class for the intermediate nodes, every
5.3 Implementation of DSRMP

node has to put all paths received from one request. For example, if two nodes are about to process a request, every intermediate node will have two lists in its table. The first list corresponds to one request and the second one to the other one. Each list is filled with the path received from the corresponding request in the intermediate node. Each request can be identified by the address of the source and the request number. And, for each list, the intermediate nodes will make only one broadcast.

Then, when an intermediate node receives one reply, it just adds the first path from its cache on it. For the second reply, it adds the second one, and etc. If an intermediate node does not have any more paths in its cache, it just drops the reply.

The destination node has to do nothing, just replies when a request arrives.

By testing the new protocol with the previous simple scenarios, all intermediates nodes make only one broadcast (figure 5.10 and 5.11).

![Broadcast 1](image1.png) ![Broadcast 2](image2.png)

**Figure 5.10:** Broadcast 1  **Figure 5.11:** Broadcast 2

The order of the broadcasts is listed in the following table:

<table>
<thead>
<tr>
<th>Broadcast number</th>
<th>Made in the node</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>
The previous table will be useful in the description of the next version.

Now, by doing this simulation, the source node gets only two routes as shown in figure 5.12.

This result is quite unexpected. It is the mix of two main problems:

- Why are there only two paths available, when the source can have three?
- Why the second path (the blue one) is going to pass by the node 1 after the node 3, instead of going directly to the source node?

The first question is solved in the version 3 of DSRMP.

The second question is a consequence of the last one, and to the fact that we have implemented the AOMDV without our extension. In fact, even if the second path is not as good as possible, it forms a link disjoint topology with the first one.

5.3.2 Second version

As in the DSRM protocol, the tap system has been disabled, because DSRMP is a multipath protocol, and the nodes need to have paths in their cache which are less correlated.

So, before to solve the two main problems of the version 1, we have implemented the second version of DSRMP, that minimises the request from the intermediate nodes.

The process implemented in DSRMP is different from the one of DSRM, for three main reasons:
5.3 Implementation of DSRMP

- It is more simple
- The process is similar to the former DSR: the protocol built is a multipath protocol based on DSR, so DSRMP differs from DSR only by the multipath aspect
- It is more adaptive for high mobility scenarios

The idea is to use exactly the same system as in DSR: the DSRMP protocol is using the tap system. But the routes discovered by the tap system are not added in the route cache of the nodes. They just fill a new cache. This second cache can be used by an intermediate node in the case of a broken link occurs.

This implementation is more adaptive for high mobility scenario, because when a link fails from an intermediate node to the destination, the probability that another link fails between the intermediate node and the source is high.

In the previous example, two routes have been discovered: 0-1-4-7 and 0-2-1-3-6-7. It is first one that is used to send the packet from the source node (node 0) to the destination node (node 7).

But the node 4 is going to move in the top right corner. So, the link 1-4 is going to be break. Then, the intermediate node (node 1) sends an error packet to the source node (node 0) and sends the data packet to the node 3 (figure 5.13), because when checking in its second cache, it knows that it can reach the destination node from the node 3.

![Figure 5.13: Movement 1](image)

Then, the node 3 sends directly the data packet to the destination node (using the node 6 as intermediate node), as shown in figure 5.14.
5.3 Implementation of DSRMP

The source node (node 0), after receiving the error packet, sends its packet with its second path (0-2-1-3-6-7), as shown in figure 5.15.

At the end, no data packet has been dropped.
5.3 Implementation of DSRMP

5.3.3 Third version

The third version has been built to avoid the fact that in the simple scenario, the source node adds only two paths in its cache, instead of three.

By debugging the second version of DSRMP, we can see that the destination node sends three replies back. By checking more carefully, we can understand that the first reply disappears between the node 3 and the node 0.

Finally, this problem didn’t occur in the routing layer, our DSRMP protocol, but in the link layer. The link layer in ns2 used the 802.11 protocol. Before to send any packet, each node sends an ARP broadcast to make sure that the next node is one of its neighbour. When the node 3 wants to send the reply to the node 0, it does the same. But the ARP broadcast has a collision with another reply. So the reply packet in node 3 is lost, because the node 0 can not answer to the ARP broadcast coming from the node 3.

This problem can not be entirely solved in our protocol, because it occurs in the layer below. But it is possible to improve our solution.

In fact, the collision occurs because the destination node sends many replies at the same time. So, if the destination can uncorrelate the replies, by adding a delay for example, the probability to have a collision will decrease.

In the third version of the DSRMP protocol, a delay is added between each reply. It means that one new class has been added in the C++ code:

- One containing a table for the final node

The final node has to count the number of replies to send, and add a delay between each. Because all requests will arrive in the final node more or less at the same time, this delay has to be increased at every reply.

This implementation will not increase the end to end delay to much, because the first request is sent back without delay, and the destination will use this first path to send its packets, even if all replies have not reached the source.

By simulation with the previous scenario, the source node will effectively add three paths in its cache (figure 5.16).

First, the paths are not optimized. They all use the node 3 instead of using directly the nodes 2 and 1 to reach the source. Then, two paths uses the same link. It proves that AOMDV is not a link disjoint protocol. The reason of it will be explain in the next version.

By comparison with the previous results, the red path 0-3-5-7 has been added. This
5.3 Implementation of DSRMP

Figure 5.16: Three Paths v3

path corresponds to the lost reply.

Then, the blue path 0-2-3-1-4-7 corresponds to the previous path 0-1-4-7. It does not use the link 0-1 again, because the green path 0-1-3-6-7 uses it already. The first sent reply is the red one, then the green, and the last one is the blue.

During the simulation of the version 2, this order was the same. But the green and the blue replies were sent at about the same time. Because the green path has to reach one node more to go from the destination node to the node 1, the blue reply reached the node 1 before the green one. The blue reply was allowed to use the link 0-1.

Now, by introducing a delay between each reply, the green reply will reach the node 1 before the blue one, and the link 0-1 will be allocated for the green reply. So the blue reply has to find an other path, and by checking in its request cache, the node 1 will attribute the node 3: the node 3 has made its broadcast before the node 2. But the link 0-3 is already used by the red reply, so the node 3, will send the reply to the node 2.

5.3.4 Fourth version

The third version of DSRMP works, but does not provide good results. The paths are long and all of them use a same node instead of using neighbours to reach the source node.

The first problem is that two replies can use the same link, as shown in the previous example. But one reply uses the link in one way, and the second one in the other one. For instance, to reach the destination, the green path uses the link 1-3, and the blue path uses the link 3-1.

It is simple to avoid this scenario. When an intermediate node receives a reply from one of its neighbours, it just drops all routes in its cache using directly the neighbour. It
5.3 Implementation of DSRMP

corresponds to only one route, because the neighbour can have made one broadcast only.

Even if replies will not use the same link by implementing that, the routes found are not optimized. The extension of AOMDV proposed in the last chapter has to be implemented.

However this extension is quite complicated to implement. An intermediate node has to choose to send back a reply or not: it will increase the routing overhead traffic, even if the number of packet due to routing overhead will be the same. So, the solution implemented is based on the tap system.

By listening the replies of its neighbour, a node can know which node have been already used by other replies, and which are not. Then, when an intermediate node receives a reply, it can check its cache, select all the shortest routes, and choose one, which has been less used than the others.

To improve this mechanism, the intermediate nodes will use a new field in the reply header. In this field, intermediate nodes just tell the number of nodes that they will use in addition if they receive another reply. Then the neighbours, by listening this reply, will be able to calculate the new length of the path if they want to use this node.

So, in the C++ code, two variables have been added in the intermediate table:

- One to indicate the number of time that the neighbours have been used
- One to indicate the new length of the path by using one neighbour

Then, the system of the cache has been changed to take into account all the replies.

By simulation with the previous example, we get 3 paths, which are node disjoint (figure 5.17).

![Figure 5.17: Three Paths Node Disjoint v4](image-url)
5.3 Implementation of DSRMP

The source node has in its cache the route 0-3-5-7, the route 0-2-6-7 and the route 0-1-4-7.

With another scenario where the paths can not be node disjoint, we get the next result, the node 0 is the source node, and the node 9 is the destination node (figure 5.18).

![Three Paths Link Disjoint v4](image)

Figure 5.18: Three Paths Link Disjoint v4

First the route 0-2-4-7-9 is added in the source cache. Then, it is the route 0-3-5-8-9; these two routes are node disjoint. Then the route 0-1-4-6-9 is added. This route is link disjoint with the first one.

