Department of Computer Science

Guillermo Alonso Pequeño
Javier Rocha Rivera

Extension to MAC 802.11 for performance improvement in MANET

Computer Science
D-level thesis (20p)

Date: 061221
Supervisor: Andreas Kassler
Examiner: Kerstin Andersson
Serial Number: D2007:06

Karlstads universitet 651 88 Karlstad
Tfn 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se

Computer Science

Guillermo Alonso Pequeño

Javier Rocha Rivera

Extension to MAC 802.11 for performance improvement in MANET

Master's Project

2007:06

Extension to MAC 802.11 for performance improvement in MANET

Guillermo Alonso Pequeño

Javier Rocha Rivera

This report is submitted in partial fulfilment of the requirements for the Master's degree in Computer Science. All material in this report which is not our own work has been identified and no material is included for which a degree has previously been conferred.

Guillermo Alonso Pequeño

Javier Rocha Rivera

Approved, 2006-12-21

Supervisor: Andreas J. Kassler

Examiner: Kerstin Andersson

Abstract

In the last few years, the exploit of ad hoc wireless networks has increased thanks to their commercial and military potential. An application of wireless ad hoc networks is Bluetooth technology, which allows wireless communication among different devices. As a military application, we can report the establishment of communications between groups of soldiers in a not safe territory. Additionally, ad hoc networks are useful in emergency operations, where no fixed infrastructure is feasible.

A mobile ad hoc network (MANET) represents a system of wireless mobile nodes that can self-organize freely and dynamically into arbitrary and temporary network topology. On one hand, they can be quick deployed anywhere at anytime as they eliminate the complexity of infrastructure setup. On the other hand, other problems arise, such as route errors or higher overhead, caused by the mobility of nodes.

The main goal of this master's thesis has been the improvement of the communication between MAC 802.11 protocol and DSR (Dynamic Source Routing) protocol, to run in the ns-2 network simulator.

Contents

List of Figures

List of tables

1 Introduction

1.1 Overview

In the 1970's and the 1980's, computer networks were considered an infrastructure of fixed form. In the latest years, the proliferation of mobile computing devices, such as laptops, personal digital assistance (PDA's), or mobile phones, has led to a revolutionary change in the computer world. To communicate all those devices, a wired network is not feasible, since it has no mobility. Thereby, a new technology, wireless networks, is needed [35]. Wireless networks use electromagnetic radio waves for exchanging data. However, in the last years, a demand of mobility by users is increasing; hence, a special kind of networks is needed: wireless mobile ad-hoc networks.

In wireless ad-hoc networks, the nodes themselves form the network, and they do not need fixed infrastructure, therefore each node executes routing functionalities, such as forwarding network traffic. Before designing an ad hoc wireless, we should consider different features, such as the use of MAC protocol, routing protocol, transport layer protocol, quality of service, or support of security.

To work properly, the different protocols in wireless ad-hoc networks must handle different issues, such as noise of the network, routing information, transmission ranges, etc. Sometimes in one node, only part of the information collected by one protocol is delivered to another protocol and a misinterpretation among these protocols may happen. To deal with this, we propose a modification in the MAC 802.11 protocol to avoid launching unnecessary operations in the DSR (Dynamic Source Routing) protocol., achieving better performance in the network, i.e. less routing overhead, less routing changes, less collision of packets, less route errors, less mac errors, and more throughput.

Concretely, DSR protocol launches route error when a neighbouring node is still near, because it understands the information received from the MAC layer as a broken link. Usually, the interferences among radio ranges of nodes could lead to this misunderstanding. The proposed approach tracks the signal strength of each node, informing the routing layer that the node has enough signal strength, skipping the route error launched by DSR.

1.2 Proposed goals and accomplishments

The proposed goals of this master's thesis are:

- To detect in MAC 802.11 [48] the presence of neighbour nodes for each node within its reachable radio transmission, using ns-2 network simulator [11]. This is achieved by tracking the signal strength of neighbouring nodes, that is, storing the signal strength of neighbouring nodes for each transmission they make.
- To inform the upper layer, routing protocol DSR [18] in this case, when a transmission was not successful, and if a neighbouring node is in the transmission range of each other.
- To adapt routing protocol DSR using this information, detecting error links and avoiding unnecessary route maintenance processes.
- To compare the achieved results with previous values, performing simulations in several (static and mobility) scenarios, and for different type of traffic (cbr,ftp,..)

The main goals were achieved successfully. The received power of each neighbour node at MAC layer was stored away in every node and used later to inform the routing protocol when a transmission among nodes was unsuccessful. DSR protocol may conclude mistakenly that there was an error link and triggers a "route maintenance" process upon receiving the information about broken link from the MAC layer. In our approach, DSR protocol is able to distinguish if either link errors in MAC 802.11 are due to interferences among radio ranges of the nodes, or neighbouring nodes are moving away. In other words, if packet retransmission is not possible at MAC layer because of the interferences among radio ranges of the nodes, our approach detects if the node is still present, nearby enough to transmit. If so, to send a route error packet is skipped and not triggered. Hence, routing overhead is decreased and the overall performance in the network is increased.

1.3 Document overview

The rest of the documentation is outlined as follows.

- Chapter 2 introduces the background about mobile ad-hoc networks, MAC layer operation, routing operation, and transport layer.
- Chapter 2.1 introduces mobile ad-hoc networks in general, describing its operation and applications from a practical outlook.

- Chapter 2.2 introduces MAC layer in mobile ad-hoc networks, describing particularly 802.11 protocols, which are used in the current version of network simulator.
- Chapter 2.3 describes the routing operation in mobile ad-hoc networks; concretely it analyzes the performance of DSR protocol.
- Chapter 2.4 gives an overview about the transport layer protocol, and describes TCP and UDP protocols in wireless networks.
- Chapter 2.5 describes previous work in network simulator as base of this project.
- Chapter 3 introduces the ns-2 network simulator. It provides a description of several functionalities and examples about node configuration, trace files, agents, and scripts. Afterwards, the rest of the chapters is outlined as follows.
- Chapter 3.4 gives an overview of the MAC protocol in the ns-2 network simulator, and describes different features and operations of these protocols from a language-programmed point of view.
- Chapter 3.5 describes DSR protocol in ns-2 network simulator from a language-programmed point of view.
- Chapter 3.6 introduces the different radio propagation models that can be set in ns-2 network simulator
- Chapter 4 describes a cross layer design for MANET. This chapter focuses on the importance of interactions between the layers from a theoretical point of view. In addition, it describes the interaction layer of MAC 802.11 protocol and DSR protocol in ns-2 network simulator.
- Chapter 5 shows and explains the simulation results using ns-2 network simulator.

2 Background

2.1 Mobile Ad Hoc Networks

2.1.1 Introduction

A Mobile ad-hoc wireless network (MANET) is a system of wireless mobile nodes that dynamically self-organize in arbitrary and temporary network topologies [16]. Ad-hoc wireless networks can be located in networks that use multi-hop radio relaying and may operate without any support of fixed infrastructure. As multi-hop, we refer to routes between nodes that may contain multiple hops. In mobile ad hoc networks, the system may operate in isolation, or may include gateways to interfaces with wired networks, such as internet [35].

The remaining part of this chapter describes the classification of ad-hoc networks, depending on their coverage area: operation, principal issues, main objectives, and contemporary applications. Finally, we provide a description of advantages and disadvantages compared to other type of networks as a critical evaluation

2.1.2 Classification

We may classify mobile ad-hoc wireless networks into three sub-types [16]: Body, Personal, and Local area networks.

Body area network

A body area network (BAN) provides connectivity between wearable devices, such as mobile phones, earphones, microphones or mp3 players. The main requirements of a BAN are:

- Interconnection between heterogeneous devices. E.g. mobile phones with microphone.
- Auto configuration. It should be easy add or remove devices in a BAN
- Services integration. Data transfer of audio and video should be compatible with non-real time data, such as internet traffic.
- Interconnection with other BAN's or personal area computers (PAN's) to exchange information

The radio covered for BAN may be 2-3 meters, because usually they are devices to use close to the body.

Personal area network

Personal area networks (PAN) connect mobile devices one to each other, usually in a range of 10 meters around a person. It is possible to connect a PAN with a BAN dealing to the possibility to make ad hoc networks with any electronic device, such as laptops, PDA's or mobile phones.

Wireless local area network

Wireless Local Area Networks (WLAN) have a range about 100-500 meters; therefore, they achieve more flexibility than wired LAN. It is a good solution for home and office networks. On the implementation of a WLAN, we can target two different approaches: an infrastructure-based approach or an ad-hoc approach. An infrastructure-based approach is based on the existence of an access point that provides access for the mobile devices towards a fixed network, such as internet. On the other hand, in ad-hoc wireless networks a centralized infrastructure is not required.

2.1.3 Operation

The main issues in an ad hoc network design are as follows [35]: medium access scheme, routing, multicasting, transport layer protocol, quality of service provisioning, self-organization, security, energy management, addressing and service discovery, energy management and deployment considerations.

Medium access control

The main responsibility of a medium access control (MAC) protocol in ad hoc wireless networks is the distributed arbitration for the shared channel for transmission of packets. In the chapter 2.2, we will discuss this protocol.

Routing

The main objective of the routing protocols is exchanging route information finding a feasible path to a destination based on different criteria, such as hop length, minimum power required, or lifetime of the wireless link. The foremost challenges of routing protocol in ad hoc wireless networks are explained in the next section [35]:

- Mobility: The mobility associated with the nodes leads to path breaks, packet collision, transient loops, stale routing information, and difficulty in resource reservation. A good routing protocol should be able to minimize these issues.

- Bandwidth constraint: In the same radio range, all nodes share the channel, but only an amount of bandwidth is assigned to each one.

- Error-prone and shared channel: The bit error rate (BER) in a wireless channel is very high, compared to wired networks. The bit error rate is the percentage of bits that have errors relative to the total number of bits received in a transmission. A fair routing protocol should consider this handicap, since a high BER value affects the limited energy resources of a wireless network leading to lost packets.

- Location-dependent contention: When the number of nodes increases in the network, the load on the wireless channel varies and the contention goes up. High contention leads to higher packet collision and wasted bandwidth. Routing protocols should be able to avoid this issue.

- Minimal route acquisition delay: When we need a route, the time to acquire the path towards a node should be as minimal as possible.

- Quick route reconfiguration: The mobility of nodes in wireless ad-hoc networks leads to broken paths. Routing protocols should be able to reconfigure broken paths as quick as possible.

- Loop-free routing: The mobility of nodes in mobile ad-hoc networks may lead to temporary loops formed in the routes. A routing protocol should detect such loops and correct them as soon as possible.

- Distributed routing approach: usually, a central routing approach is not useful in a mobile ad-hoc network, where the mobility of nodes is high. Should be proposed a distributed routing approach.

- Minimum control overhead: Control packets are used to find and maintain routes in the network. A routing protocol should be able to minimize the use of control packets, because it consumes resources of the network.

- Scalability: When the number of nodes grows in a network, a routing protocol should perform well in the new topology with minimum control overhead.

- QoS: Routing protocols should be able to provide quality of service (QoS). QoS is a network guarantee that satisfies a set of predetermined service performance constraints for a client in terms of the end-to-end delay, available bandwidth, and probability of lost packets [16]. For example, QoS is required for multimedia applications.

- Security and privacy: Routing protocols should be able to avoid vulnerability threats or any kind of attacks suffered by an ad hoc network.

Transport layer protocols

The principal responsibilities of the transport layer protocols are congestion control, flow control, and the order delivery of the packets [37]. The major problem with transport protocols in mobile ad-hoc networks arises due to frequent path breaks, stale routing information, high channel error rate, and network partitions. A common transport protocol in ad hoc networks is TCP (transport control protocol), which will be discuss in chapter 2.4

2.1.4 Applications of Ad Hoc Wireless Networks

The principal applications of ad hoc wireless networks are [35]:

- Military applications: In a military environment, could be not feasible to establish a fixed network, due to topology and hazard constraints. In such environments, it is preferable a mobile and fast to develop infrastructure.
- Collaborative and distributed computing: Ad hoc wireless networks are also useful in temporary communication infrastructure, such as conferences. In this kind of events, security is not as crucial as in military environments, whereas the reliability of data transfer is highly importance.
- Emergency operations: In any disaster, the communication infrastructure can be destroyer. Usually, in these environments time and resources are limited, thereby to set up a communication network as quick as possible is required. The main advantages of ad hoc networks in these environments are self-configuration of the system with minimal overhead and fast set up of network configuration.
- Wireless mesh networks are ad hoc wireless networks to provide an alternate communication infrastructure for mobile or fixed nodes/users, without the requirement constraints of cellular networks. Cellular networks are a type of wireless networks composed of cells that cover an operator territory [16]. Examples of mesh networks are mesh networks over highways, over university campus, or over residential zones, where each roof operates like a node in the network.
- Wireless sensor networks: Sensor networks are a kind of ad hoc networks used to provide a wireless infrastructure among the sensors deployed in a specific application domain [35]. The

sensors can be used measuring parameters, such as temperature, humidity, smoke, etc. The major differences between sensor networks and ad hoc wireless networks are listed below:

- Mobility of nodes: in sensor networks, mobility of nodes is not always present. A sensor network monitoring the temperature in an area is an example of no mobility of nodes, whereas sensors installed in vehicles across a metropolis working as sensor nodes is an example of mobility of nodes.
- Size of the network: The number of nodes in sensor networks can be much larger than in ad hoc wireless networks.
- Data-centric: sensor networks focus on the data generated by sensors [8]. For example, if we are interested in those nodes achieving 25 degrees of temperature, data from nodes with less temperature will not be important.
- Power constraints: The sensor network may work in harmful conditions. In these cases, such as emergency operations, there is no human supervision, and recharging the battery of nodes is almost impossible. Hence, power constraints are higher than in ad hoc networks.
- .Data information fusion: data fusion refers to the aggregation of multiple packets into one before retransmitting it. This leads to minimize bandwidth consumed by redundant headers of the packets and smaller medium access delay provoked by transmitting multiple packets.