5.3.5 Fifth version

The fourth version of DSRMP performs well for scenarios with low mobility. Hence, the fifth version is an update of the fourth one, due to some problems in high mobility. So, it is not possible to describe this version with the previous scenario.

In high mobility, the number of packets due to routing overhead can be much bigger in v4 than with the normal DSR, and in some case, comparing to the second version of DSRMP too.

This unexpected result is due to the delay. Adding a delay permits to uncorrelate the replies. But, in high mobility, the probability to get a broken link during this delay is high. So the destination source will send the reply back, and the reply will not be able to reach the source, because of broken links.

Then, it will be the same procedure as for the version 2. The intermediate node will find a route to send the packet to the source node. Some new request will occur, increasing considerably the number of packets due to overhead.

To avoid this, if a link breaks, all reply packets are dropped in the fifth version. This process is normal to understand, because there is no sense to send back a reply to a source,
5.4 Installation of the DSRMP protocol

if one of its links is already broken.

By doing this the number of packets due to overhead decreases.

For example, with a high mobility scenarios (20m/s), with no pause time, we obtain the next results for the different versions of DSRMP:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Number of packets sent</th>
<th>Number of packets received</th>
<th>Routing overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSR</td>
<td>14,876</td>
<td>11,111</td>
<td>4,584</td>
</tr>
<tr>
<td>DSRMP v1</td>
<td>14,876</td>
<td>6,534</td>
<td>178,228</td>
</tr>
<tr>
<td>DSRMP v2</td>
<td>14,876</td>
<td>12,430</td>
<td>14,492</td>
</tr>
<tr>
<td>DSRMP v3</td>
<td>14,876</td>
<td>11,641</td>
<td>19,661</td>
</tr>
<tr>
<td>DSRMP v4</td>
<td>14,876</td>
<td>12,183</td>
<td>16,468</td>
</tr>
<tr>
<td>DSRMP v5</td>
<td>14876</td>
<td>12,668</td>
<td>8,819</td>
</tr>
</tbody>
</table>

This case is based only on 1 scenario. The results can be very different depending on the random variables. However, this case shows the different routing overhead packets.

In the first version, the number of overhead packets is very important, because when a link breaks, every intermediate node process to a new request. There are so many routing overhead packets, the sent packets can not reach the destination.

So, in the second version of DSRMP, the number of routing overhead packets decreases a lot. Then the transmission can be established: there is less traffic, and fewer collisions.

Nonetheless, in the third version, the number of routing overhead packets increases again: this is due to the fact that some intermediate nodes starts new requests to process reply packets.

In the fourth version, the number of routing overhead packets decreases again, due to the fact that all replies are less correlated.

It is in the last version of DSRMP that the number of routing overhead packets is rather small. It permits to increase a little bit the transmission quality. Nevertheless, the number of routing overhead packets is still higher than the normal DSR protocol.

5.4 Installation of the DSRMP protocol

As DSRM, the protocol built from our first algorithm, showed really bad results, mainly because of the broadcast storm phenomenon, this DSRM protocol is not available for installation. It would complicate a lot the whole installation: DSRM and DSRMP would
use common files, and there will be four cases: DSRM is installed and not DSRMP, or
DSRM is not installed but DSRMP is, and the cases where both are not installed or
installed.

Thus, only the different versions of DSRMP installation are taken into account.

In this part, we are going to describe quickly how to install ns2, and then how to
install the DSRMP extension.

5.4.1 Installation of NS2

The simplest way to install the ns2 simulator is to download the current release 2.27 at
the URL: http://www.isi.edu/nsnam/ns/ (one version is available in the CD given with
this report)

It is a package that contains all needed library. So, you need to unzip the package
where you want to install ns2, and then write the command ./install. Ns 2 will be installed
after that.

At the end of the installation, the program will ask you to add three environment
variables. To do this, you need to edit your profile file (generally in the /etc directory) or
you have to write the three commands every time you run a new consol.

To complete the installation, you can do some test of the ns2 performances with your
hardware. To do so, just run the command ./validate. The tests will take a very long
time (around 1 hour).

Now, ns2 is installed in your computer.

5.4.2 Installation of DSRMP

There are two ways to install the DSRMP extension:

- You can use the patch that we have developed. It has been tested only in a knoppix
distribution, and it does not work with the cygwin emulator. So, if it does not work
you have to use the second way.

- You can do the installation manually.

Installation with the patch

The installation of DSRMP is easy. You just need to run the batch file ./patch, by
giving two arguments. The first one is the path to the ns2 directory (not the ns2-27
allinone folder, but the ns2-27 folder inside) and the version you want to install. For example:

```
./patch /usr/bin/ns2-27allinone/ns2-27 v5
```

will install the last version of DSRMP. You can install the version from \textit{v1} to \textit{v5} or \textit{clear} the DSRMP protocol from ns2.

Then the DSRMP extension will be installed, and you will be allowed to run simulations with DSRMP protocol. But, if the installation failed, you have to use the second method.

**Installation by copying file**

In the CD given with the report, you have a folder called \textit{/DSRMP extension/Without patch}. Inside this folder there are 5 folders containing all the needed files to install the five versions of DSRMP.

And inside each directory, there is a readme file explaining what you have to do. Notice that if you have already installed a version of DSRMP, you will not need to change the common files again.

After copying all the files you need, you have to run the command \textit{./configure}, and then \textit{make} in the ns2-27 directory. Ns2 will compile again, and you will be able to run simulations with the DSRMP protocol.
Performance evaluation of DSR-MP

Wireless simulation models have been enhanced thanks to the support of CMU research group. This includes features like a complete radio propagation, data link and medium access control (MAC) layer models. MAC uses IEEE 802.11 specifications for wireless LANs and an unslotted carrier sense multiple access (CSMA) technique with collision avoidance (CSMA/CA) to transmit the data packets. The radio model uses characteristics similar to a commercial radio interface, Lucent’s WaveLAN and has a nominal bit rate of 2 Mb/s and a radio range of 250 m [5]. Simulation results provide us with knowledge of the behavior of the protocol under different simulation environments.

6.1 Performance evaluation

Below are the parameters used to evaluate the evolution of DSRMP versions, as well as the final one called DSRMPv5, cross-compared with other proactive protocols: DSR and AODV.

6.1.1 Area and node density

In order to see how a protocol performs on a specific topology, some restricted scenarii have been used with a maximum of 8 to 10 nodes. But to evaluate the overall performance of a protocols, larger scenarii have been used.

A choice of 50 random moving nodes on a squared 1000 m by 1000 m area has been used since most studies uses parameters close to these ones. These parameters have been set in order to be close to reality and to be not too computer intensive. To simulate these 1980 scenarii, a week of processing on a university computer had been required.

The size and density of the area has been chosen in order to obtain multiple node connectivity. An area of 1000 m by 1000 m which covers an area of $1 \times 10^6$ m$^2$ has been chosen. As each nodes can transmit up to 250 m it can covers an area of nearly 1570m$^2$, hence a mean connectivity of 12 nodes could be achieved.
6.1 Performance evaluation

A simulation time of 250 s is used, which is long enough to evaluate packet loss and to have a similar-to-reality model.

6.1.2 Velocity

As the velocity is an important parameter for a proactive or a reactive protocol and as it is known that DSR performs better in low mobility than in high mobility both scenarii has been studied. First two speed grades have been used: a high speed (20m/s) and low speed (1m/s).

The high speed set of scenarii is close to the velocity of a car (20m/s or 72km/h) and the low speed one is similar to the velocity of a pedestrian (1m/s or 3.6km/h).

But as the difference between 20m/s and 1m/s is large a medium value of 10m/s has been introduced, which is close to the speed of a (fast) bicycle, in order to obtain more accurate results.

Three set of scenarii have been produced (see appendix for details):

- for pedestrian using speed of 1m/s (ie: sq_randomMOV_1_250s_v2-p0-0)
- for a bicycle using a speed of 10m/s (ie: sq_randomMOV_10_250_v2-p0-0)
- for s car using a speed of 20m/s (ie: sq_randomMOV_20_250_v2-p0-0)

6.1.3 Random waypoint movement model

To generate random motion movement, with various velocities, a tool called setdest has been used (see appendix for details). This tool generates a random movement model based on the random waypoint algorithm.

This choice has been made since this model is the commonly used model in ad hoc networks [3].

In this stochastic model, every node of the network chooses uniformly a random destination point ("waypoint") in a rectangular deployment region Q. A node moves to this destination with a velocity $v$ chosen uniformly random in the interval $v_{min} \ to \ v_{max}$. When it reaches the destination, it remains static for a predefined pause time $t_{p}$ and then starts moving again according to the same rule [3]. In the random movement model used, the interval $v_{min} \ to \ v_{max}$ is set to a predefined value. (1m/s, 10m/s or 20m/s).

This Random Waypoint Model has been used in the scenarii produced because this pattern is close to the behaviour of a node used in an ad hoc network. For example, if ad hoc networks are used in shopping center or in a conference hall.
6.1 Performance evaluation

6.1.4 Constant Bit Rate (CBR) traffic model

Two sets of scenarios have been used, one using 1 source and 1 destination and another one using 10 sources and 10 destinations. (see appendix for more details)

- The purpose of the first set of scenarios, where source and destination have been fixed (node 0 and node 49), has been defined in the tcl files. A UDP agent is attached to a CBR flow. The aim is to evaluate how a protocol can perform in a single connexion environment, ie. how to deal with the multipath aspect of DSRMPv5 protocol.

- The purpose of the second set of scenarios which is loaded from a file called ”traffic”, is to evaluate how this protocols can deal with multiple connexions, collisions and buffer overflows. This set is an evaluation of the overall performance of DSRMP compared to the performance of what we added only.

A CBR flow over UDP is chosen in order to avoid the slow start TCP mechanism, which would influence the overall results.

A non random flow of 0.1Mbps is used on 1 source 1 destination set of scenarios as previous simulations shows that this bit rate do not overflow the ifq buffer, and therefore the packet drop will only be due to the wireless channel.

A seen on page 475 and page 488, and as seen on some commercial products like the 802.11g Wireless LAN traveler kit http://www.planet.com.tw/product/pdf/C-WAP4050-1.pdf Some common MTU (Maximum Transfer unit) are:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Data packet size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arpanet</td>
<td>1000</td>
</tr>
<tr>
<td>Ethernet (802.3)</td>
<td>from 46 to 1518</td>
</tr>
<tr>
<td>Token Ring (802.5)</td>
<td>4550 (4Mbps) or 18200 (16Mbps and 100Mbps)</td>
</tr>
<tr>
<td>Wifi (802.11b and 802.11b)</td>
<td>1024 (typical at PER &lt; 8%)</td>
</tr>
</tbody>
</table>

Table 6.1: Common packet size in bytes

Therefore the packet size for UDP has been chosen in order to match the Ethernet packet size and therefore set to 1500 bytes as a wireless Ethernet model is simulated.

Using these parameters DSRMPv5 protocol could be evaluated without having any other influence, then the protocol itself.

6.1.5 Routing Metrics

- Packet delivery fraction - Ratio of data packets received by the destinations to the packets sent by the source. (Number of packet receives / number of packet sends) x 100
6.1 Performance evaluation

- Normalized routing load - The number of routing packets transmitted per data packet delivered at the destination. Each hop-wise transmission of a routing packet is counted as one transmission (number of routing packets/ number of packets receives).

- Average end-to-end delay of data packets - The time taken for the packet to reach the destination, it includes queuing at the interface queue, delay during route discovery (ARP) (sum of delay experienced by each packet of the flow)/number of packets).

6.1.6 Simulation environment

Simulation is carried with 50 nodes randomly distributed in a field of 1000 × 1000. A constant bit rate of 0.1Mbytes with a packet size of 1500 Bytes is used. Mobility models for pause time (0 to 100 s) is generated with .tcl scripts in ns2. The simulation time is set to 250 s.
6.2 Analysis of various DSRMP versions

This section deals with the analysis of the results we obtained for various versions of DSRMP.

Version 1 was the first strategy for implementing multipath concept in DSR protocol. It involved the multipath algorithm used in AOMDV. As discussed in previous chapter, the algorithm when implemented in DSR had serious problems to be dealt with (figures 6.1 and 6.2).

- It does not have sufficient uncorrelated paths.
- At high mobility, due to frequent link failures lead to high number of routing overhead.

The problem was partially solved in DSRMP version 2, where we introduced a second cache for intermediate nodes. At high mobility, when a link fails the intermediate node can then use its second cache instead of rebroadcast. Thus reducing the number of routing packets. The concept worked and the results are presented in figure 6.3 and 6.4. By careful observation, we found that not all possible routes were obtained. Some route reply packets were lost. The reason was ARP resolution problem. To cross over this problem version 3 was developed. We introduced a delay in the route reply sequence at the destination node. The result was most of RREP did manage to the source. Major concern with the implementation of this method was to evaluate the effect of additional delay introduced.

It can be observed from figure 6.4 that introducing additional delay did not effect the overall end to end delay. As stated earlier, the first route request is sent back without any delay, and the source node will use that path. Not much difference is observed in the Packet delivery ratio (pdf) either as both version use two cache thus are robust to link failures.

Extension of version 3 was required to optimize the route. In DSRMP version 4 two new variables are introduced in the route reply packet. An intermediate node using tap function listens to its neighbour node and evaluates if the length of the path in its cache
6.2 Analysis of various DSRMP versions

Figure 6.3: Packet delivery ratio (1m/sec 1src-dest)

Figure 6.4: Average end to end delay (1m/sec 1src-dest)

Figure 6.5: Packet delivery ratio (1m/sec 1src-dest)

Figure 6.6: Normalised routing overhead (1m/sec 1src-dest)

to the destination is shorter than that of the other nodes. With this method we were able to obtain optimized route. It included both node and link disjoint paths. Not much difference was observed in the performance of our protocol refer (figure 6.5 and 6.6).

Intentions to bring the normalised routing overhead lower than DSR led to DSRMPv5. At high mobility, nodes experience frequent link failure. To make matter worse the delay introduced to RREP packets in version 2 in order to avoid ARP resolution problem led to additional routing overhead. It was observed that the first Route reply was sent without any delay but until the second route reply makes to the source node the link was broken and the RREP packets did not make it to the source, resulting in a rebroadcast. In DSRMPv5 all RREP packets are dropped when a link broke. This result in considerable reduction of routing packets for DSRMPv5. Figure (figure 6.8 and 6.9 depicts the same.

The figures above (figure 6.10 and 6.11) show the improvement from DSRMP version 1 to version 5. The results show an overall increase of 50 percent in packet delivery ratio at high mobility and a relevant decrease in normalized routing overhead.
6.2 Analysis of various DSRMP versions

Figure 6.7: Average end to end delay (1m/sec 1src-dst pair)

Figure 6.8: Packet delivery ratio (20m/sec 1src-dst)

Figure 6.9: Normalised routing overhead (20m/sec 1src-dst)

Figure 6.10: Improvement from Version 1 to Version 5 (20m/sec 1src-dst)

Figure 6.11: Improvement from Version 1 to Version 5 (20m/sec 1src-dst)
6.3 Assumption used for DSRMPv5

In this part the results obtained with the previously explained parameters will be shown. Results have been organized according to the velocity. First low, then medium and high mobility will be exposed. Three graphics for each speed has been produced to evaluate the overall performance:

To evaluate DSRMPv5 (Dynamic Source Routing Protocol Multi Path version 5), a comparison has been done between reactive ad hoc protocols, using DSR (Dynamic Source Routing Protocol - ETF draft), and AODV (Ad hoc On Demand Distance Vector - RFC 3561). To evaluate these three protocols, scenarii with random mobility models and random traffic models have been used.

A set of scenarii has been produced, a total of 1980 scenarii have been studied. A summary of the number of scenarii done is shown in the table below.

<table>
<thead>
<tr>
<th>Speed</th>
<th>traffic</th>
<th>DSR</th>
<th>DSRMP</th>
<th>AODV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1m/s</td>
<td>1src-1dst</td>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>1m/s</td>
<td>10src-10dst</td>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>10m/s</td>
<td>1src-1dst</td>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>10m/s</td>
<td>10src-10dst</td>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>20m/s</td>
<td>1src-1dst</td>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>20m/s</td>
<td>10src-10dst</td>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
</tbody>
</table>

Table 6.2: Scenarii summary.