- Hybrid ad hoc networks are hybrid wireless architectures between ad hoc wireless networks and cellular networks. These kind of networks combines advantages of fixed stations with multi-hop features of ad hoc wireless networks. The principal feature of these networks compared with ad hoc wireless or cellular networks is the channel reuse. Several methodologies are used to achieve this objective, such as cell sectoring, cell resizing, and multi-tier cells. The main advantages achieved are: Higher capacity than cellular networks obtained due to better channel usage. Increased flexibility and reliability in routing: flexibility is achieved from selecting the best mobile node or base station. Reliability is obtained from multi-hop paths in the case a base station fails. Better coverage: due to multi-hop performance over the cell, the connectivity in areas with transmission difficulties is improved.

2.1.5 Critical evaluation

We may evaluate ad-hoc wireless networks comparing them with cellular networks, as they are wireless networks as well. The main differences between cellular networks and ad-hoc wireless networks are as follows:

In cellular networks, routing decisions are taken in a centralized manner with more information about the available destination node; whereas in ad-hoc wireless networks those decisions are taken in the node due to absence of a base station. Consequently, nodes have to manage routing information and host information in a distributed manner. For the reasons exposed above, routing performance is more complex in ad-hoc wireless networks [35].

In ad-hoc wireless networks, paths break frequently because of the mobility of nodes. The routing protocols employed in cellular networks become obsolete, because they cannot manage a good performance searching paths among nodes with high mobility. Additionally, routing protocols should fulfil both lost packets and battery constraint challenges [38].

2.2 The medium access control sublayer

2.2.1 Introduction

Networks can be divided into two categories: those using point-to-point connections and those using broadcast channels. Broadcast networks have a single communication channel that is shared by all the nodes in the network, and point-to-point networks include many connections between individual pairs of nodes.

In every network, there is a channel where data is transmitted. In a broadcast network, the protocols that determine who gets to use the channel are called Medium Access Control (MAC) protocols, which belong to a sub layer of the data link layer. The data link layer is responsible of providing service interface to the network layer, dealing with transmission errors and regulating the flow of data [37].

2.2.2 Issues and design goals

The main challenge in the MAC layer is how to allocate the channel among competing users. Before discussing the major protocols used in the MAC layer, there are several key issues to be noted [37]:

- Single channel assumption: just one single channel is available for all stations.
- Collision assumption: if two frames are transmitting at the same tame, they overlap in time and the signal becomes garbled.

- Slotted time: Time is divided into intervals called slots. Transmission of frames always starts in the beginning of a slot.
- Carrier sense: Stations are able to realize if the channel is busy or not. If so, a station will not attempt to use the channel until it becomes idle.
- No carrier sense: Stations cannot sense the channel before using it. Only after transmission, a station may verify whether the transmission was successful or not.

2.2.3 Operation

The classical Local Area Networks (LANs) use Carrier Sense Multiple Access with Collision Detection (CSMA/CD) in the MAC layer for channel allocation [37]. Using CSMA/CD, if two stations try to get the channel at the same time, they will detect a collision and will abort the transmission. After the collision is detected, a station waits a random period, and then it tries again to get the channel.

This protocol cannot be used in wireless networks because the range of the nodes must to be considered. In wireless networks, the interferences may happen in the receiver, whereas CSMA/CD only considers interference in the sender. This will be explained in the hidden/exposed terminal problem (chapter 2.2.4.2).

2.2.4 MAC protocols in wireless networks

2.2.4.1 Introduction

A common radio channel is shared in ad-hoc wireless networks. Over this radio channel, access channel protocols used in wired networks become obsolete and new challenges must be managed, such as mobility of nodes, limited bandwidth availability, quality of service support, and hidden/exposed station problems.

2.2.4.2 Issues and design goals

Bandwidth efficiency: The radio spectrum where ad-hoc wireless networks operate is limited. Therefore, the MAC protocol must be designed in such a manner that all nodes receive a fair share from the bandwidth available. In addition, the MAC protocol should grant channel access to a node only when its transmission does not affect any ongoing other transmission.

Quality of service support: It is quite complicated to provide quality of service in ad-hoc wireless networks since bandwidth reservation performed at a concrete instant of time may become invalid once the node moves towards out-range positions. In addition, the bandwidth

reservation is hindered by the lack of a centralized station. The MAC protocol should be able to manage those constraints.

Synchronization: Synchronization is crucial for bandwidth reservation, since time slots assigned to the nodes cannot be assigned in a randomize manner. In chapter 2.2.4.6 is showed an example of synchronization and channel reservation in a wireless environment, where the order in which a node gets the channel is very important

Mobility of nodes and no fixed infrastructure: In cellular networks, the base station coordinates the bandwidth reservation among the nodes. In ad-hoc wireless networks, there is not a base station, therefore nodes must schedule access to the medium sharing more control information. The MAC protocol must minimize this overload.

Hidden and exposed problem: The hidden station problem arises when there is collision of packets at the receiving node, because when nodes are transmitting, they are not within the transmission range of each other, but they are in the transmission range of the receiver [35]. Consider figure 1, where S1 is transmitting to R1 and S2 can potentially interfere with R1 but not with S1. If S2 sense the channel, it will not hear S1 because it is out of range, and therefore mistakenly conclude that it can transmit to R1.At this moment, if S1 starts to transmit, it will collide at node S2, resulting in lost packets. The exposed station problem happens when a node concludes mistakenly that cannot transmit, because a nearly node is transmitting to another node. Consider again figure 1, where S1 is transmitting to R1. If S3 sense the channel, it will hear an ongoing transmission, and it will falsely conclude that cannot transmit to R2. In this case, collision could happen only in the zone between S1 and R1.

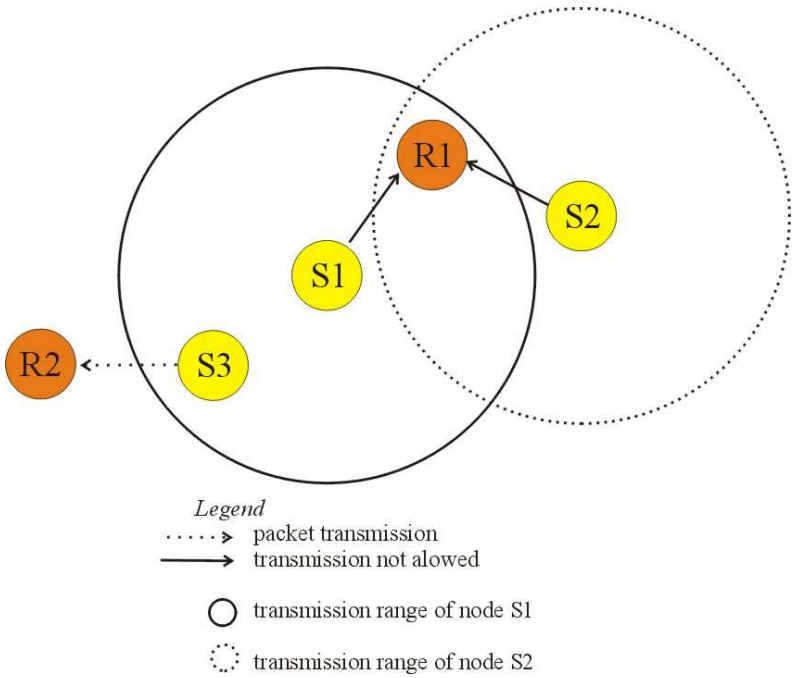


Figure 1:Hidden and exposed terminal problems [35].

2.2.4.3 Operation

MAC protocols for ad hoc wireless networks are classified into three types [42]: contention-based protocols, contention-based protocols with reservation mechanism and scheduled-based protocols. In contention-based protocols, a node does not make any resource reservation a priori; therefore, its periodic access to the channel is not guaranteed. In contention-based protocols with reservation mechanism, nodes are able to reserve bandwidth a priori; therefore, they can support quality of service. In this kind of protocols, if synchronization exists, all nodes in the network are advised when another node is performing a reservation. Finally, scheduled-based protocols are based on scheduling information exchange among the nodes.

Into contention-based protocols suitable for MANETS, most important protocols are MACA[19], MACAW[3], and IEEE 802.11[35].

2.2.4.4 MACA

Since CSMA protocol sense the channel only at the transmitter, the interference may still take place in the receiver, therefore the hidden station problem does not occur.

MACA protocol uses request-to-send (RTS) and clear-to-send (CTS) dialog. Each node upon receiving a RTS or CTS packet, avoids using the channel. Let us now consider A sending a frame to B. A sends a RTS packet to B. This packet contains the length of the data,

and its size is only 30 bytes. Then, B replies with a CTS packet, which also contains the data length, copied from the RTS packet. Afterwards, when A receives the CTS packet, the transmission starts.

Since there is no carrier sense in MACA, each station waits a random amount of time before trying to get the channel when it has heard a RTS or CTS packet. A binary exponential back off (BEB) algorithm performs the amount of time to wait. BEB has not always the same value: a node increases it each time a collision is detected [19]. In MACA, most of collisions occur among RTS packets. Since RTS packet size is so much smaller than data packet, there are fewer overloads compared to CSMA. However, data collision is not guaranteed.

As RTS and CTS packets carry the expected duration of data transmission, each node hearing them will defer its transmission until data delivers to the destination. Based on this approach, MACA will solve hidden station problem [35].

2.2.4.5 MACAW

The binary exponential backoff mechanism used in MACA performs several disadvantages. For example, consider figure 2, where S1 is transmitting packets. In this case, the packets transmitted by S2 are collided, and its backoff window is incremented. Afterwards, the probability of S2 to obtain the channel decreases, becoming blocked after a period. To rectify this, in the packet header is attached the current value of the backoff counter. When a node receives the packet, it copies this value into its own backoff counter. A fairest mechanism to allocate bandwidth is obtained with this modification.

Another improvement at MACAW from MACA is a control packet called acknowledgment (ACK). In MACA, transport layer deals with transmission errors, but the typical implementations of the transport layer have a timeout period of about 0.5 seconds; hence, it is slow recovering errors. In MACAW, the data link layers manage that responsibility. Its performance starts when the sender receives an ACK packet once data has successfully delivered. If ACK packet is lost in transmission, the sender retransmits a RTS for the same packet. In this case, the receiver does not send back a CTS packet; instead, it sends an ACK for the packet received.

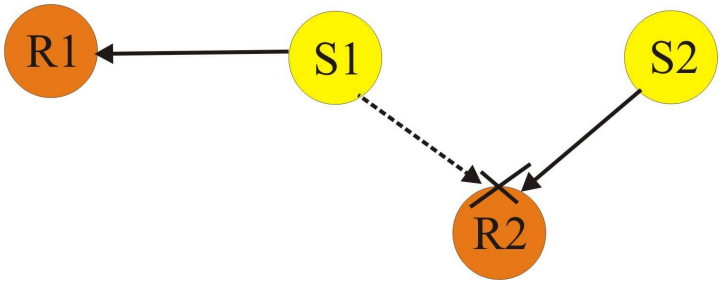


Figure 2: Backoff problem in MACA

There are two more control packets used by MACAW. Data-sending (DS) carries information such as the duration of the data transmission. An exposed node hearing the DS packet realizes that the previous RTS-CTS exchange was successful. Therefore, it defers to transmit until the expected duration of DATA-ACK exchange.

Request-for-request-to-send (RRTS) packet is used to achieve synchronization [3]. Consider figure 3. If there is a current transmission between S1 and R1, and node, S2 wants to transmit to node R2; R1 hears CTS packets from node R1. Therefore, R1 defers its transmission. Node S2 does not know anything about the contention periods during which it can contend for the channel, and it continues trying, incrementing its backoff counter. We can solve this problem by having R2 do the contending on behalf of S2. Then, if a station receives a RTS packet to which cannot respond, it contends during the next contention period and sends a RRTS packet. Neighbouring nodes hearing the RRTS packet wait for two successive slots, enough to hear for a successful RTS-CTS exchange. S2, upon receipt the RRTS packet transmits a RTS to node R2, and the common packet exchange (RTS-CTS-Data-ACK) continues.

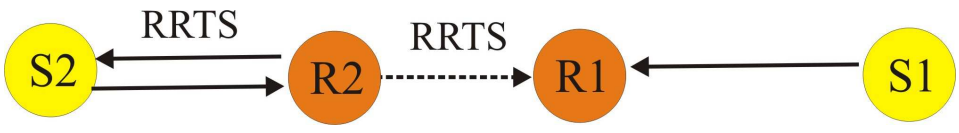


Figure 3:RRTS packet transmission

2.2.4.6 IEEE 802.11

Introduction

Among all these protocols, IEEE 802.11 [48] is the standard for wireless LAN's. Nowadays, IEEE 802.11b and IEEE 802.11g are plentifully used in laptops and personal computers. Using IEEE 802.11g, one communication achieves up to 54 Mbit/s of data rate at the physical layer. In 2007 it is expected that IEEE launches the 802.11n standard, which will enable transmission rates of 540Mbit/s in wireless LAN's.

Operation

Classical LAN uses CSMA/CD protocol to access the channel, listening for other transmission and only transmitting if no one else is doing so. However, this approach does not work in wireless networks, because the interference may occur at the receiver, instead of the sender, as we explained in hidden and exposed station problem [37].

To deal with this, 802.11 MAC protocol supports two models of operation, distributed coordination function (DCF), and point coordination function (PCF). Whereas DCF does not use a centralized control, PCF needs an access point (AP) to coordinate the activity of nodes in its area. PCF is an optional feature at different 802.11 implementations, DCF is obligatory.

DCF

DCF is based on CSMA/CA. Two methods of operation are supported by CSMA/CA [37]. In the first one, when a station wants to transmit, it senses the channel. If the channel is idle, it starts to transmit. If the channel is not idle, the sender defers until the channel gets idle and then starts transmitting. When a collision arises, the station involved waits a random time, using the backoff algorithm, and then tries again.