Note:
As with AODV, the interface queue model CMUPriQueue produces a "segmentation faults error" therefore the queue type "Queue/DropTail/PriQueue" has been used instead, as seen on FAQs. http://www.isi.edu/nsnam/ns/ns-problems.html
6.3 Assumption used for DSRMPv5

6.3.1 Low mobility (1m/sec)

DSR is known to perform well in low mobility environment, in this set of scenarii close to the velocity of a pedestrian (1m/s or 3.6 km/h), DSR, DSRMPv5 and AODV will be analyzed.

1 source 1 destination continuously sending data

These scenarii are based on 1 source and 1 destination continuously sending data (CBR over UDP) in a slow mobility environment, as explain in appendix.

![Packet Delivery fraction](image1)

![Average end to end Delay](image2)

![Normalized Routing Load](image3)

As seen on figure 6.12, DSRMPv5 in low mobility seems to gives worse results than DSR. DSRMPv5 performs nearly 3 to 5% below DSR. The AODV curve is highly correlated with the DSR curve even if these two protocols have two different purposes and algorithms.
6.3 Assumption used for DSRMPv5

The average end to end delay curves (figure 6.13) are similar for DSR and DSRMPv5 but DSRMPv5 always introduced an additional delay between 10 ms up to a maximum of 350 ms. AODV performs better than DSR and DSRMPv5 with an average end to end delay lower than DSR, spread from 90 ms to 320 ms.

For both AODV and DSR the normalized routing overhead, figure 6.14, is low (0.01 to 0.155) whereas for DSRMPv5 the overhead is between 0.05 to 0.44.

These results show that the mechanisms implicated to enhanced DSR toward DSRMPv5 are not sufficient by themselves to increase DSR’s performance since it lower the results from 3 to 5% in that specific case.
6.3 Assumption used for DSRMPv5

10 sources 10 destinations randomly sending data

These scenarii below are based on 10 sources and 10 destinations randomly sending a CBR traffic over UDP (see appendix) on a low mobility environment.

![Packet Delivery fraction](image1.png)

**Figure 6.15:** Packet Delivery fraction 1m/s - 10src-10dst

![Average end to end Delay](image2.png)

**Figure 6.16:** Average end to end Delay 1m/s - 10src-10dst

![Normalized Routing Load](image3.png)

**Figure 6.17:** Normalized Routing Load 1m/s - 10src-10dst

In this scenarii 10 sources and 10 destinations are used. Data packets are send in a random manner, which is much more realistic than using 1 source and 1 destination continuously sending data. In this scenario, the DSRMPv5’s PDF is 1 or 2% higher then DSR and is higher then AODV’s PDF (8 to 10% higher).

In this topology the end to end delay is quite similar for DSR and DSRMPv5, at some pause time (10 s 70 s and 100 s) DSRMPv5 end to end delay is significantly reduce up to a maximum of 100 ms for a pause of 100 s.

Finally, the normalized routing load is always higher with DSRMPv5 (3 to 4) than with DSR (nearly 2) but not as high as with AODV (over 9).
6.3 Assumption used for DSRMPv5

6.3.2 Medium mobility (10m/sec)

DSR is known to perform well in low mobility environment, but in high mobility DSR is not the best protocol to use. Therefore in order to have a more accurate view of how DSR and DSRMPv5 are performing on various speed, a medium velocity is studied. 10m/s or 36 km/h is close to the speed of a fast bicycle.

1 source 1 destination continuously sending data

The graphics shown below have been produced with 1 source and 1 destination continuously sending data (CBR over UDP) in a medium mobility environment.

In medium mobility motion, DSRMPv5 performs better than DSR in a range of 3 to 5%, but AODV’s PDF (figure 6.18) is always 5 to 10% higher, than DSRMPv5.

One drawback of DSRMPv5, for the 1 source 1 destination case, is the fact that the
6.3 Assumption used for DSRMPv5

average end to en delay (figure 6.19) is always increased compare to DSR to a maximum of nearly 150 ms here. AODV end to end delay stay between 90 ms to 190 ms, whereas DSRMPv5 the end to end delay is between 300 ms to 600 ms.

Due to the mechanism added to DSR to obtain multipath, DSRMP routing overhead (figure 6.20) is higher than DSR and AODV.
6.3 Assumption used for DSRMPv5

10 sources 10 destinations randomly sending data

In a medium mobility environment 10 sources and 10 destinations randomly sending data (CBR over UDP) have been studied, and graphics have been produced as shown below.

![Packet Delivery fraction](image1)

![Average end to end Delay](image2)

![Normalized Routing Load](image3)

In medium mobility motion, DSRMPv5 performs better than DSR. DSRMPv5 is 20 to 25% higher than DSR for PDF (figure 6.21), and outperforms AODV between 12 to 20%.

The average end to end delay (figure 6.22) is always higher both with DSR or DSRMPv5, but DSRMPv5 in medium mobility environment decrease considerably the end to end delay and reduce from a minimum of 100 ms up to a maximum of 800 ms compared to DSR.

Due to the routing used with DSRMPv5 the normalized routing load (figure 6.23) is increased with DSRMPv5 but in medium mobility this routing load is quite similar to AODV’s one.
6.3 Assumption used for DSRMPv5

6.3.3 High mobility (20m/sec)

DSR, which has been designed to be used in low mobility environment, has been simulated in a high mobility environment. In this section the improvement of DSRMPv5 over DSR will be shown, for a high mobility velocity of 20m/s or 72 km/h, which is similar to the speed of a car.

1 source 1 destination continuously sending data

In a high mobility environment, the graphics shown below have been produced with 1 source and 1 destination continuously sending data (CBR over UDP).

Figure 6.24: Packet Delivery fraction 20m/s - 1src-1dst
Figure 6.25: Average end to end Delay 20m/s - 1src-1dst
Figure 6.26: Normalized Routing Load 20m/s - 1src-1dst

DSR is known to perform better in low mobility environment, with 1 source and 1 destination it is clearly shown on graphics that DSRMPv5 improves the packet delivery fraction (figure 6.24). As seen on the medium mobility movement scenarii, the PDF of DSRMPv5 increases gradually from 3 to 5% in medium mobility to 8 to 10% in high
6.3 Assumption used for DSRMPv5

mobility scenarii. Adding multipath to DSR improves the packet delivery ratio in high mobility scenarii.

For Both DSR and DSRMPv5 the average end to end delay (figure 6.25) is highly correlated, but DSRMPv5 always introduces an additional delay up to 350 ms.

As for the average end to end delay, the normalized routing overhead (figure 6.26) is also highly correlated between DSR and DSRMPv5. only on short pause time up to 30 s the routing overhead of DSRMPv5 is up to 4 times higher as DSR one.
6.3 Assumption used for DSRMPv5

10 sources 10 destinations randomly sending data

To produce the graphics shown below 10 sources and 10 destinations randomly sending data (CBR over UDP) have been used.

In high mobility environment the performance of DSRMPv5 is clearly seen. The PDF (figure 6.27) is up to a maximum 30% higher then in DSR, and outperforms AODV by a maximum of 20%.

The average end to end delay (figure 6.28) is also reduced with DSRMPv5 up to a maximum of 2 s. This end to end delay is higher than AODV, but greatly improved compare to DSR.

The normalized routing overhead (figure 6.29) still remains higher with DSRMPv5 than with DSR, but is reduced as the pause time increased.
6.4 Discussion about the results

At low speed DSRMPv5 performs a bit lower (figure 6.12) than DSR, as seen in the packet delivery fraction, and is nearly 3 to 5% lower for one flow. This is due to the fact that with 1 source and 1 destination, all DSR mechanisms like overhearing are not used, and therefore only an evaluation of the algorithm added in DSRMPv5 is done in that very case.

Nevertheless, when the speed increases, DSRMPv5 gets better than DSR, and with multiple flows at 20m/s, DSRMPv5 performs up to a maximum of 30% better than DSR (figure 6.27). This is probably mainly due to the fact that paths are decorrelated and overhearing is used, hence, better results are produced, and this PDF is even higher than AODV.

For the average end to end delay of a constant flow from 1 source to 1 destination it is higher for DSRMPv5, due to the algorithm used in which the protocol search for more specific paths. Indeed, with DSRMPv5, the destination node sends the first reply directly after receiving the request and add a delay between replies for alternate paths. However, for the more realistic case of 10 sources and 10 destinations, the average end to end delay is better for DSRMPv5, as more paths are available and decorrelated.

DSRMPv5 is outperform DSR protocol on the overall, the only drawback is the fact that it increases the routing overhead along with the speed as expected, and slightly more then with normal DSR. This is a trade off we would probably not be able to avoid in most of the cases.

Nonetheless, even with this drawback, the benefit from DSRMPv5 is effective and it is a relevant improvement of DSR.
6.5 Group Mobility

In real scenario, applications such as search and rescue operations require working in groups (units) and efficient communication between other groups should be guaranteed. Such applications require high reliability i.e. efficient routing and minimum control overhead.

Efficient simulation of such scenario is modelled using group mobility. Pursue model is used to evaluate group mobility. The model was developed by Sanchez [11]. In our simulation 50 nodes are divided into 10 groups of 5 modes each. In each group a node is assigned as a target and this target node use waypoint mobility model. The scheme employs all other nodes in the group pursuing the target node with certain velocity. The movement of each node is centered to a point where the target node is supposed to be after a certain interval of time. The model simulates the behaviour of an operation used for tracking people or equipment [18].

The simulation results of DSRMPv5 with pursue model is presented in this section. The simulation environment consists of 50 nodes randomly distributed in a field of 1000 x 1000. There are 10 active source destination pairs with a constant packet sending rate. The packet size is fixed at 1500 bytes. Random traffic connections of CBR is generated with tcl scripts (cbrgen.tcl) available in NS2 simulator. The simulation results are produced for low and high mobility. The total simulation time is set to 250 seconds.

6.5.1 Low mobility (1m/sec)

At low mobility not much difference is seen between DSRMPv5 and DSR. Packet delivery ratio (pdf) is well above 95 percent and from the simulation result it can be concluded that DSR and DSRMP performs well at low mobility. AODV does suffer with large routing overhead due to its routing mechanism. As the algorithm schedules 1 route per destination and does not use cache to store additional paths to the destination. It require periodic update (HELLO PACKETS) at distinct time interval to ensure local connectivity between neighbouring nodes.

![Figure 6.30: Packet delivery ratio (1m/sec 1src-dst)](image)

![Figure 6.31: Normalised routing overhead (1m/sec 1src-dst)](image)
6.5 Group Mobility

6.5.2 High mobility (20m/sec)

It is interesting to analyse the behaviour of DSRMPv5 at high mobility. The simulation environment involves frequent link failures. Compared to AODV and DSR, DSRMPv5 shows distinct improvement in packet delivery ratio. The protocol performs well at higher pause time accounting for 95 percent of packet delivery.

Figure 6.32: Packet delivery ratio (20m/sec 1src-dst)

Figure 6.33: Normalised routing overhead (20m/sec 1src-dst)

Figure 6.34: Average end to end delay (1m/sec 1src-dst)

As explained earlier, AODV suffers from high routing packets due to its periodic update involved in its routing mechanism. It is noted that DSRMPv5 has a higher routing overhead and average end to end delay at zero pause time. This is because of frequent link failure. As explained earlier, Additional delay introduced at the destination node to avoid ARP resolution problem. Here the first route reply packet was sent without any delay but the time when the second packet reaches the source node the link may be broken and it may require another broadcast. In DSRMPv5, sufficient time is required to establish multiple connectivity between source and destination.
6.6 Discussion about the results

The simulation results prove that DSRMPv5 has certainly improved the performance of DSR in terms of packet delivery ratio. It is understood that implementing multipath in DSR requires additional routing packets. The results show us that though DSRMPv5 shows an increase in normalised routing overhead. It is reasonable low when compared to AODV.
Chapter 7

Conclusion and Future Work

Simulations have shown that DSRMP protocol can be considered as reliable. The routing metric packet delivery fraction stays above 80%, even in high mobility case. Besides, the average end to end delay is improved on the overall: significantly along with multiple flows scenarii and increased mobility speed, and at least highly similar to DSR otherwise. To obtain those results, the routing overhead has been slightly increased; this is the trade off for optimization. These additonal routing packets allow to find several paths from the very beginning. The combination of advantages of a source routing protocol and the multipath concept leads to better results in realistic scenario such as the group mobility model.

Characteristics and results for DSRMP were achieved after an extensive design part in which other algorithm has been tested and changes step by step have been done. Indeed, it is that gradual validation of the algorithm and its implementation that permitted to reduce routing overhead. Design has been a key part to get reliable results.

This project, achieved for the ninth semester in Mobile Communication programme of Aalborg University, allowed us to become familiar with the Network Simulator 2 which is an important tool in Network Engineering field. It also gave us the possibility to analyse Ad Hoc Networks and some of their protocols.

This study could be continued by, for instance, developing the multipath aspect of our protocol. It could be achieved by splitting data packets from the source to the destination; the whole message would not be transmitted by the same path, nor the same nodes all the time. Another solution could be to enforce reliability adding some redundancy code; in that case, it would allowed not to send again packets in case one link breaks.
Appendix A

Tcl file

A.1 Aim of the .tcl file

Ns-2 uses a type of file called ".tcl" files. This file is used to generate and define a simulation. In this file all parameters needs to define a type of connexion (wire or wireless, type of queue, type of antenna) could be modified. As a large amount of options are available, and could not be evaluated, most of these options have been left as default, according to the parameters described in the ns-2 user guide.

The file below is based on the file available on the following link.

A.2 The tcl file

Below is the complete TCL file used for a 1 source 1 destination scenario. The 10 sources and 10 scenario is loaded from a file and therefore the definition of the flow is commented, and the 10 scenario flow is loaded from the file defined by the "sc" value.

# wrls-dsr-nDSSRmodel.tcl
# Stephane DAHLEN 3 november 2004
# revisions:
#2 December 2004: modification on the TCL file (remove the folder "model") to simplify multiple scenarii.
#5 November 2004: wrls-dsr-nUDPmodel version 0.11: adapt the TCL file for rectangular simulations
#5 November 2004: wrls-dsr-nUDPmodel version 0.1: Enhancement of the code (remove duplicates fields) and adjust parameters (UDP packet size, bitrate, queue type...)
#7 November 2004: wrls-dsr-nUDPmodel version 0.0: first random version with 50 random moving nodes

# Define options (use default values)
set val(chan) Channel/WirelessChannel ;# channel typ
set val(prop) Propagation/TwoRayGround ;# radio-propagation model (cf pg150 & 186)
set val(netif) Phy/WirelessPhy ;# network interface type
#set val(ifq) Queue/DropTail/PriQueue ;# interface queue type (cf pg 77)
set val(ifq) CMUPriQueue ;# only for DSR, to avoid "segmentation fault"
set val(ll) LL ;# link layer type
set val(ant) Antenna/OmniAntenna ;# antenna model (cf pg 150)
set val(ifglen) 50 ;# max packet in ifq
set val(nn) 50 ;# number of mobilnodes
A.2 The tcl file

```tcl
set val(rp) DSR ;# routing protocol
# for squared simulation
set val(x) 1000 ;# X dimension of topography
set val(y) 1000 ;# Y dimension of topography
# for rectangular simulation
#set val(x) 405 ;# X dimension of topography
#set val(y) 2469 ;# Y dimension of topography
# define random movement file
set val(cp) "sq_randomMOV_1_250x2.5v2-p0.0" ;# define random traffic model
# not currently used
set val(stop) 250 ;# simulation end time

# ———— Main program ————
# create a simulator object (cf pg 41)
set ns [new Simulator]
# create a topography object (cf pg 147)
set topo [new Topography]
# set up topography object
$topo load flatgrid $val(x) $val(y)

# Setup trace File
set tracefd [open simple.tr w] # create a .tr file
set namtrace [open out.nam w] # create a .nam file
# use new trace format (cf pg 157)
$ns use-newtrace
$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# Create Object God
set god [create-god $val(nn)]
# use parameter given under
# nesl.ee.ucla.edu/courses/ee206a/2002s/guest_presentations/GP02_Park
# Create $val(nn) mobilenodes [val(nn)] and attach them to the channel.
set chan_1 [new $val(chan)]
# configure the nodes (cf pg 144)
$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)
-channel $chan_1
-topoInstance $stop
-agentTrace ON
-routerTrace ON
-macTrace OFF
-movementTrace ON

# Create the specified number of node and "attach" them to the channel
for set i 0 $i $val(nn) incr i
set node_i($i) [$ns node]
# Define node movement model
puts "Loading connection pattern..."
source $val(cp)
# Define traffic model
# puts "Loading scenario file..."
source $val(sc)
```