The second mode bases on MACAW and uses virtual channel sensing (figure 4) [18]. Here, before sending data to a destination, the source sends a control packet (RTS) to the destination. In this packet, the length of the transmission is attached, hence every station receiving this packet stores this information in a local variable named network allocation vector (NAV). The NAV of a station specifies the earliest time when the station is permitted to attempt transmission. After waiting a SIFS (see figure 4), the destination replies with a CTS packet. This CTS packet also contains the duration of the transmission, therefore any station hearing this packet will set its NAV. All stations within the range of the source and the destination are informed that the medium is allocated. The sender, after waiting for SIFS,

starts the data transmission. Then, the receiver, after another SIFS, sends back the acknowledgment (ACK) packet. Afterwards, when the transmission is over, the NAV in each node marks the medium as free, and the process can start again.

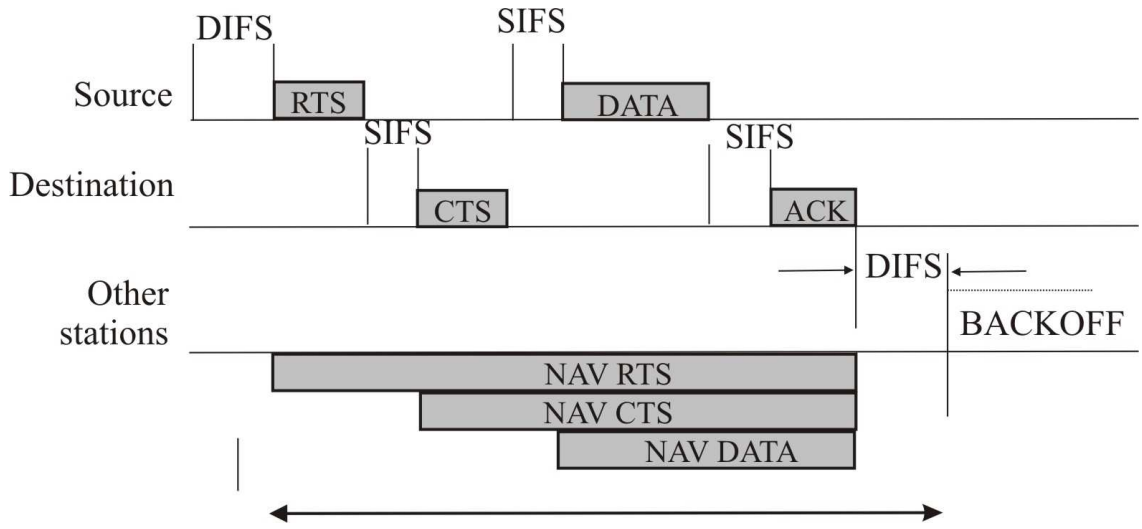


Figure 4: RTS/CTS mechanism in IEEE 802.11[16]

Inter-frame spacing (IFS) is the time interval between the transmissions of two consecutive frames. We can differentiate four IFS in IEEE 802.11 [35]:

- Short IFS (SIFS): it is the shortest interval, which allows the highest priority to access the medium. For example, before a station replies a CTS packet.
- PCF IFS (PIFS): its value lies between SIFS and DIFS.
- DCF IFS (DIFS): it is used by stations operating under the DCF mode to transmit packets.
- Extended IFS (EIFS): it is the longest time interval, which allows the least priority to access the medium. It is used only when a station has to report a bad or unknown frame.

Table 1: IEEE 802.11 parameters [35]

Parameter	802.11(FHSS)	802.11 (DSSS)	802.11(IR)	802.11b	802.11a
t_{slot}	50 µsec	20 µsec	8 µsec	20 µsec	9 µsec
SIFS	28 µsec	10 µsec	10 µsec	10 µsec	16 µsec
PIFS	SIFS + t_{slot}				
DIFS	SIFS + (2 x t_{slot})				

Table 1 shows different values of time slots and SIFS depending on the 802.11 parameter used. However, we may realize that for any parameter, DIFS will always be larger than PIFS, since the value of PIFS lies between SIFS and DIFS.

To reduce the collision probability, the IEEE 802.11 uses a backoff mechanism, which leads to fair time distribution of the transmissions. If a station senses the medium as busy, it defers until the ongoing transmission finishes. At this time, the station initializes a backoff timer by selecting a random interval (backoff interval). The backoff timer is decreasing when the channel is sensed as idle, and stopped when an ongoing transmission is heard. It is re-activated again when the channel is idle an amount of time bigger than DIFS. Afterwards, when the backoff reaches zero, the station starts transmitting [16].

Another improvement from this approach is to set priorities depending on the time spent by a station waiting for the medium. 802.11 DCF uses a binary-exponential backoff for this purpose. The initial backoff window (also called contention window) is established at (0,CWmin). The interval is important, since choosing a too large interval could result in more overload, and choosing a too short interval could result in more collisions. The main advantage of DCF is that contention window is chosen dynamically depending on collision occurrence [35].

PCF

PCF [37], defined by IEEE 802.11, allows the access to the medium with different priority access. This access method uses a point coordinator (PC), which operates at an access point; therefore, PCF operates only in infrastructure-based networks. The point coordinator asks the other stations if they have any packets to send. Since transmission order is organized from the point coordinator, there are no collisions using PCF mode. PCF was never deployed commercially [39].

QoS

Besides these protocols, the current 802.11 MAC protocol has been enhanced to support multimedia applications and quality of service. The 802.11 Task Group (TGe) has developed the virtual distributed coordination function (VDCF) [13], as an enhanced distributed coordination function (EDCF) to be incorporated into IEEE 802.11e standard. The TGe also has specified a hybrid coordination function (HCF) [13] [4].

2.2.5 Critical evaluation

The main goal of MAC protocols is to share a single channel for all communications. When the state of the channel can be sensed, stations should be able to avoid starting transmissions while another station is transmitting. This objective is achieved by CSMA/CD, which is used in classical LAN's. Different approaches are needed in wireless environments.

The hidden and exposed terminal problem is one of the biggest problems to solve for wireless environments. Classical MAC protocols used in wired networks cannot manage this problem, since they are sensing the channel only at the sender [37]. Several approaches were proposed in ad hoc wireless networks. MACA tries to solve the hidden problem with RTS/CTS dialog, but it cannot be solved completely. Additionally, the responsibility of recovering data lies at the transport layer, due to the lack of acknowledgments at MAC layer.

MACAW protocol is an improvement from MACA protocol that achieves faster error recovery than MACA using acknowledgments packets (ACK). Additionally, better performance of hidden and exposed problem is obtained from RTS/CTS/DS/DATA/ACK dialog.

The last wireless protocol analyzed is IEEE 802.11 protocol. This protocol is based on CSMA with collision avoidance for channel access. The collision avoidance is performed by adding a network allocation vector to the RTS-CTS dialog. Although CSMA/CA works well, its performance is better with few terminals. At IEEE 802.11 DCF, widely used in ad hoc wireless networks, a binary backoff algorithm is used to achieve better use of the time among all stations. This algorithm, however, bears several disadvantages. The time spent counting down in the backoff algorithm increases the overhead in the network. Although the backoff interval chosen should be appropriate for efficiency, 802.11 DCF is far away from this purpose. Moreover, the 802.11 DCF leads to highest power consumption, and collision avoidance is not completely achieved [38].

2.3 Routing in ad hoc wireless networks

2.3.1 Introduction

The main goal of the network layer [37] is to choose a correct path to transmit packets from a source towards a destination. For this purpose, routing protocols set up and maintain routing tables, which store information on where packets should be sent next to reach their destinations. Routing protocols should be able to choose the appropriate paths and deals with different network topologies from a source through a destination of data.

Due to the high mobility of nodes in ad-hoc wireless networks, traditional routing protocols used in wired networks cannot be applied directly [35]. Moreover, other characteristics should be considered before choosing a wireless routing protocol, as we have discussed in chapter 2.1.3.

2.3.2 Classification

In wireless ad hoc networks, for unicast routing there is a single source node and a single destination node. In unicast routing two types of protocols are identified: proactive and reactive.

- Proactive or table-driven protocols: In proactive routing protocols, the topology information of the network is stored in routing tables at every node. Therefore, when a node wants to send packets towards a destination, it obtains immediately the path information from its routing table. However, if routing changes happen frequently, keeping and updating routing tables leads to additional overhead in the network. Some proactive protocols are Destination Sequenced Distance Vector (DSDV) [31], Optimized Link State Routing (OLSR) [16], Path-Finding Algorithms (WRP) [27], or Source-Tree Adaptive Routing (STAR) [12].
- Reactive or on-demand protocols: In reactive protocols, when a node attempts to transmit, it calculates the path before data transmission. On one hand, reactive protocols could achieve less overhead since routes are calculated only if needed. On the other hand, the mechanism to discover and maintain routing paths leads to additional overhead than table-driven approach. In addition, the connection setup delay is higher with on-demand protocols. The principal reactive protocols are Dynamic Source Routing (DSR) [18] (which is used in our simulations and it is described below), Ad Hoc On Demand Distance Vector (AODV) [1], Adaptive Distance Vector (ADV) [16], Temporally Ordered Routing Algorithm (TORA) [30], Associative Based Routing Algorithm (ABR) [16], Location-Aided Routing (LAR) [22], Signal Stability-Based Adaptive Routing (SSA) [7], and Flow Oriented Routing[36].

There also exist hybrid protocols, combining both the proactive and the reactive approach [35].

2.3.3 Properties of ad-hoc routing protocols

Ad-hoc routing protocols are desired to accomplish the following issues [35].

- Distributed operations: A distributed routing is more fault-tolerant than centralized routing, since the stability of the network is not supported by just one single point.
- Minimum setup: Quick access to routes by nodes is required

- Loop-free: Stale routes should be avoided, which it usually happens when paths are stored in the cache of each node.
- Packet collision: The number of broadcasts made by each node to discover routes should be minimized.
- Mobility: Routing protocols should be adaptive to topology changes. However, the changes in a part of the network that not affects the node should be avoided. Additionally, it should be able to cover optimal routes once the network becomes stable.
- Best uses of resources: It should achieve an optimal use of resources, such as bandwidth, computing power, memory, and battery power.
- Quality of service: It should be able to provide a certain level of quality of service (QoS).

2.3.4 Dynamic source routing protocol

2.3.4.1 Introduction

The Dynamic Source Routing protocol (DSR) is an on demand routing protocol used in ad hoc wireless networks to allow communication over multiple hops among nodes. As other on-demand routing protocols, the path-finding process is launched only when a path is required by a node to communicate with a destination [35].

2.3.4.2 Operation

DSR was designed to restrict the bandwidth consumed by control packets in ad hoc wireless networks, by eliminating the periodic table-update messages used in proactive protocols.
DSR protocol is based on two mechanisms: Route discovery and route maintenance.

Route discovery

Route discovery is the mechanism by which a node S wishing to send a packet to a destination D obtains a route to D. Route discovery is launched only when S wants to send a packet to D and it does not know a route to D.

Consider figure 5a). S attempts to discover a suitable route to D. The process starts with S broadcasting a route request packet (RREQ), which is received by all nodes within the range of S (B and H in this case). Here, a route request message is composed of the following fields:
- Initiator (sender), and target (destination) of the route discovery.
- Id: Unique identifier for the request.
- List record: stores all the intermediate addresses of nodes through which the route request has been forwarded.

This route request packet is flooded to all its neighbours. Once a node receives a route request, if it is an intermediate node, it appends its own address to the route record in the route request packet, and forwards it by broadcasting. However, the route request packet is discarded if the same route request has already been forwarded, or its own address is in the list record of addresses attached to the route request packet.

Afterwards, a destination upon receiving a route request packet, replies to the sender with the reverse route attached into the route request packet. Consider figure 5a.The node D upon receiving a request packet from S, replies with the route S-B-C-F-D.

Several DSR implementations have been improved using routing cache at intermediate nodes, as it is explained on the coming section

Optimizations

Using routing cache, a node S will initiate the route discovery process whenever cannot find a good source route in its cache. Routing cache is a technique by which a node learns routes from packets that were forwarded. The cache of a node may learn either from the source route used in a data packet, the accumulated route record in a route request, or the route returned in a route reply.

If no route is found in its cache, S will initiate the route discovery process to find dynamically a new route to D. An intermediate node that is receiving a route request, could reply with route reply message (RREP) whenever the target node is already in its cache routing.

In the route reply packet, it is appended the path from the list record where the route request has been forwarded, concatenated with the path that was found in its cache towards the destination node.