A.2 The tcl file

# Define different colors for data flows
$ns_ color 0 Red

# Define node name:
$node_(29) label "Source"
$node_(13) label "Destination"

# Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$udp0 set packetSize_ 1500 #(default 1000 cf pg272)
$udp0 set class_ 1
$ns_ attach-agent $node_(29) $udp0

# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set interval_ 0.2 # not used
$cbr0 set rate_ 0.1Mb
$cbr0 set random_ false # send data continuously
$cbr0 attach-agent $udp0

# Create a Null agent (a traffic sink) and attach it to node n7
set null0 [new Agent/Null]
$ns_ attach-agent $node_(13) $null0

# Connect the traffic sources with the traffic sink
$ns_ connect $udp0 $null0

# Start sending data
$ns_ at 0.1 "$cbr0 start"

# Tell nodes when the simulation ends
for set i 0 $i ! $val(nn) incr i
$ns_ at $val(stop) "$node_(i) reset"
$ns_ at $val(stop)+0.0001 "stop"
$ns_ at $val(stop)+0.0002 "puts NS_ EXITING...; $ns_ halt"
proc stop
    global ns_ tracefd
    close $tracefd
    exec nam out.nam & # automatically run nam to visualized the result
# Start the scheduler (cf pg 41)
$ns_ run

Note:

- In order to see what is happening on the mac layer the option -macTrace OFF could be set to ON. For our large scale simulation this option has been left off as it increase consequently the .tr file.

- About the GOD object:
Directives for GOD are present as well in node-movement file. The General Operations Director (GOD) object is used to store global information about the state of the environment, network, or nodes that an omniscient observer would have, but that should not be made known to any participant in the simulation.

Currently, the god object is used only to store an array of the shortest number of hops required to reach from one node to an other. The god object does not calculate this on the fly during simulation runs, since it can be quite time consuming. The information is loaded into the god object from the movement pattern file where lines
A.2 The tcl file

of the form
$ns_\text{ at } 899.642 \text{ } \$\text{god}_\text{ set-dist 23 46 2}$
are used to load the god object with the knowledge that the shortest path between node 23 and node 46 changed to 2 hops at time 899.642.

From http://www.isi.edu/nsnam/ns/tutorial/nsscript7.html
Appendix B

Traffic scenarii

Two traffics scenarii has been defined. One using 1 source and 1 destination and another one using 10 sources and 10 destinations.

The purpose of the first set of scenarii is to evaluate how a protocol can performs in a single connexion environment, ie how to deals with multipath aspect of DSRMP protocol.

The purpose of the second set of scenarii is to evaluate how this protocols can deals with multiple connexion, and how a protocols can deals with collision and buffer overflows.

B.1 1 source and 1 destination

The original 1 source and 1 destination scenario was defined inside the tcl file, with parameters similar as state on page 272 of the ns-2 manual.

Node 0 was defined as the source node and node 49 as the destination node.

A UDP agent has been attach to these two nodes, and the packet size has been increased to 1500 octets to match the maximum size of an ethernet packet.

A CBR agent is attached on the top of the UDP agent with a rate of 0.1Mb, to avoid packet dropped due to buffer queue overflow, as discovers with premilinari set of scenarii. The random option has been disabled which means that CBR packets will be send in a regular manner.

Below is the code used to define these parameters

```tcl
# define node name:
$node_(0) label "Source"
$node_(49) label "Destination"

# Create a UDP agent and attach it to node n0
create UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$udp0 .set packetSize 1500
$udp0 .set class 1
$ns .attach-agent $node_(0) $udp0

# Create a CBR traffic source and attach it to udp0
create CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 .set rate 0.1Mb
$cbr0 .set random false
```
B.2 10 sources 10 destinations version

```tcl
$cb0 attach-agent $udp0

# Create a Null agent (a traffic sink) and attach it to node 49
set null0 [new Agent/Null]
$ns _attach-agent $node(49) $null0

# Connect the traffic sources with the traffic sink
$ns _connect $udp0 $null0

# Start sending data
$ns _at 0.1 "$cb0 start"
```

Figure B.1: 1 source 1 destination scenario.

B.2 10 sources 10 destinations version

This set of scenarios has been generated using the `cbrgen.tcl` file located under

```
[...]/ns-allinone-2.27/ns-2.27/indep-utils/cmu-scen-gen
```

The usage is: cbrgen.tcl [-type cbr—tcp] [-nn nodes] [-seed seed] [-mc connections] [-rate rate]
the traffic file has been generated using the following command:

```
ns cbrgen.tcl -type cbr -nn 10 -mc 10 -rate 0.1 >> traffic
```

All options have been set similar to the first scenario. The only change on this scenario is that the traffic is random.
B.2 10 sources 10 destinations version

The packet size has been changed inside the traffic file and set to 1500 octets as for the 1 source and 1 destination scenarii.

Figure B.2: 10 sources 10 destinations scenario.

below is a part of the generated traffic file:

```
# nodes: 50, max conn: 10, send rate: 10.0, seed: 1
#
# 1 connecting to 2 at time 2.5568388786897245
#
$node(1) label "Src 1"
$node(2) label "Dst 1"
set udp(0) [new Agent/UDP]
$ns_attach-agent $node(1) $udp(0)
set null(0) [new Agent/Null]
$ns_attach-agent $null(0) $udp(0)
set cbr(0) [new Application/Traffic/CBR]
cbr(0) set packetSize 1500
cbr(0) set interval 10.0
cbr(0) set random 1
cbr(0) set maxpkts 10000
cbr(0) attach-agent $udp(0)
$ns_connect $udp(0) $null(0)
$ns_at 2.5568388786897245 "$cbr(0) start"
[..]
#
# 11 connecting to 12 at time 62.77338456491632
#
$node(11) label "Src 11 Dst 10"
$node(12) label "Dst 11"
set udp(9) [new Agent/UDP]
$ns_attach-agent $node(11) $udp(9)
set null(9) [new Agent/Null]
```