If the node is the target of the route discovery, (D in the example), it replies by sending a route reply packet (RREP) towards the source S, with a copy of the route list saved from the route request. In figure 5a), the list is S-B-C-F-D. Upon hearing the route reply packet, the sender and the intermediate nodes save the route into the route cache for future operations. For example, consider again figure 5a. If node C, upon receiving a route request packet, has already a route towards node D in its cache, it may to send back a route replay message to node S with that route

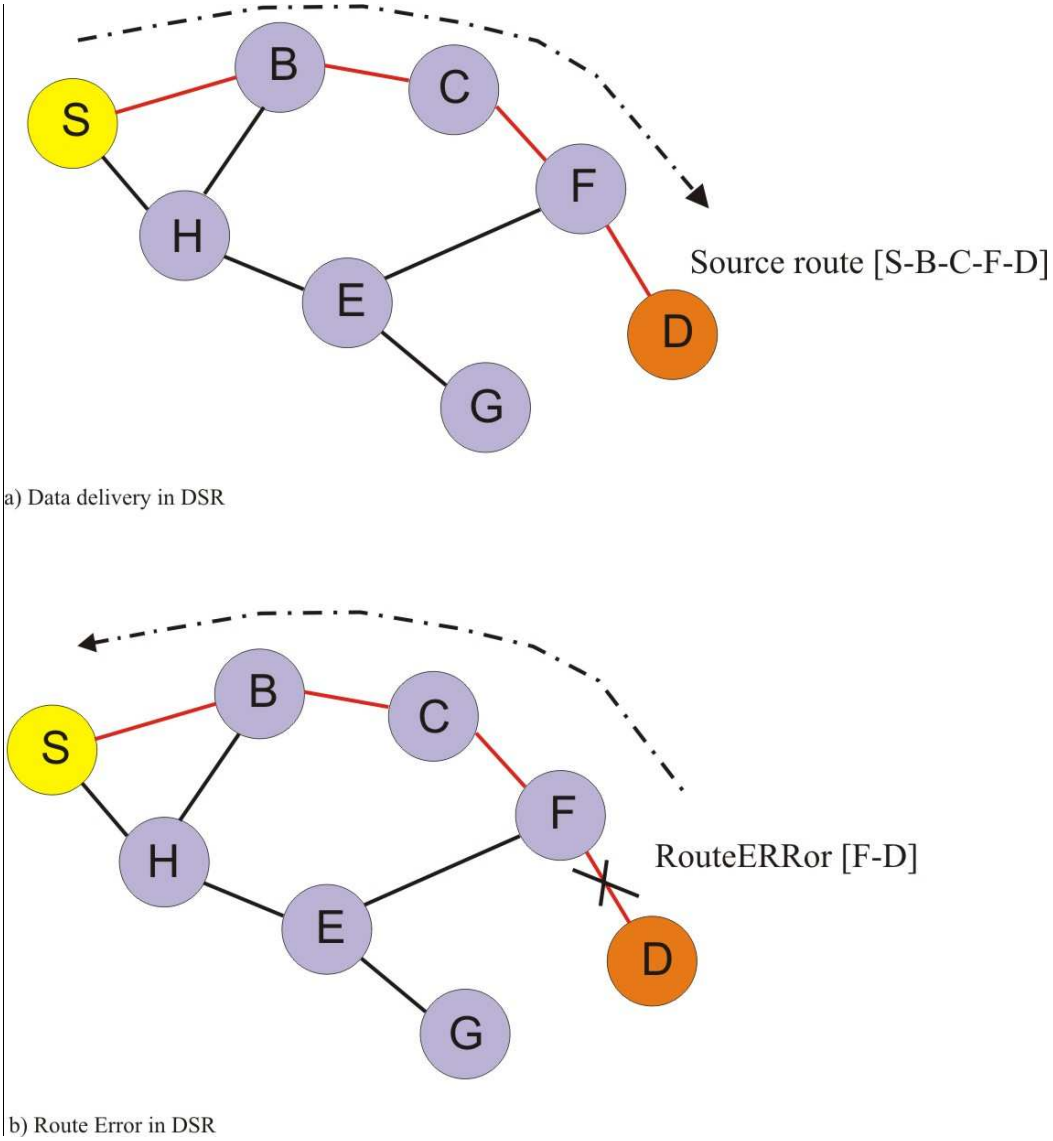


Figure 5:DSR operation

Route maintenance

Sometimes a node cannot deliver the packet in the process of forwarding the packets toward the destination, usually due to the mobility of nodes. In these cases, the adjacent node to the broken link must return a route error packet (RERR) towards the original sender of the packet, identifying the link on which the data could not be delivered. When the sender node receives the RERR with the broken link, it removes the link from its cache, and all the intermediate nodes as well. In figure 5b, each intermediate node will remove the link F-D from its routing cache upon receiving the route error packet . At the same instance, if the sender node wants to send data towards the same destination, it searches in the route cache for another route for this destination; otherwise, the source performs a new route discovery process [18].

2.3.4.3 Critical evaluation

The main advantages are enumerated below [18] [35]:

- DSR, as reactive protocol, avoids the need of flooding periodically the network with table update messages. The route is set up only when it is needed, and using routing cache at intermediate nodes leads to less control overhead, since the route discovery process is triggered less frequently.
- When a node is forwarding packets, it may keep in its cache the routes stored in the packets for future requests. Hence, a single route discovery can produce many routes towards the destination, since intermediate nodes may reply from their local caches.
- Due to its on-demand approach, routing packet overhead automatically scales when mobility or network topology increases.

The main disadvantages are enumerated as follows [35] [38]:

-The ability for intermediate nodes to reply from their caches would lead to route reply storms in some cases. As an example, in figure 5a, if B and D already have a path for a destination D in their cache, they will attempt to send a route reply message, wasting bandwidth and increasing the possibility of collision. A collision can also happen among nodes in the route request phase, when nodes are flooding the route request packet to the entire neighbourhood.
-At high mobility, the routing cache could contain some stale routes, leading to inconsistencies when those routes are used.
-The route maintenance process does not locally repair a broken link.
-The packet header size and per packet overhead grows with route length, because the source length is into the header of the packet.
-The connection setup delay is higher than in proactive protocols.

2.3.4.4 Implicit source routing

Introduction

Using DSR, the source route is stored in the header of the packet. Hence, all routing decisions for a packet are made by the sender, avoiding the need to update information at intermediate nodes. On the other hand, per-packet overhead is increased each time the packet with the

source route is originated or forwarded. Implicit source routing [15] keeps the advantages of DSR source routing and avoids the per-packet overhead.

Operation

Using implicit source routing, each packet is marked with a flow identifier when the sender sends the packet. The flow identifier indicates the route to be followed by all packets belonging to a logical flow from the sender to the destination. Intermediate nodes store a soft state indicating the next hop in the route for that flow, therefore they do not need to save the whole route in the packet and less overhead is achieved.

A source can set up a flow by sending a flow establishment packet. When an intermediate node forwards a packet, it creates a flow table to store information about the flow and the source route. The flow establishment packet contains two headers: the flow identifier and a source route with the timeout of the flow. When an intermediate node receives a packet sent by implicit source routing, it checks its flow table. Whenever the flow identifier into the packet matches with a flow identifier in the flow table, the node forwards the packet setting the MAC layer destination address to the MAC address of the next hop indicated in the flow table entry. Otherwise, the node sends a flow unknown error towards the source.

When the source of the packet receives a flow unknown message, it marks the flow table for this packet as flow must be re-established.

Critical evaluation

Compared with DSR, Implicit source routing achieves improvements in packet delivery ratio. Additionally, although routing packet overhead increased around 12.3 % with implicit source routing, total bytes of overhead decreases between 44% - 80% [15].

2.4 Transport layer in ad hoc wireless networks

2.4.1 Introduction

The main function of the transport layer is to provide reliable and cost-effective data transport from the source towards the destination, independently of the physical and network protocols used. Normally, these services are provided to the users in the application layer.

The Transport Control Protocol (TCP) [35] is a reliable, connection-oriented and full duplex protocol used extensively in wired and wireless networks.

Connection-oriented refers to the service that the transport layer offers to the upper layers. A connection-oriented service follows the next phases: the sender establishes a connection, the sender use the connection, and finally the sender release the connection. In some cases, before a connection is established, negotiation of parameters is performed [37].Full duplex means that traffic can go in both directions at the same time. The main responsibilities of TCP protocol are congestion control, flow control, and ordered delivery of packets. The network layer does not guarantee that packets are properly delivered , neither in the correct order. Thereby TCP builds the right sequence of packets, managing network congestion as well. From these responsibilities, reliability is achieved.

Although the transport layer should not care about whether the network layer is working in wireless or wired networks, in the reality it does matters Transport protocols, such as TCP, are not suitable for wireless networks, due to mobility of nodes, energy constraints, or length of paths [35]. Therefore, new approaches are needed to fulfil these challenges.

2.4.2 Issues and design goals

The main constraints of ad hoc wireless networks affecting TCP performance are explained as follows [9][35]:

- Misinterpretation of congestion window: In wireless networks, TCP interprets all data lost as congestion in the network, because in wired networks data lost are provoked mostly due to congestion in the network than transmission errors. As a response of congestion, TCP slows down the transmission of packets. In wireless networks, links are not reliable at all, because the mobility of nodes leads to frequently broken links. Therefore, it is not effective to slow down the transmission of packets, since the problem is not the congestion. In this case, the proper approach is to send that information again as soon as possible.
- Path length: The TCP throughput rapidly decreases when the path length increase in wireless ad hoc networks. The drop is caused by link loss, which in turn is caused by interference between neighbours.
- Channel asymmetry: Resolving channel contention is usually asymmetric because the sender takes more time to transmit than the receiver does. Therefore, TCP acknowledgments can be queued at the receiver and they can be dropped when they are sent back to the sender. This performance reduces the throughput of the TCP connection.
- Uni-directional path: TCP use acknowledgment packets to ensure reliability. In wired networks, ACKs packet size is short, therefore it consumes low bandwidth. In ad hoc wireless networks using 802.11 protocol in the MAC layer, every TCP ACK packet requires RTS-

CTS-DATA-ACK exchange among the nodes. In consequence, the overhead in the network increases up to 70 bytes without retransmissions.

- Multipath routing: Some routing protocols use multiple paths between source and destination to transmit packets. TCP with multi-path approach can increase out-of-order packets, which provokes duplicate acknowledgments, leading to launch congestion control and degrading the performance of the network.

- Network partitioning: Due to the mobility of nodes, ad hoc wireless networks can be divided in several networks, leading to a partitioned network. TCP should be able to not merge these networks when the sender and the receiver of the TCP session remain in different networks after the partition.

- The sliding-window: When a sender transmits a segment, a timer is also started.. The destination, upon receiving a segment, replies with a segment bearing an acknowledgment, which indicates the next segment expected. If the time expires before the acknowledgment is received, the sender transmits again. In ad hoc wireless networks, the sliding-window leads to degraded performance, due to bandwidth constraints, where the MAC protocol could not perform correctly.

2.4.3 Operation

There are several TCP enhancements in ad hoc wireless networks. The main protocols are feedback-based TCP [4], TCP with explicit link failure notification [14], TCP-Bus [20], ad-hoc TCP [25] and split TCP [23].

NewReno TCP

The presented results in this master's thesis project have been obtained using NewReno TCP enhancement in the transport layer. NewReno TCP is an improvement of Reno TCP in ad hoc wireless networks used by Nahm, Helmy and Kuho [28]. Reno TCP is, as well, an improvement from TCP that incorporates Fast Recovery and Fast Retransmit algorithm [17]..

With fast retransmit, the sender, upon receiving multiple duplicate acknowledgments, concludes packet has been lost and retransmits the packet without waiting for timeout.

Fast recovery operates together with fast retransmit. During Fast Recovery, the sender is able to estimate the amount of outstanding data. Fast recovery starts receiving an initial threshold of duplicate ACK's. Upon receiving the threshold, the sender retransmits one packet and reduces its congestion window by one half. During the whole process, the sender increase its congestion window by the number of duplicate ACK's was received. Fast Recovery is

optimized to work with a single packet dropped from a window of data. However, its performance is not so good when multiple packets are dropped from a window of data [10].
Some problems of Reno are that it cannot distinguish between random loss, usual in wireless environments, and congestion. Therefore, the congestion window is reduced in wireless environments, decreasing the overall throughput. In addition, Reno does not handle loss of packets during Fast Recovery phase.

NewReno incorporates partial acknowledgments (partial ACK). If there is a single packet dropped from a window of data, the acknowledgement for this packet will acknowledge all the packets before Fast Retransmit was started. However, we call partial ACK when there are multiple packets lost, and the acknowledgment for the retransmit packet will acknowledge some, but not all of the packets transmitted before the Fast Retransmit.

In Reno, partial ACK's take TCP out of Fast Recovery decreasing the usable window back to the size of the congestion window. In NewReno, instead, partial ACK's are understood as lost packets, leading to retransmit the packet. Therefore, NewReno can recover all lost packets without a retransmission timeout.

2.4.4 TCP FeW

2.4.4.1 Introduction

Nahm et al. research [28] aims to evaluate the effect of congestion and MAC contention from the interaction between routing and transport layer. We already discussed in chapter 2.4.2 the constraints of congestion window algorithm in TCP. Nahm et al. proposes a fractional window increment (FeW) scheme to improve TCP performance over 802.11 ad hoc wireless networks.

2.4.4.2 Operation

In several previous works based on 802.11 multihop networking, it is analysed how the transport layer is affected by routing protocols. Nahm et al. treat this problem from another point of view; the routing layer is affected by the transport mechanism. Usually, TCP consumes almost all resources in the network, taking resources destined for the routing protocol. Lack of resource in the routing process affects the quality of the end-to-end connection, and decrease the TCP performance.

Figure 6 shows the connection blackout cycle, which explains the problem of the bad interaction among the layers. The loop starts when a TCP sender sends more data than the network can manage (step A). Then, the network overhead provokes MAC contention at the

link layer (step B), which is interpreted as routing failure by the routing protocol (step C). The routing layer triggers routing maintenance process (step D), leading to network overload (step E). The loop repeats until TCP timeouts, or as long as the MAC contention is persistent.

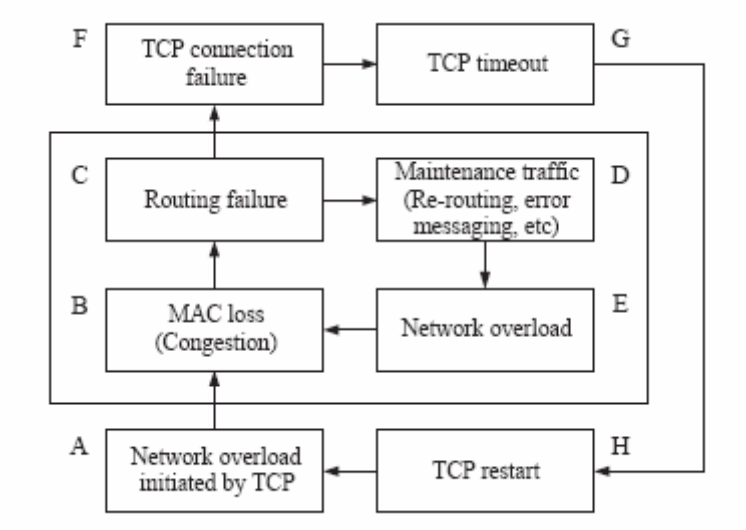


Figure 6: The connection blackout cycle for chain topologies [29].