Project report - Alexandre Boursier - Stéphane Dahlén - Julien Marie-Françoise - Thior Santander-Marin - Shikar Nethi
B.2 10 sources 10 destinations version

```bash
$ns_ attach-agent $node_(12) $null_(9)
set cbr_(9) [new Application/Traffic/CBR]
$cbr_(9) set packetSize_ 1500
$cbr_(9) set interval_ 10.0
$cbr_(9) set random_ 1
$cbr_(9) set maxpkts_ 10000
$cbr_(9) attach-agent $udp_(9)
$ns_ at 62.7738456491638 "$cbr_(9) start"
#
#Total sources/connections: 7/10
#```

#Project report - Alexandre Boursier - Stéphane Dahlen - Julien Marie-Françoise - Thior Santander-Marin - Shekar Nethi#
Random model file for scenarii

C.1 Overview and usage

Three speed grade have been used in the scenarii, and speed from low speed (1m/s), medium speed (10m/s), and high speed (20m/s) have been generated.

As each pause time needs 10 sets of scenarii 3 sets of 110 files have been generated.

Two versions of the tool to generate random motion movement under ns-2.27 are available version 1 and version 2. For both version this tool called setdest needs previously to be compiled before being used. As state on http://www.isi.edu/nsnam/ns/tutorial/nsscript7.html#second setdest generates movement pattern using the random waypoint algorithm.

![The Random Waypoint Algorithm](http://keskus.hut.fi/~esa/java/rwp/index.shtml)

**Figure C.1: The Random Waypoint Algorithm**

This animation is from [http://keskus.hut.fi/~esa/java/rwp/index.shtml](http://keskus.hut.fi/~esa/java/rwp/index.shtml)
C.2 File generation & name structure

Setdest tools could be found in the directory
[../ns-allinone-2.27/ns-2.27/indep-utils/cmu-scen-gen/setdest

Below is the available option for each one:

<original 1999 CMU version (version 1)>
./setdest -v <1> -n <nodes> -p <pause time> -M <max speed> -t <simulation time> -x <max X> -y <max Y>

OR
<modified 2003 U.Michigan version (version 2)>
./setdest -v <2> -n <nodes> -s <speed type> -m <min speed> -M <max speed> -t <simulation time> -P <pause type>
-p <pause time> -x <max X> -y <max Y>

Speed Selections
1. uniform speed [min, max]
2. normal speed [min, max]

Pause Selections
1. constant
2. uniform [0, 2 x pause]

As seen up setdest version 1 do offers a pause time option , but as seen on simulation
this pause time is not between the motion of each node but is an offset from the beginning
of the simulation, and therefore cause a error ”node(x) undefined” under nam.

C.2 File generation & name structure

To avoid this problem under nam and to have a pause time between the motion of each
nodes, version 2 of setdest was used instead. Below are the parameters used to define the
random scenarii:

./setdest -v 1 -n 50 -s 1 -m 10 -M 10 -t 250 -P 1 -p 50 -x 1000 -y 1000 >>
sq_randomMOV_10_250s_v2-p50-3

The data produced by setdest has been redirected to a file, and each random model
file has an unique name therefore 3 sets of files for each speed.

A random motion file has the following format
i.e. sq_randomMOV_10_250s_v2-p50-3

- sq_randomMov is common for each file and means that this file is the random motion
model for a squared area.
- _10_ (values 1m/s, 10m/s, and 20m/s) means that this file is a 10m/s file.
- 250 means that this file is a 250 s file version.
- v2 means that setdest v2 has been used to produce this file
- p50 (values from 0 to 100 respectively a pause from 0 s to 100 s) means a pause of
50 s between each motion of a node is used in this file.
- -3 (values from 0 to 9) is the reference number of the current file.

See scripts for the generation of these files.
C.3 Scenarii organization

110 scenarii files are generated for each set of parameters (speed and number of connections) therefore a strict organization is needed to avoid lost of data and lost of scenarii. Below is a sketch on how all these files are organized in order to match scripts and parser as explain on next pages.

![Diagram](image.png)

**Figure C.2:** Scenarii organization
Java parser

D.1 Purpose of ns-2 files

When a simulation is produced with ns-2 and as defines in the TCL file, two files are produced:

- A nam file (.nam): which is a graphical visualization of what is happening during the simulation.
- A trace file (.tr): which is a file where all send and received packets by all nodes are written.

In order to evaluate a protocol, graphically it is possible to see how a protocol is performing on various small range of large range simulations, but in order to evaluate more deeply, and accurately if a protocol is performing well, it is better to use the trace file, to extract data and to do some calculations. Some calculation could be done in order to evaluate the amount of routing packets or in order to know the throughput.

In order to extract data from a trace file, a file called parsetrace has been used. This file is from http://www.cs.utk.edu/~gupta/Adhoc.doc only small modifications has been done on this file, ie to add a new protocol name (DSRM & DSRMP).

In order to build this tool, JDK (Java Developer Kit) is needed in order to have the javac, java compiler.

To compile the java parser the following command should be done:

```
javac parsetrace.java
```

This will produce a java class called parsetrace.class this is the compiled version of the parsetrace file.

When this tool is build and if all trace files have the same name (ie in the .TCL file all .tr files are called simple.tr) it is not necessary to build this tool again, and could be run using JRE (Java Runtime Environment).
D.2 Java parser listing

To extract data from a file called simple.tr the following command should be done:

```java
java parsetrace
```

This will take the available simple.tr file and extract data and produced the following results:

- Number of packets sends.
- Number of packets receives.
- Number of routing overhead packets
- Normalized routing load (routing packets/number of packets receives).
- Packet delivery fraction (number of packet receives/number of packets sends)*100.
- Average End to End delay (sum of delay experienced by each packet of the flow)/number of packets.

D.2 Java parser listing

Below is the listing of the modified version of the java file used for the simulations:

```java
import java.util.*;
import java.lang.*;
import java.io.*;
public class parsetrace {
    public static void main (String args[]) {
        String s, thisLine, currLine, thisLine1;
        int j=0;
        FileInputStream fin, fin1;
        FileOutputStream fout, fout1;
        final int FILES = 45;
    }final int MAX_PACKETS = 1000000;
    try {
        int i=0, sends=0, receives=0;
        int drops=0, packet_id=0, highest_packet_id = 0;
        int line_count=0, current_line=0, routing_packets=0;
        int count=0;
        float pdfraction, time=0, packet_duration=0, end_to_end_delay=0;
        float avg_end_to_end_delay=0;
        float start_time[] = new float[MAX_PACKETS];
        float end_time[] = new float[MAX_PACKETS];
        float sent_packets[] = new float[MAX_PACKETS];
        float received_packets[] = new float[MAX_PACKETS];
        String tokens[] = new String[100];
        // initialize the start time
        for (i=0; i<MAX_PACKETS; i++)
            start_time[i] = 0;
        fout = new FileOutputStream ("traceoutput.txt");
        DataOutputStream op = new DataOutputStream(fout);
        for (int h=0; h<FILES; h++) // for (int h=0; h<FILES+1; h++)
        {
            // // initialize the start time
            for (i=0; i<MAX_PACKETS; i++)
                start_time[i] = 0;
            fout = new FileOutputStream ("traceoutput.txt");
            DataOutputStream op = new DataOutputStream(fout);
            for (int h=0; h<FILES; h++) // for (int h=0; h<FILES+1; h++)
```
D.2 Java parser listing

```java
j = 0;
sends = 0; receives = 0; routing_packets = 0;
highest_packet_id = 0;
end_to_end_delay = 0;

for (i = 0; i < MAX_PACKETS; i++)
{ start_time[i] = 0; end_time[i] = 0; }

fin = new FileInputStream("simple.tr");
DataInputStream br = new DataInputStream(fin);

while ((thisLine = br.readLine()) != null ) {
    // scan it line by line
    java.util.StringTokenizer st = new java.util.StringTokenizer(thisLine, " ");
    i = 0;
    while (st.hasMoreElements())
        tokens[i++] = st.nextToken();

    if (tokens[0].equals("s") || tokens[0].equals("r") || tokens[0].equals("f"))
    {
        // parse the time
        if (tokens[1].equals("-t")) time = Float.valueOf(tokens[2]).floatValue();

        // parse the packet id
        if (tokens[39].equals("-Ii")) packet_id = Integer.valueOf(tokens[40]).intValue();

        // calculate the sent packets
        if (tokens[0].equals("s") && tokens[18].equals("AGT") && tokens[34].equals("cbr"))
            sends++;

        // find the number of packets in the simulation
        if (packet_id > highest_packet_id) highest_packet_id = packet_id;

        // set the start time, only if its not already set
        if (start_time[packet_id] == 0) start_time[packet_id] = time;

        // calculate the receives and end-end delay
        if (tokens[0].equals("r") && tokens[18].equals("AGT") && tokens[34].equals("cbr"))
            { receives++; end_time[packet_id] = time; }
    }

    else end_time[packet_id] = -1;

    // calculate the routing packets
    if (tokens[0].equals("s") || tokens[0].equals("f") || tokens[18].equals("RTR")
        || tokens[34].equals("AODV") || tokens[34].equals("DSR") || tokens[34].equals("DSRM")
        || tokens[34].equals("DSRMP") || tokens[34].equals("message"))
        routing_packets++;

} // calculate the packet duration for all the packets for (packet_id = 0; packet_id <= highest_packet_id ; packet_id++)
{ packet_duration = end_time[packet_id] - start_time[packet_id];
    if (packet_duration > 0) end_to_end_delay += packet_duration;
}

// calculate the average end-end packet delay
avg_end_to_end_delay = end_to_end_delay / (receives);

// calculate the packet delivery fraction
pdfraction = (float)receives / (float) (sends) * 100;