Nahm et al. propose a fractional window increment scheme (FeW), that keeps TCP congestion window in an interval α <= 1 at every round-time. Consider formula 1. The TCP sender updates the window W upon receiving the correspondent acknowledgments.

(1) $$W^{new} = W^{current} + \frac{\alpha}{W^{current}}$$

Here, α represents the growth rate of the TCP window W at every round-trip-time (α = ΔW). For α = 1, legacy TCP is performed, and TCP window increases one packet at every round-trip-time. This value of α performs well in wired networks, where the bandwidth is high. However, the value of ∞ could be too aggressive in 802.11 ad-hoc wireless networks.

The fractional window interval can be 0<W<=1, but also W>=1. Afterwards, W is updating according formula 1. With α = 0.01, Nahm et al. have found an optimal throughput for the congestion window.

From the results [28], using chain topology, FeW over dynamic routing achieves better throughput than basic TCP for 1 flow (90% more in 6 hops). In grid topology, the results also

show the improvement of the FeW, achieving better results in 7x7 grid than 13x13 grid. Finally, random waypoint shows an improvement of 35 % for 32 flows.

2.4.4.3 Critical evaluation

Table 2: Comparison of TCP solutions for ad hoc wireless networks [35]

Issue	**TCP-F**	**TcP-ELFN**	**TCP-Bus**	**ATCP**	**Split-TCP**
Lost packets due to BER or collision	Same as TCP	Same as TCP	Same as TCP	Retransmits the lost packets without invoking congestion control	Same as TCP
Path breaks	RFN is sent to the TCP sender and state changes to snooze	ELFN is sent to the TCP sender and state changes to standby	ERDN is sent to the TCP sender, state changes to snooze, ICMP DUR is sent to the TCP sender, and ATCP puts TCP into persist state, where TCP's congestion window size is set to one in order to ensure that *TCP* does not continue using the old congestion window value	Same as TCP	Same as TCP
Out-of-order packets	Same as TCP	Same as TCP	Out-of-order packets reached after a path recovery are	ATCP reorders packets and hence TCP avoids sending	Same as TCP

			handled	duplicates	
Congestion	Same as TCP	Same as TCP	Explicit messages such as ICMP source quench are used	ECN is used to notify TCP sender of congestion. Congestion control is same as TCP	Since the connection is split, the congestion control is handled within a zone by proxy nodes
Congestion window after path reestablishment	Same as before the path breaks	Same as before the path breaks	Same as before the path breaks	Recomputed for new route	Proxy nodes maintain congestion window and handle congestion
Explicit path break notification	Yes	Yes	Yes	Yes	No
Explicit path re-establish notification	Yes	No	Yes	No	No
Dependency on routing protocol	Yes	Yes	Yes	Yes	No
End-to-end semantics	Yes	Yes	Yes	Yes	No
Packets buffered at intermediate nodes	No	No	Yes	No	Yes

- Feedback-based TCP [4]: The main advantage of TCP-F is the ability to restore a broken path in a short amount of time. Furthermore, a failure point is able to send a route failure notification to the sender noticing a broken path. On the other hand, the main disadvantages are that its implementation requires modifications from TCP, and the congestion window used (after a new route is obtained), could not achieve a transmission rate suitable for the network.

- TCP with explicit link failure notification [14]: The advantage of TCP-ELFN is an improvement from TCP by using a link failure notification upon detecting a link failure to the TCP sender. The disadvantages are wasting of bandwidth and resources, because of the use of probe packets to check the route reestablishment. Additionally, like TCP-F, the congestion window used after a new route is obtained could be insufficient for the network.

- TCP-Bus [20]: The advantages of TCP-Bus are the improvement and avoidance of fast retransmission by using buffering, sequence numbering, and selective acknowledgments. The disadvantages include high dependency from the routing protocol, and its decreased performance when path breaks at intermediate nodes.
- Ad Hoc TCP [25]: One of the advantages of ad hoc TCP is its compatible performance with TCP, allowing it work with the internet. In addition, other advantage is the improvement achieved to work over ad hoc wireless networks. Some disadvantages include the dependency on the routing protocol, and the addition of a new layer to the TCP/IP protocol stack that requires some changes in the interface functions used.
- Split TCP [23]: The main advantages of Split TCP are the improvement of throughput and less impact of mobility. We have been discussed in section 2.4.2 how TCP throughput decreases due to the path lengths. Using split TCP, shortest path segments operates at its own range, increasing the throughput and minimizing mobility of nodes problems. The main disadvantage of split-TCP is that requires several modifications from TCP protocol. Since TCP uses end-to-end approach, it does not perform TCP mechanisms at intermediate nodes. Using split-TCP, the intermediate nodes process TCP packets, therefore certain security mechanisms that require IP payload encryption cannot be used. Additionally, split-TCP is affected by frequent path breaks.

2.4.5 UDP IN WIRELESS NETWOKS

2.4.5.1 Introduction

The User Datagram Protocol (UDP) is a connectionless transport protocol, which transmits packets of 8 bytes of header followed by data [37]. It does guarantee neither datagram delivery, nor the right order. Using connectionless service, it is not required a session connection between the sender and the receiver, as connection-oriented does.

2.4.5.2 Operation

UDP does not perform flow control, error control, or retransmission upon receiving a bad packet; but it is applicable in communications where less overhead is required, due to the small size of the UDP header.

For example, in client-server situations, the client sends a short request to the server, which replies with a message back. If some message is lost, the client just tries again the request.

With this approach, less overhead is achieved, since is not needed an initial connection setup as other protocols performs.

UDP header is composed by four fields: source port, destination port, UDP length, and UDP checksum. The ports are required for the transport layer to deliver packets correctly. UDP length indicates the length of the packet, and UDP checksum is an optional field.

Although UDP does not suffer from the same problems that TCP in wireless networks, it also decreases its efficiency in this kind of networks [37].

2.4.5.3 Critical evaluation

The main disadvantage of UDP includes that is not reliable, does not guarantee delivery of packets, and does not guarantee ordered delivery. On the other hand, its low overhead makes it suitable for client-server communications, such as Domain Name System (DNS) [5].

A useful study comparing UDP performance in wireless and wired networks is provided by Adigum, M. and Akintola, A. [2].

3 The ns-2 Network Simulator

3.1 Introduction

The use of network simulations could be understood as a cheaper way of protocol validations (both money and time), where the experimental conditions can be controlled. The last version of the network simulator, ns- 2 [51], provides many protocols like TCP, UDP or HTTP, different traffic source behaviour like CBR, FTP or VBR, propagation models, MAC layer protocols, tools for topology generation and visualization [11]. In this master's thesis project, we focused on MAC 802.11 implementation and DSR routing protocol.

The network simulator ns-2 was developed as collaboration among researchers at UC Berkeley, USC/ISI, LBL and Xerox PARC with the main purpose of simulating the behaviour of networks. The version 2 of network simulator is written in C++ and OTcl, and it is continuously evolving.

The main reason for using two languages is time saving. C++ is a powerful programming language, which enables fast execution of applications, but some modifications may be requested in order to perform several simulations, that is, keeping the main structure of the simulation but modifying some parameters, with the purpose of comparing different results. That implies additional time recompiling C++ code every time a modification is requested. OTcl is an interpreted language, and the main advantage is that these modifications do not need additional time recompiling but on the other hand, the execution time for an interpreted language is slower than compiled languages. Ns-2 network simulator makes feasible this unification through tclcl, i.e. OTcl linkage. The main objects of a simulation such as nodes and protocols are implemented using C++ and the configuration of the parameters such as number or position of nodes, time of the simulation, etc. are implemented in OTcl.

In the last years, ns-2 became in the most popular tool in MANET research [24], because it includes wireless and ad hoc networking support and easy configuration files and scripts. Basically, the network simulator is an interpreter of OTcl with network simulator‘s object libraries [52].

3.2 Running Simulations

The components of the ns-2 are the network objects, the event scheduler, an input simulation program and the simulation results. To manage these objects and the event scheduler an OTcl script should be defined, i.e. the input simulation program. The event scheduler keeps a record of simulation time triggering all events in the event queue scheduled at this moment. The communication among network components does not consume simulation time, except the necessary time that a node needs for handling a packet, implying a delay, which is managed by the event scheduler. The event scheduler is also used as a timer, e.g. in a packet retransmission.

The execution of the ns-2 is mainly the execution of an OTcl script. In this section, an overview about these concepts is described.

3.2.1 Scenarios

Since the simulation scenarios are scripts, it is necessary to define them. The simulation scenario is composed of topology, agent and routing information. The first step is to initiate a simulator instance and choose the output for the results and the position and numbers of nodes are configured. Later it should be defined the transport protocol agent and links among nodes. The last step is to define the routing protocol to be used in the simulation. The result of the simulation is one or more text files containing detailed simulation data. These files can be used for simulation analysis or as an input for a visualization tool called NAM (Network Animator). [11][52].

3.2.2 Nodes

In the network simulator, nodes are crucial for the transmission of packets. Upon receiving a packet in a node, the fields of the packet are analyzed, which include the destination address, that is, the receiver node.

All the nodes contain at least the following components [11]:

- An address or id_, monotonically increasing by 1 (from initial value 0) across the simulation namespace as nodes are created
- A list of neighbours (neighbour)
- A list of agents (agent_)
- A node type identifier (nodetype)
- A routing module

The configuration of nodes takes place in the definition of the scenario. Here, we can set up important parameters for our simulation, such as type of addressing structure, network components for mobile nodes and the type of routing used.
As an example of the configuration of nodes:

```
$ns_ node-config -addressType hierarchical \
    -adhocRouting DSR \
    -llType LL \
    -macType Mac/802_11 \
    -ifqType Queue/DropTail/PriQueue \
    -ifqLen 50 \
    -antType Antenna/OmniAntenna \
    -propType Propagation/TwoRayGround \
    -phyType Phy/WirelessPhy \
    -topologyInstance $topo \
    -channel Channel/WirelessChannel \
    -agentTrace ON \
    -routerTrace ON \
    -macTrace OFF \
    -movementTrace OFF
```

This configuration file can take the following values from the options of the network simulator [11]:

general		
addressType	Flat (Node address is the same that node id), hierarchical (Node address in form of string: “1.5.2”)	flat
MPLS	ON, OFF	OFF
both satellite and wireless oriented		
wiredRouting	ON (in the case of a base station), OFF (mobile nodes)	OFF
llType:	LL (data link layer. Functionalities: queuing and link-level retransmission), LL/Sat (link layer for satellite links. “The transmit and receive interfaces must be connected to different channels, and there is no ARP implementation.”)	“”
macType	Mac/802_11, Mac/Csma/ca, Mac/Sat, Mac/Sat/UnslottedAloha, Mac/Tdma Medium access protocol between the link layer and physical layer. It may content carrier sense or collision avoidance depending on the physical layer.	“”

ifqType	Queue/dropTail, Interface queue type, Queue/DropTail/PriQueue (It gives priority to routing protocol packets, head of queue.)	“”
phyType	Phy/WirelessPhy, Network interface type for wireless, serves as a hardware interface. Phy/Sat, Interface for satellite nodes.	“”

satellite-oriented		
satNodeType	Polar: the polar orbiting satellites in ns have purely circular orbits, defined by altitude, latitude and inclination. Terminal: A terminal is specified by its latitude and longitude Geo: A geostationary satellite is specified by its longitude above the equator, for processing satellites. geo-repeater. Degenerate satellite node, for processing bent pipe satellites.	“”
downlinkBW	<bandwidth value for downloading, e.g. “2Mb”>	“”

wireless-oriented		
adhocRouting	DIFFUSION/RATE, DIFFUSION/PROB, DSDV, DSR, FLOODING, OMNIMCAST, AODV, TORA. Routing agents implemented for mobile networking.	“”
propType	Propagation model, attached when the physical layer is defined. See section 3.6. Propagation/TwoRayGround, Propagation/Shadowing	“”
propInstance	Instance of the propagation model. See section 3.6. Propagation/TwoRayGround, Propagation/Shadowing	“”
antType	An omni-directional antenna having unity gain is used by mobilenodes. Antenna/OmniAntenna	“”

Channel	The Channel object simulates the shared medium and supports the medium access mechanisms of the MAC objects on the sending side of the transmission. Channel/WirelessChannel, for wireless nodes Channel/Sat, for satellite nodes	“”
topoInstance	It is used to provide the node with a handle to the topography object. <topology file>	“”
mobileIP	The mobileIP scenario consists of Home-Agents(HA) and Foreign-Agents(FA) and have Mobile-Hosts(MH) moving between their HA and FAs. ON, OFF	OFF
energyModel	The energy model represents level of energy in a mobile host. EnergyModel	“”
initialEnergy	Value for the energyModel at the beginning of the simulation. <value in Joules>	“”
rxPower	Energy usage for received packet <value in W>	“”
txPower	Energy usage for transmitted packet <value in W>	“”
idlePower	Energy consumption in idle state <value in W>	“”
agentTrace	ON: enables tracing at agent level, visualization of agent events in the trace file OFF	OFF
routerTrace	ON: enables tracing at router level, visualization of routing events in the trace file OFF	OFF
macTrace	ON: enables tracing at MAC level, visualization of MAC events in the trace file OFF	OFF
movementTrace	ON: enables mobilenode movement logging OFF	OFF

errProc	Error model simulates link-level errors or loss by either marking the packet's error flag or dumping the packet to a drop target. UniformErrorProc	""
FECProc	?	?
toraDebug	ON: it enables debug messages if tora routing protocol is used.	OFF

3.2.3 Agents

Agents are used in ns-2 in the implementation of protocols [11]. These are some protocol agents available, which we have used in our simulation**:** TCP/Reno, a Reno TCP sender (with fast recovery); TCPSink, a Reno or Tahoe TCP receiver (not used for FullTcp); UDP, a basic UDP agent; LossMonitor, a packet sink with checks for losses.

Within OTcl, it is possible to create and modify objects of type agent. In ns-2, it is also possible to create a new Agent class, composed of internal state and methods, supporting packet generation and reception [11].

3.3 Traces files

The main objective of the traces files is to provide different types of information from the simulation. In ns-2, there are three types of traces, the old format, the new format, and a tagged trace format.

3.3.1 Trace configuration

In the ns-2 network simulator, the next commands are used to configure the trace format:

use-newtrace: selects the new trace format.
use-taggedtrace: selects the tagged trace format.
trace-all $fd means that tracing referring to a trace file should be performed by the fd file descriptor.
namtrace-all $namfd means that tracing should be performed to a NAM (network animator) trace file referenced by the namfd file descriptor.
nametrace-all-wireless $namfd <x> <y> is similar to namtrace-all, but it extends the NAM trace with information about the size of the topography (its x and y dimensions).
flush-trace behaves like a pipe, flushing all open traces to a disk. Must be used in the end of the simulation.

As an example of trace configuration:

```
set val(x) 6000 ;# X dimension of the topography
set val(y) 3000 ;# Y dimension of the topography
set val(stop) 200.0 ;# simulation time
set tracefd [open $val(tr) w]
set namtrace [open $val(nam) w]
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)
$ns_ node-config -agentTrace OFF \
                 -routerTrace ON \
                 -macTrace ON \
...
proc finish {} {
global ns_ tracefd namtrace
$ns_ flush-trace
close $tracefd
close $namtrace
$ns_ halt
}
$ns_ at $val(stop) "finish"
```

3.3.2 Trace formats

In ns-2, we can choose between two types of trace format, the old format and the new format.

Old format

In the old format, we can notice that it is easier to understand than the new format. Wireless traces starts with one of four characters followed by one of two different trace formats (X or Y coordinates of the mobile node).

Table 3 : Old format of traces

Event	Abbreviation	Type	Value
		%.9f %d (%6.2f %6.2f) %3s %4s %d %s %d [%x %x %x %x]	
		%.9f _%d_ %3s %4s %d %s %d [%x %x %x %x]	
		double	Time
		int	Node ID

Wireless Event	s: Send r: Received d: Drop f: Forward	double	X Coordinate (if Logging Position)
		double	Y Coordinate (if Logging Position)
		string	Trace name
		string	Reason
		int	Event identifier
		string	Packet type
		int	Packet size
		hexadecimal	Time to send data
		hexadecimal	Destination MAC address
		hexadecimal	Source MAC address
		hexadecimal	Type (ARP, IP)

New format

The simplicity of the old trace format has as a main drawback that it is needed a different trace for each wireless protocol. The new trace format was introduced aiming to merge these wireless traces using cmu-trace objects [11]. The new trace format is compatible with the old trace format.

In ns-2, it is possible to set up this feature with the command $ns use-newtrace. In our simulations, we used the new format of traces, which allow us to offer all the information about simulation traffic. The next table explains briefly the meaning of each field [11]:

Table 4 : New format of traces

l	**s** send **r** receive **d** drop **f** forward // describes the type of event in the trace
-t **-t***	time global setting
Node properties	
-Ni	node id
-Nx	node'x-coordinate
-Ny	node's y-coordinate
-Nz	node's z-coordinate

-Ne	node energy level
-Nl	trace level (AGT, RTR, MAC)

-Nw	**"END"** DROP_END_OF_SIMULATION i.e. indicates that the simulation has finished. **"COL"** DROP_MAC_COLLISION i.e. there was a collision in MAC layer **"DUP"** DROP_MAC_DUPLICATE i.e. there is already the same packet in MAC **"ERR"** DROP_MAC_PACKET_ERROR i.e. incoming packet with errors in MAC **"RET"** DROP_MAC_RETRY_COUNT_EXCEEDED i.e. maximum number of retransmission attempts in MAC reached. **"STA"** DROP_MAC_INVALID_STATE i.e. Internal MAC state is not valid, e.g. in receiving function: state = sending. **"BSY"** DROP_MAC_BUSY i.e. MAC is occupied in other task. **"NRTE"** DROP_RTR_NO_ROUTE, i.e. no route is available. **"LOOP"** DROP_RTR_ROUTE_LOOP i.e. there is a routing loop **"TTL"** DROP_RTR_TTL i.e. TTL has reached zero. **"TOUT"** DROP_RTR_QTIMEOUT i.e. packet has expired. **"CBK"** DROP_RTR_MAC_CALLBACK **"IFQ"** DROP_IFQ_QFULL i.e. no buffer space in IFQ. **"ARP"** DROP_IFQ_ARP_FULL i.e. dropped by ARP **"OUT"** DROP_OUTSIDE_SUBNET i.e. dropped by base stations on receiving routing updates from nodes outside its domain.

Packet information at IP level		**Packet information at app. level (Cont.)**	
-Is	source address source port number	**-Pa**	destination mac address
-Id	dest address dest port number	**-Pd**	destination address
-It	packet type	**-Pn**	how many nodes traversed
-Il	packet size	**-Pq**	routing request flag
-If	flow id	**-Pi**	route request sequence number
-li	unique id	**-Pp**	routing reply flag
-lv	ttl value	**-Pl**	reply length
Next hop info		**-Pe**	source of srcrouting->dst of the source routing
-Hs	id for this node	**-Pw**	error report flag
-Hd	id for next hop towards the destination	**-Pm**	number of errors
Packet info at MAC level		**-Pc**	report to whom
-Ma	duration	**-Pb**	link error from linka->linkb
-Md	dst's ethernet address	**-Pi**	sequence number
-Ms	src's ethernet address	**-Pf**	how many times this packet was forwarded
-Mt	Ethernet type	**-Po**	optimal numbers of forwards
Packet information at app. level		**-Ps**	sequence number
-Po	ARP request/reply	**-Pa**	acknowledgment number

-Pm	source mac address	**-Pf**	how many times this packet was forwarded
-Ps	source address	**-Po**	optimal numbers of forwards

3.4 Analysis of trace files

The NS-2 [51] network simulator produces a trace file, which may include all information available within the packets transmitted along the simulation, like sender and receiver node id, position, time, energy, etc., and their presence is configurable in the set up “main.tcl” file and in the file “cmu-trace.cc”. The analysis of these files in our dissertation was performed using PERL scripts to extract and to present the information in a graphical view with the help of Excel, giving a result better to understand than the trace files.

In this chapter, we present how we performed the analysis of the trace files. For more information about the analysis files or the modifications in the source code from ns-2, Appendix C should be consulted.

Note that the following sections of PERL are based on our setup files and the index of the parameters in the trace files may vary. First, we open the trace file, and, iteratively, each line is introduced in an array “ll” using the blank space as separator for each element in the line:

```
open TRACE, "result/scenario/tracefile.tr";
while (<TRACE>) {
   @ll = split(' ');
. . .
}
```

All scripts we used for different scenarios share the same basis and only some parameters were modified, for instance, for calculating the throughput all received packets were counted, the only difference could be the number of flows used in different scenarios, but they share the same basis.

```
if ( $ll[0] eq "s" && $ll[18] eq "AGT")
   {$nsend +=1;}
if ( $ll[0] eq "r" && $ll[18] eq "AGT" )
   {$nrecv +=1;}
```

In the code above, all sent and received packets by the agent are stored in variables, later on, the bytes per packet will be multiplied by those totals. The throughput is the received bytes (divided by 1024 for getting Kilobytes) divided by the time of simulation (KB/sec).

The number of lost packets is a subtraction between sent and received packets. All sent packets should reach their destinations, if they do not do it is because these packets were lost.

In order to calculate the routing overhead in the network the number of routing packets is also needed. This is obtained introducing the following line inside the analysis script:

```
if ( $ll[18] eq "RTR" && $ll[34] eq "DSR" )
   { $nRTR_OH++;}
```

Routing overhead is the number of additional information used for a transmission of data divided by the total of bytes for the complete transmission.. For instance, if there are needed 5 additional bytes for sending 10 bytes of data that means that 33% of the information is routing overhead, this should be minimized.

At MAC layer can be found four types of discarded packets depending on the reasons like collisions, duplicated packet, retry exceeded count or mac busy. Most of dropped packets at MAC layer are due to collisions, approximately 95 percent. These collisions may affect to routing layer, therefore it is important to know the reasons of dropped packets and in which situations that may happen.

```
if ( $ll[0] eq "d" && $ll[18] eq "MAC"){
   elsif ( $ll[20] eq "COL" )
      {$col++;}
   elsif ( $ll[20] eq "DUP" )
      {$dup++;}
   elsif ( $ll[20] eq "RET" )
      {$ret++;}
   elsif ( $ll[20] eq "BSY" )
   {$bsy++;}
}
```

The number of route errors triggered by DSR is one of the main parts of our work. We added special information in the trace produced by the DSR exactly when they are detected, for analyzing then the trace file generated.

```
if ($ll[0] eq "RERR" )
   {$rerr +=1;}
```

In order to define this information in the trace file, the following code must be included in the method “xmitFailed”, in the file dsragent.cc. The variable “gamma” was used to switch between normal DSR and DSR-AR approach. Note that a route error will not be triggered in DSR-AR if the “xmit_reason” is “high power”, that means that communication to the neighbouring node was not possible, but it is reachable, that is, the route exists. Therefore, a route error should not be triggered and that route should not be deleted from the cache.

```
   if ( gamma_) {
      if ( cmh->xmit_reason_ == XMIT_REASON_HIGH_POWER
         && strcmp(reason, "DROP_RTR_MAC_CALLBACK") == 0 ) {
         Packet::free(pkt);
         pkt = 0;
         return;
      }
     }

     trace ("RERR %.5f %s -> %s -GH %d", Scheduler::instance().clock(), from_id.dump(),
to_id.dump(),God::instance()->hops(from_id.getNSAddr_t(), to_id.getNSAddr_t())

   link_down *deadlink = &(srh->down_links()[srh->num_route_errors()]);
   deadlink->addr_type = srh->addrs()[srh->cur_addr()].addr_type;
   . . .
```

Each time a route error is triggered, the route to reach the destination node changes. Note that the process to discover a route from a sender node to a destination may include several changes in the route, which are counted as well.

In the code below for route changes in chain scenario, the variable ”H” indicates the number of hops.

```
   if ($ll[0] eq "RChange"){
      if ($N == 2){
         if ($ll[4] eq "0" && $ll[6] eq $H)
            {$RChangeFlows++;}
      }elsif ($N == 4){
         if  ($ll[4] eq "0" && $ll[6] eq $H)
            {$RChangeFlows++;}
      }elsif ($N == 8){
         if  ($ll[4] eq "0" && $ll[6] eq $H )
            {$RChangeFlows++;}
      }
   }
```

It is possible as well to track the number of route changes for a specific route between two nodes. For instance in the following code, we created a graphic along the simulation time for the route changes between node 21 and node 27. In the file defined by RT_CHG will be saved the time, the length of the route in hops and the optimal length of the route, which is also called “god hops”.

```
if ( $ll[0] eq "RChange"){
   if ($ll[4] eq "21" && $ll[6] eq "27")
      print RT_CHG "$ll[1] "$ll[3] "$ll[8]\n";
   }
}
```

The new trace format of the ns-2 contains a special trace for these cases called SFEST, but if a user wants to personalize a new message, the following code can be useful (method “sendOutPacketWithRoute”):

```
if (srh->num_addrs()) {
   trace ("RChange %.9f h: %d %s -> %s -god %d",
      Scheduler::instance().clock(),p.route.length()-1,
      p.src.dump(), p.dest.dump(),
      God::instance()->hops(p.src.getNSAddr_t(), p.dest.getNSAddr_t());
   }
```

In the code above, the actual time, actual hops, source, destination and god hops are written into the trace file.

The final step after analyzing the trace file is to display the results in a graphical view. We used excel for most of the graphics and gnuplot for the length of the route between source and destination along the simulation time.

3.5 802.11 MAC in Network Simulator

3.5.1 Introduction

We can choose between two MAC protocols in ns-2, 802.11 and TDMA. Our simulations results are obtained using 802.11 protocol. In ns-2, the MAC layer performs an important function, since it has the responsibility to manage and to understand the packet information. The MAC layer can receive different kind of packets, such as RTS, CTS, ACK or data. The response of the MAC layer depends on the kind of packet received. The 802.11 protocol follows the dialog RTS-CTS-data-ACK, which was explained in section 2.2.5

3.5.2 MAC features

The code can follow four different paths in ns-2; transmitting a packet, receiving a packet destined for itself, overhearing a packet not destined for itself or packet colliding.