// System.out.println(" \n sends "+sends);
// System.out.println(" receives "+receives);
// System.out.println(" routing overhead (packets) "+ routing_packets);
```
//System.out.println(" Normalized routing load "+(float)routing_packets/(float)receives);
//System.out.println(" pdfraction "+pdfraction);
//System.out.println(" Avg End-End delay "+avg_end_to_end_delay);

System.out.print(" "+sends);
System.out.print(" "+receives);
System.out.print(" "+routing_packets);
System.out.print(" "+(float)routing_packets/(float)receives);
System.out.print(" "+pdfraction);
System.out.print(" "+avg_end_to_end_delay);
System.out.println(" ");

op.writeBytes(" "+sends);
op.writeBytes(" "+receives);
op.writeBytes(" "+routing_packets);
op.writeBytes(" "+(float)routing_packets/(float)receives);
op.writeBytes(" "+pdfraction);
op.writeBytes(" "+avg_end_to_end_delay);
op.writeChar(\n);
}
}
catch (Exception e) {
    e.printStackTrace();
    }
}
Appendix E

Bash scripts

A total of 1980 (18 set of 110 files) have been generated in order to evaluate DSRMPv5.

In order to avoid a wrong human manipulation scripts have been generated, in bash and in perl which reduce the overall human impact on simulations.

More than 10 scripts were generated and all could not be explained (ie to patch a file from DSR to DSRMP or AODV, to remove fields of 1 source 1 destination definition to a loaded file from 10 source to 10 destination, to concatenate all results, to modify the random model file from 1m/s, 10m/s or 20m/s, to run all 110 simulation in a run, to generate a set of random model files, to copy a set of random model files, to remove all unnecessary files...)

Below 3 scripts will be shown:

- **gen.sh**: which is simple script used to generate 110 scenarii for one set of speed.

- **runall.sh**: which purpose is to run an ns-2 simulation, parses the data and removes all nam file, trace files and java files, and keeps only a file called throughput.txt (generated by the java parser), which is a summary of the trace file, with the number of packets sends and receives, the number of routing overhead packets, and the calculation of the packet delivery fraction (throughput), and the average end to end delay.

- **1src-2-10src.sh**: is the script use to parse a .tcl file and to patch it from a 1 source - 1 destination version to a 10 sources - 10 destinations version.

## E.1 gen.sh

This script is a simple 2 nested loop using a for loop.

```bash
#!/bin/bash
clear
echo "Script done in order to automatism the generation of a set of scenarii"
for pause in 'seq 0 10';
do
  p1=`expr $pause 10`
```
E.2 runall.sh

```
echo "============ generation of pause $p1 ================

echo " value of Pause: $p1"
echo "pause time= $pause"
for i in \`seq 0 9`;
do
echo "Simulation number: $i"

./setdest -v 2 -n 50 -s 10 -M 10 -t 250 -P 1 -p $p1 -x 1000 -y 1000
> sq_ranrandomMOV_10_250s_x2-p$p1-$i

./setdest -v 2 -n 50 -s 1 -m 10 -M 10 -t 250 -P 1 -p $p1 -x 1000 -y 1000
> sq_ranrandomMOV_10_250s_x2-p$p1-$i
done
done
echo "All simulation done"
```

E.2 runall.sh

This script is also a nested loop but as the first folder is not a pure digit it is necessary to change all folder name before starting the first loop. This folder are renamed at the end of the two loops. Inside this script the trace file produced by ns-2 is parsed by a java file which is called inside this script.
This script has been used to patch original .tcl files, where a connexion is defined inside the .tcl file to a .tcl file which is load from an outside file. To achieve this perl has been used.

The nested loop is similar to the one shown in runall.sh. Only a part of the patch process is shown below.

```plaintext
... 
# g is for global 
echo "———–file $i done...———-" 
...
```

Perl search on a text file from a first string in forward slash (/) and than change it by the second one.
Matlab fonction to plot the results

F.1 Aim of the Matlab script

A Matlab script is used to plot the data generated by simulations. The aim of this Matlab script is to read data from the text files and then to plot them. For each protocol and each set of simulation (110 simulations), 11 files are produced, corresponding to each pause time step (from 0 s to 100 s). On each file i.e. allpause10.txt, 10 results are concatenated inside this file. Matlab read this 110 results and calculate the mean, and then plot the result.

F.2 The Matlab script

Below is the script used to generate graphics.

Note:
Some redundant part have be removed.

```matlab
% (c) Stéphane DAHLEN 6 December 2004
% review 12 December (legend)
% global values
clear
close all

in_dir0 = 'dsr\_m_per_s\10src-10dst\';
in_dir1 = 'dsrmp\_m_per_s\10src-10dst\';
in_dir2 = 'aodv\_m_per_s\10src-10dst\';
in_file='allpause';
no_file=10;

% create matrix for x = zeros(1,11); % built a matrix of 1 line and 11 values for X axis
```
The Matlab script

```matlab
sends = zeros(1,11); % matrix for the send values
receives = zeros(1,11); % matrix for the received values
routing = zeros(1,11); % matrix for the routing overhead (packets) values
nrl = zeros(1,11); % matrix for the Normalized routing load values
pdf = zeros(1,11); % matrix for the pdfraction values
avgend2end = zeros(1,11); % matrix for the Avg End-End delay values
troughtput = zeros(1,11); % matrix for the troughtput delay values

%Loop for DSR
% loop to generate file name
for n = 0:no
    y=n*10; %multiply by 10 for the step and the name
    str = num2str(y); % convert a number to a string
    file=strcat(in_dir0,in_file,str,'.txt') % concatenate the string
    x(n+1) = y; % x+1 because matrix starts at 1 and not 0
    %create x axis
    % open file
    fid = fopen(file,'rt'); % 'rt' means "read text"
    if (fid ¡ 0)
        error('could not open file file');
    end;
    % read from file
    tab = fscanf(fid,'%f
'); % read float values
    fclose(fid);
    %process data
    B = transpose(tab); % put columns in line
    res = reshape(B,6,10); % change the 1 by 60 matrix to a 6 by 10 matrix
    res1 = transpose (res); % put columns in line
    final= mean (res1); % calculate the of all columns
    %keeping data on matrix to plot it
    sendsDSR(n+1) =final(1);
    receivesDSR(n+1) =final(2);
    routingDSR (n+1) =final(3);
    nrlDSR(n+1) =final(4);
    pdfDSR(n+1) =final(5);
    avgend2endDSR (n+1) =final(6);
end

%Loop for DSRMP

%Loop for AODV

% plot all graphs
if (sends_g==1)
    figure;
    hold on % to put 2 graphics on on
    grid on
    plot (x,sendsDSR,'r+:','LineWidth',2)
    plot (x,sendsDSRMP,'b+-','LineWidth',2)
    plot (x,sendsAODV,'g+-.','LineWidth',2)
    legend('DSR','DSRMP','AODV',0); % 0 means any available space
    hold off
    title ('Number of Packet Sends (1m/s 10src-10dst)');
    xlabel ('Pause time (sec)');
    ylabel ('Number of Packet Sends');
    if (record==1)
        print -dpng send.png
    end;
end;
```

Project report - Alexandre Bourquier - Stéphane Dahlen - Julien Marie-Françoise - Thior Santander-Marin - Shekar Nethi

112
Appendix G

CD

Content of this CD:

- all scenarii used to produced graphics about DSRMPv5 evaluation could be found.
- Java parser
- raw results of DSRMPv5
- Matlab code to produced the graphics
- Patch for ns-2 for DSRMP v1 to v5


5. RFC 3561 (AODV) Ad hoc On Demand Distance Vector.  
   http://www.ietf.org/rfc/rfc3561.txt

Retrieve on 29th September 2004 from the World Wide Web: 

Hoc Networks: Issues and Challenges, Wireless Communication Technologies Group, 
provide a set of typical problems to face in ad-hoc networks. Retrieve on 23rd September 
2004 from the World Wide Web: 

[9] George Aggelou, (no date) Publications on Mobile Ad Hoc Networks (MANETs) 
(IEEE), On-line copies of the journal, conference papers and theses on Mobile Ad Hoc 
Networks (MANETs). Retrieve on 15th September 2004 from the World Wide Web: 
http://www.ee.surrey.ac.uk/Personal/G.Aggelou/MANET_PUBLICATIONS.html

Elizabeth Royer and C-K Toh.  

http://www.ieee-infocom.org/2003/papers/07_01.PDF

[10] Tony Larsson, Nicklas Hedman, (1998); Routing protocol in wireless ad hoc networks- 
a simulation study, LULE Tekniska Universitet This document is a master thesis 


[12] Liang Qin, Thomas Kunz, (2004); Survey on mobile ad hoc network routing protocols 
and cross-layer design, Carleton University This document is a Technical report published 
in August 2004. Retrieve on 20th September 2004 from the World Wide Web: 

[13] Wong, K.D. Kwon, T.J. Varma, (2004); Towards commercialization of ad hoc net-
works, Networking, Sensing and Control, IEEE International Conference, Volume: 2,
BIBLIOGRAPHY