Transmitting a packet takes the following path (if not errors or congestion):

recv()->send()->sendDATA() and sendRTS()-> start defer_timer-> deferHandler -> check_pktRTS()-> transmit()-> recv()-> receive_timer started-> recv_timer()-> recvCTS -> tx_resume()-> start defer_timer ->rx_resume()-> deferHandler()-> check_pktTx() -> transmit()-> recv()-> receive_timer started -> recv_timer()-> recvACK()-> tx_resume() -> callback-> rx_resume()

When the first RTS fails:

recv()-> send()-> sendDATA() and sendRTS()-> start defer_timer-> deferHandler() -> check_pktRTS()-> transmit-> start send_timer-> send_timer()-> RetransmitRTS() -> tx_resume()-> backofftimer started backoffHandler()-> check_pktRTS()-> transmit -> recv()-> receive_timer started-> recv_timer()-> recvACK()-> tx_resume()-> callback_ -> rx_resume()

Receiving a packet takes the following path (if not errors or congestion):

recv()-> receive timer started-> recv_timer()-> recvRTS()-> sendCTS()-> tx_resume()-> start defer_timer-> rx_resume()-> deferHandler()-> check_pktCTRL()-> transmit()-> recv() -> receive_timer started-> recv_timer()-> recvDATA()-> sendACK()-> tx_resume()-> start

defer_timer-> uptarget_-> recv()-> deferHandler()-> check_pktCTRL()-> transmit()-> start send_timer-> send_timer()-> tx_resume()

3.5.3 MAC functions behaviour implemented in NS-2

One of each function summarizes above is explained as follows [45]:

recv (down): This function checks the direction saved in the packet header. If the direction is down, it means that the packet comes from an upper layer, hence the packet is followed to the send() function.

recv (up): In this case, the packet is received from a lower layer, and different threads can be followed on depending what the MAC is doing in such moment. If the packet is received while the MAC is already transmitting other packet, then the received packet will be ignored. If the MAC is idle not receiving any packets, then rx_state is changed to RECV and checkBackoffTimer process is called. After that, the incoming packet is assigned to pktRx_, and the receiver timer is set for the txtime() of the packet. If the MAC is receiving any packet when the new packet arrives, the power of both packets is compared. At this moment, if the power of the new packet is smaller than the old packet by at least the capture threshold, the new packet is ignored, calling the capture() function. When the power of both two nodes is too close, there will be a collision and collision() function is called, which will drop the new packet and the old packet (this last one when the reception is completed).

send(): First, this function checks the energy model; if the packet is in sleep mode, then is dropped. Afterwards, sendData() and sendRTS() are called, which build the MAC header for the data packet and the RTS packet. Meanwhile, a sequence number is assigned to the MAC header. At this moment, the MAC checks if the medium is idle (using is_idle() function, and only if the backofftimer is not currently counting down). If so, the node will defer a DIFS time plus a random time chosen from the interval [0,cw (congestion windows)]. If the node is already waiting on it's defer time, it will continue waiting; but if the medium is busy, the node will start its backoff timer. The flow of control of the send function ends at this point, where deferHandler() or backoffHandler() functions will take it.

sendData(): Sets up the MAC header for the data packet and it establishes the packet type as data. Additionally, txtime() function saves the txtime of the packet. Txtime refers to the size of the packet multiplied by the data rate. When the packet is not a broadcast packet, the duration field of the MAC header is calculated. This field refers to the time that the communication needs after the data packet has been transmitted. If this packet is a data

packet, it refers to the time to transmit an ACK plus a short inter-frame spacing. Wherever seems to be a broadcast, the duration is set to zero, hence no ACK is needed in a broadcast. Finally, the MAC header for the packet has been built, and this is noticed with the pktTx_ variable. At this point, the flow of control returns to the send() function.

sendRTS(): The purpose of this function is to create an RTS packet with the destination plus the data packet that the MAC is sending. First, the RTS threshold is checked. Wherever is a broadcast packet, or smaller than the threshold, RTS/CTS mechanism is avoided and control is returned to send() function. Otherwise, a new packet is created as a MAC packet complemented with the RTS fields provided by the rts_frame. In this case, the duration field is calculated as the time to transmit a CTS, the data packet (pktTx_) and a ACK (plus 3 SIFS). After the RTS has been built, the control leads to the send() function again.

sendCTS(): in this function, the CTS packet is created with a pktCTL_. In this case, the duration value is set to be the same as RTS minus the txtime of a CTS and a sift_time. Once the packet is built, pktCLT_ points to the new packet and flow of control returns to recvRTS().

sendACK(): When the data packet arrives successfully to the destination, an ACK packet is set up and sent towards the sender. This procedure is achieved by sendACK(). The duration field is marked as zero, suggesting to other nodes that once the ACK is sent, they do not have to delay other communication. When the packet is built, pktCLT_ points to the new ACK and flow of control goes back to recvDATA().

deferHandler(): This function is called when defer time expires. In this case, the node was waiting an amount of time before to proceed with the transmission, and now it will try to transmit. First, the function calls check pktCTRL() to check if the backoff timer is already running. check_pktRTS() and check_pktTx() are called to check if these functions are managing the current packet transmission; if so, deferHandle stops. After the expiration of the interface timer, the control will be resumed. Additionally, control will be resume if another packet is received from recv() (can be a CTS, data packet or ACK), or upon expiration of the send timer, sendHandler(), which will call send_timer().

check_pktCTRL(): The major purpose of this function is to transmit and check CTS and ACK packets. For CTS packet, first the MAC senses the medium calling is_idle() function. If the medium is busy, CTS will be dropped; otherwise the function set tx_state () to MAC transmitting a CTS and checkBackoffTimer() is called. After that, function calculates timeout value, this is, the time for how long MAC decides the packet was not delivered successfully. If the control packet is an ACK, the MAC performs the same except sensing the medium.

Finally, transmit() is called with pktCTRL_ and the timeout as arguments. Afterwards, the physical layer starts the transmission of the control packet.

check_pktRTS(): This function aims to transmitting a RTS packet. First, the channel is sensed before sending the packet. If the channel is busy, the congestion window (cw_) is doubled (with inc_cw()), and the backoff timer is started again. If the channel is idle, the tx_state_ of the MAC is set up to RTS and the function checkBackoffTimer() is called. In addition, timeout value is calculated, hence the MAC will know for how long it has to wait until the CTS confirms the packet. Afterwards, the function transmit() is called with RTS packet and the timeout as arguments. Finally, the physical layer starts the transmission of the RTS packet.

check_pktTx(): This function is used to transmit the actual data packet. If the channel is busy, sendRTS is called, which means that despite the RTS/CTS dialog, another node is using the channel. Otherwise, if RTS is not used, sendRTS will do nothing. Furthermore, the congestion window (cw_) is doubled with inc_cw() and the backofftimer is started, so the MAC will be idle until the other node finishes the transmission. If the channel is idle, tx_state is set to MAC_SEND and checkBackoffTimer is called. The timeout is calculated depending if it is a broadcast or not. If it is a broadcast packet, the timeout will be the transmission time of the packet. Otherwise, the timeout is just how long the MAC should wait until it decides the ACK was not delivered. Finally, transmit() function is called with data packet and timeout as arguments.

checkBackoffTimer (): This function checks two options. First, if the medium is idle and the backoff timer is already paused, it will continue the timer. Second, if the medium is busy and the backoff is running, then the function will pause the timer. Briefly, the MAC only counts down its backofftimer if the channel is idle; otherwise, the timer is not running.

transmit(): this function receives two arguments, the packet and the timeout value. If the MAC is already sending a packet (tx), then the function checks if the packet is an ACK packet to know when the node is receiving a packet. If so, the packet would be missed. In the case that the MAC is receiving a packet and an ACK packet is being transmitted, the packet received is marked as having errors. Afterwards, the packet is sent to the lower layer (downtarget_). Finally, two times are set up, the sender time with the timeout value (which alert the MAC when the transmission fails), and the interface timer, which advise the MAC when the packet has been transmitted.

send_timer(): At the expiration of the TxTimer, send_timer() is called. In this function, there are several options depending of the kind of packet to be treated. If the last packet sent was an

RTS and time expires, then a CTS was not deliver and retransmitRTS() is called (collision of RTS packet or deferring node). If the last packet sent was a CTS packet and time expires, means that no data packet was received, therefore the MAC just re-setting itself to an idle state. If the last packet was a data, means that ACK was not received and RetransmitDATA() is called. Finally, if the last packet was an ACK and time expires, means that ACK was transmitted without response.

After all checks, the packet is ready to transmission, and tx_resume() takes the control. If a packet is going to be retransmitted, the backoff timer is started with an increased congestion window.

retransmitRTS(): This function is called when a CTS was not received after a RTS packet. In this case, the MAC increment the ssrc_ (short retry count). When ssrc (short retry count) reaches the value of ShortRetryLimit in the MAC MIB, the MAC knows when to drop the packet. Dropping the packet is managed by the Discard() function. At this moment, the ssrc_ is reset to zero and the congestion window is reset to its established value. Otherwise, the pktRTS pointer to RTS is not modified, but a retry field in RTS is incremented. Finally, the congestion window is doubled and the backoff timer is triggered with the new congestion window.

retransmitDATA(): Once an ACK is missed after a data has been sent, this function aims to retransmit the data packet. If the data packet was a broadcast packet, an ACK is not expected and the transmission is treated as successfully. Two counters are used to manage depending if a RTS is used to send the data packet. If it is RTS, a short retry limit is used, otherwise a large retry limit is used. If the retry count exceeds the threshold, the data packet is discarded calling discard() function and the counter and congestion windows are reset. Otherwise, the data packet is ready to be sent, retry field is incremented in the mac header, congestion window is doubled, and backofftimer is started. Afterwards, control of flow returns to backoffHandler()

tx_resume() This function is called when the MAC is ready to send data, but needs to set more some timers. If a CTS or ACK packet is waiting to be sent, tx_resume() starts the defer time for a sifs_ amount of time (the time that a node have to wait before transmitting). In the case of a RTS packet, the MAC checks if the backoff timer is busy. If so, the MAC will wait to start the defer timer; otherwise, the defer time is started for a random time between the threshold [0,cw_) plus a difs time. In the case of a data packet, whenever the MAC is currently backing off, the defer time will start, but depending if it is used a RTS or not, the defer time is set as sifs_time for the first assumption (channel already reserved), and the interval [0,cw_) for the second assumption.

If there are no packets to be sent, but the callback_ is defined, then it is managed like successfully packet transmission. After all, tx_state_ is set to idle, control of flow returns the MAC after defer times expires (deferHandler()), or goes back to the caller function.

capture(): This function is called if the MAC, receiving one packet, is receiving another packet, week enough to be dropped. In this case, NAV is update, so it is well known that the channel is still busy. Afterwards, capture() also discards the captured packet.

collision(): First, this function checks the rx_state_ and sets it to MAC_COLL if this collision is the first collision during the current packet. If other packet collides, then rx_state_ will already be set as MAC_COLL. After that, the MAC checks for how much longer the new and the old packet will remain alive. If the new packet is the one that will remain alive longer, the MAC builds the new packet pktRx_ and resets the receive timer, and the old packet is discarder here. However, if the old packet will remain alive longer, then the new packet is just discarded and pktTx_ does not change.

recv_timer(): This function is one of the most important and is called when mhRecv_ expires, this means that the packet has been successfully delivered. First, the MAC checks with tx_active_ if there is a packet transmitting. In this case, the incoming packet is discarded without updating the NAV. After that, if rx_state=MAC_COLL, then pktRx is the colliding packet that remains longest and then should be discarded, setting the NAV for an eifs_ time. Then, the MAC checks the packet for errors, and discards it if any errors are detected, and NAV is again set for an eifs_ time. After this, MAC checks if the packet is for itself; otherwise, the NAV is updated looking in the duration field in the MAC header. Next check aims to sending the packet to any taps, if it is a data packet. Next two checks are the last operations of this function; the first one is to keep tracking of the nodes within the radio range of the node; and the second one refers to address filtering, where the packets that are not for the current node are discarded.

recvRTS(): this function is called by recv_timer() upon receiving a RTS packet. When the tx_state is not idle, it means the packet could be not heard, therefore is discarded. Additionally, if the MAC is responding to other node the RTS packet will be ignored; otherwise, the MAC is ready to receive a packet and is able to call sendCTS() function. Afterwards, the MAC stops the defer time, calls tx_resume() and flow of control returns to recv_timer().

recvCTS(): This function is called by recv_timer() after a CTS packet has been received. Since the RTS packet that is transmitting is not useful for the MAC, it is freed and pktRTS_ is

set to zero. Finally, the control goes forward to tx_resume() setting the defer time and flow of control returns back to recv_timer().

recvACK(): This function is also called by recv_timer after an ACK packet has been received, indicating a successfully data transmission. First, the MAC checks if the data that it sent is really a data packet, otherwise it discards the ACK packet. Since MAC knows the data packet was delivered, it frees pktTx_ and sets it to zero. Furthermore, the retry count is reset as short if an RTS was not used; and is reset as large if an RTS was used. Congestion window is also reset and the MAC starts its backofftimer to avoid send data again so soon. Finally, flow of control goes first to tx_resume() and then go back to recv_timer().

recvData(): This function is called by the recv_timer once a data packet was delivered, indicating that the transmission has been successful. First, the MAC takes out the header from the packet to be sent to the upper layers. If not broadcast was used, RTS was used; therefore tx_state_ indicates that the last packet the MAC sent was a CTS. Then, CTS is freed and pktCTRL_ set to zero. If the packet was not dropped, the MAC already has received the packet and is ready to send the ACK calling sendACK(), and then tx_resume() to start the defer time. If a CTS packet was not sent, because of the absence of RTS packet, then the MAC checks pktCTRL_. Checking the packet, one of two options can be chosen: If there is a control packet, the MAC will drop the data packet, otherwise sendACK() is called. After this, tx_resume() is called to start the defer time.

To avoid duplicate packets, MAC updates the sequence number and checks it against the last sequence number received. If they match, it means that the packet is duplicated and therefore is discarded. Finally, the data packet is sent to the upper layer (usually Logical Link Control sublayer),

rx_resume(): this function is called after recv_timer, it just set the rx_state_ to idle, and it calls checkBackoffTime() function.

backoffHandler(): This function is called when the backofftimer expires. First, the function checks if there is a control packet ready to send (CTS or ACK). If so, it also checks that the MAC is either sending the packet or deferring before sending the packet. If no control packet is found, check_pktRTS() is called. If there was no RTS packet, check_pktTx() is called. This means that when backoff time expires, RTS or data packet will be transmitted if one of them is waiting.

txHandler(): This function is a handler for IFTimer and it sets down a flag in the MAC to indicate that the radio is not longer active.

3.6 DSR in Network Simulator

3.6.1 Introduction

The Dynamic Source Routing protocol (DSR) is an efficient routing protocol designed for use in multi-hop wireless ad hoc networks of mobile nodes. DSR allows the network to be completely self-organizing and self-configuring, without the need for any existing network infrastructure or administration. DSR has been implemented by numerous groups, and deployed on several testbeds. This protocol was used as base for other protocols as well. Some implementations of the DSR are:

- The Click DSR Router Project at the PecoLab at UC Boulder [55].. A user level and open source implementation of DSR implemented in accordance with the IETF draft specifications of DSR. It is ready on Linux and a 802.11g Wi-Fi card
- The Microsoft Research Mesh Connectivity Layer (MCL) MCL [56]: It implements a layer between link layer and network layer multi-hop routing protocol on Windows XP. This protocol is derived from the DSR protocol and is called Link Quality Source Routing (LQSR) protocol that means that DSR was widely modified.
- The Monarch Project implementation [57]: It is a set of kernel patches that supports FreeBSD 3.3 and 2.2.7. It is a pre-alpha release and is available because of educational purposes and for researchers.
- Alex Tzu-Yu Song [58] has implemented DSR according to the fifth draft of THE IETF DSR.
- The National Institute of Standards and Technology (NIST) [59] developed a simulation model for the DSR for MANETs based on the Internet Draft version 4 with the purpose of using it at OPNET in their communications system simulation software.

Since networks using the DSR protocol have been connected to the Internet, DSR can interoperate with Mobile IP. Nodes using Mobile IP and DSR have seamlessly migrated between WLANs, cellular data services, and DSR mobile ad hoc networks [50].

3.6.2 DSR Operation in NS-2.

The DSR agent checks every data packet for source-route information [51]. It forwards the packet in accordance with the routing information. If the DSR agent does not find routing information in the packet, one of two options can be followed. If the route is known, it provides the source route. Otherwise, if the route is unknown, it caches the packet and sends out route queries. Route queries messages are broadcasted to all neighbours whenever there is no route to reach the destination. Then, route replies messages are sent back either by the intermediate nodes, if they have such path in their cache, or by the destination node. The source files for implementation of DSR protocol in ns-2 belongs to the ns-2/dsr directory. More details can be found in tcl/mobility/dsr.tcl.

3.6.3 The DSR functions behaviour implemented in ns-2 network simulator

The DSR strategy for ns-2 network simulator was ported from the CMU/Monarch's code and it is as follows [47]:

- It is only worth discovering bidirectional routes, since all data paths must be bidirectional to work properly for 802.11 ACKs.
- Reply to all route requests in destination nodes, but reply to them by reversing the route and unicasting. Then, do not trigger a route request. By reversing the discovered route for the route reply, only routes that are bidirectional will make it back the original requestor.
- Once a packet goes into the sendbuffer, it cannot be piggybacked on a route request. The code assumes that the only reason that removes packets from the send buffer is the StickPktIn routine, or the route reply arrives routine

The OTcl variables that the programmer wants to use in the source code must follow a binding process to get the expected results. That is handled in the class creation. The variables bound to the OTcl variables must be also declared in the C++ header files, and in the tcl main file, where all the OTcl variables are stored to be bound into the C++ code.

Packets

The first step when a packet is received by DSR, the method ***recv(Packet* packet, Handler*)*** handles mainly packets which MAC address matches current host or MAC broadcast address. First, it checks if the packet has a source route checking then, if the destination of the packet is the node's net_id or the broadcast address in order to forward finally the packet to

handlePacketReceipt(p). Otherwise, the packet is classified among route request, calling ***handleRouteRequest(p)***, route error ***processBrokenRouteError(p)***, packet to be forwarded, ***handleForwarding(p)***, or invalid packet, where the packet is dropped silently.

If no source route is present, it should be a broadcast packet. In this case, it is checked to know whether an outgoing or incoming broadcast packet is. If it was not a broadcast packet (***sendOutBCastPkt(packet)***), it must be an outgoing packet, and DSR gives to the packet a SR header, calling to ***handlePktWithoutSR(p, false)***.

Therefore, here it is possible to modify packets to be forwarded, inserting code before the call to handleForwarding(p) or placing code within the handleForwarding method.

handlePacketReceipt()

This method handles packets that have as destination the current node. The first step performed by DSR is to check if the packet is a route_reply, accepting in this case the new source route with the method ***acceptRouteReply(p).*** If the received packed contains a route request, DSR respond once for each host, calling to the method ***returnSrcRouteToRequestor(p)***.

The received packet may contain a route error, then the dead route is registered using ***processBrokenRouteError(p)***. Later, the packet is given to the higher layer.

Ns-2 allows the programmer to drop packets intentionally using the Error Model.

A packet is created giving the sr and ip headers after assigning the id node. It is sent using the scheduler: Scheduler::instance().schedule(ll, p.pkt, 0.0); specifically, this code schedules the packet p.pkt to be sent immediately (in 0.0 seconds) to the link layer output.

```
SRPacket p;
p.src = net_id;
p.pkt = allocpkt();

hdr_sr *srh = hdr_sr::access(p.pkt);
hdr_ip *iph = hdr_ip::access(p.pkt);
hdr_cmn *cmnh = hdr_cmn::access(p.pkt);
```

Routes

The available routes are stored in primary_cache and secondary_cache in mobicache.cc. The routes can be added as the result of receiving a route reply (after sending a route request), or overhearing a route used (or routing information) in a packet destined for another node. If the programmers want to change the metric for route selection or to use multipath, they should do that in mobicache.cc

Routes discovered from a route reply are added to the primary cache.

```
void MobiCache::addRoute(const Path& route, Time t, const ID& who_from) {
  (void) primary_cache->addRoute(rt, prefix_len);
}
```

Routes discovered by overhearing a packet are added to the secondary cache.

```
Void Mobicache::noticeRouteUsed(const Path& p, Time t, const ID & who_from){
    (void) secondary_cache->addRoute(stub,prefix_len);
}
```

That is the way to know the next hop for a packet:

```
srh->get_next_addr() or cmh->next_hop()
```

The searchRoute(..) method, selects routes from the cache that fulfil the requested destination. It is called by findRoute(..)

Each time searchRoute(..) discover a route towards the destination, findRoute(..) checks if it is the shortest known route found so far.

Code into findRoute(..) in dsragent.cc:

```
while (primary_cache->searchRoute(dest, len, path, index))  {
```

```
    min_cache = 2;
    if (len < min_length) {
        min_length = len;
        route = path;
    }
  index++;
  }
```

"len" is the length of the route just found. "path" is the route found by seachRoute(..). "route" is the route passed to findRoute(..) by the caller. "index" is used to monitor the route cache checked..

The secondary cache is treated in a similar manner. One possible way, in order to purge these caches, could be

```
delete primary_cache;
delete secondary_cache;
```

The route of a packet is stored in srpacket.h

```
struct SRPacket {
        Path route;
}
```

The srpacket can be examined or altered using some methods in ***the hdr_sr.h***, ***path.cc*** and ***path.h***. For example, the route of a packet can be viewed ***p.route.dump()*** (where p is a SRPacket).

The route for a packet can be altered, for a SRPacket p:

```
p.route = new_route;
   p.route.resetIterator();
cmh->size() -= srh->size();
p.route.fillSR(srh);
```

That is the type of a packet ***cmh->ptype()*** and it is represented by a string. In order to know which packet is currently being executed, this is the code ***net_id.dump(),*** also used by the trace mechanism.
Packets, before being delivered to the MAC layer, are placed in the cmu-priqueue class according to their priority.

The DSR implementation of ns-2 offers the possibility to add comments or information into the trace. Into the trace() function, programmers can add the next line in order to see in the produced trace the information they want.

trace("NewEvent %.5f _%s_ %s",Scheduler::instance().clock(),net_id.dump(),
relatedInformation);
Note that the user can modify this line adding/removing these parameters.

In dsrgent.cc, important features related to DSR behaviour are implemented, such as processBrokenRouteError(SRPacket& p) or getRouteForPacket(SRPacket &p, bool retry). Furthermore, in the xmitFailed()function is implemented how the DSR protocol reacts against link failures received from the MAC layer.

xmitFailed(Packet *pkt, const char* reason) In this method the route cache is marked as failure of the link between: srh[cur_addr] and srh[next_addr], sending back the created route err message to the originator of the packet.

This method may be called from ***sendOutPacketWithRoute(..)*** that responds to the following methods: sendOutRtReq, returnSrcRouteToRequestor, acceptRouteReply, sendRouteShortening, undeliverablePacket, handlePktWithoutSR, handleForwarding, handleRouteRequest or replyFromRouteCache; ***xmitFlowFailed()*** because of these calls: handleFlowForwarding(..)->xmitFlowFailureCallback; or ***xmitFailureCallback(..)*** called from sendOutPacketWithRoute.

3.7 Radio propagation models

3.7.1 Introduction

In the physical layer of the wireless model (in ns-2), there is a value called the receiving threshold used to compare against the power of the packet arriving. When a packet is received, if its signal power is below the receiving threshold, it is marked as “error” and dropped by the MAC layer. The radio propagation model aims to predicting the receive signal power of each packet.

In ns-2, we may differentiate among three radio propagation models: free space model, two-ray ground, and shadowing model [11].

3.7.2 Free space model

In the free space model, it is assumed that there is only one clear line-of-sight path between the sender and the receiver. The following equation is used to calculate the received signal power free space at distance d from the sender.

$$\Pr(d) = \frac{P_t G_t G_r \lambda^2}{(4\Pi)^2 d^2 L} \quad (1)$$

Table 5: Formula 1 explanation

Pt	Transmitted signal power
Gt	Antenna gains of the sender
Gr	Antenna gains of the receiver
L(L>=1)	System loss
Lambda	Wave length

In ns-2, it is common to establish Gt=Gr=1 and L=1. To configure this model in ns-2 we can use the node-config command as follows:

```
$ns _node-config –proType Propagation/Freespace
```

3.7.3 Two-ray ground reflection model

The two-ray ground reflection model focuses on the direct path and a ground reflection path, instead of focusing only in the path. It is demonstrate that this model offers better performance in long distance among nodes than the free-space model. The received power at distance d is predicted by

$$\Pr(d) = \frac{P_t G_t G_r ht^2 h_r^2}{d^4 L} \quad (2)$$

where the new parameters are ht and hr, and L=1 is assumed as in the free space model.

Table 6 : Formula 2 explanation

ht	Height of the sender antenna
hr	Height of the received antenna
Gt	Antenna gains of the sender
Gr	Antenna gains of the receiver
d	Distance
L(L>=1)	System loss

The two-ray model does not achieve good results in short distances due to the oscillation caused by the constructive and destructive combination of the two rays. Therefore, from a theoretical point of view, the free space model shows better performance in short distances (d small). Actually, the two-ray ground reflection model in ns-2 calculates a crossover distance, d_c. When $d < d_c$ two-ray model is used. Otherwise, free space is used.

$$dc = (4\Pi h_t h_r) / \lambda \quad (3)$$

To configure these nodes in ns-2, we can use node-config as follows:

```
$ns_ node-config –propType Propagation/TwoRayGround
```

3.7.4 Shadowing model

The nodes ranges are not ideal circles as free space model and two-ray model assumes. In reality, the received power depends from the multipath propagation effect, hence the received power at certain distance is a random value. This model is called the shadow model.

Table 7: Typical path loss exponential values

Environment	β
Outdoor; free space	2
Outdoor; shadowed urban area	2.7 - 5
Indoor; line-of-sight	1.6-1.8
Indoor, obstructed area	4 - 6

Table 8: Typical shadowing deviation values

Environment	σ_{dB} (dB)
Outdoor	4 - 12
Office, hard partition	7
Office, soft partition	9.6
Factory building, line-of-sight	3 – 6
Factory building, obstructed	6.8

$$\left[\frac{P_r(d)}{P_r(d_0)}\right]_{db} = -10\beta\log\left(\frac{d}{d_0}\right) + X_{dB} \quad (4)$$

Table 9: equation explanation for formula 4

Pr(d)	Received signal power at the distance d
$Pr(d_0)$	Received signal power at the reference distance d_0
X_{dB}	Gaussian random variable with zero average and standard deviation δ_{dB}
ß	path loss exponent

In ns-2, before using the model we should select the values of the path exponent ß and the shadowing deviation $\delta_{dB.}$

In addition, the Otcl interface uses the node-config command. An example is showed as follows:

```
# first set values of shadowing model
Propagation/Shadowing set pathlossExp_ 2.0 ;# path loss exponent
Propagation/Shadowing set std_db_ 4.0 ;# shadowing deviation (dB)
Propagation/Shadowing set dist0_ 1.0 ;# reference distance (m)
Propagation/Shadowing set seed_ 0 ;# seed for RNG

$ns_ node-config -propType Propagation/Shadowing
```

The code above sets a random number generator (RNG) for the seed value. Another seed value can be set as follows:

```
set prop [new Propagation/Shadowing]
$prop set pathlossExp_ 2.0
$prop set std_db_ 4.0
$prop set dist0_ 1.0
$prop seed <seed-type> 0 ;# user can specify seeding method
$ns_ node-config -propInstance $prop
```

Where <seed-type> above can be raw, predef or heuristic.

4 Cross Layer Design for MANET

4.1 Introduction to cross layer design

The success of layered architecture lies in its property to provide modularity and transparency [34]. Modularity refers to the high level of uniformity of the method, and transparency refers to the clear separation between the functions of each layer. The modularity can be highly affected by the cross-layer design, since changes into the layer implementation can affected the stability of the whole network due to unexpected collateral effects. The cross-layer design leads to improve the overall performance of the network in a low coupling manner, where low coupling refers to the desired low level of interdependency between the method modified and the rest of the methods or classes. This chapter aims to explain the modifications proposed in the MAC and routing layer with respect to cross layer design. The goal is that by achieving better coordination among the layers, better performance can be achieved.

The main problem between MAC and routing layer is that the routing layer misunderstands the information provided by the MAC layer. When a sender node attempts to communicate with another neighbouring node, the communication may fail. This unsuccessful communication may happen because either the receiver node is unreachable by the sender node (broken link), or there is collision of packets at the receiver node. However, the routing layer always interprets this failure as broken link. Our purpose in this project is to clarify for the routing layer that information in order to know if the node is still reachable. If so, the receiver node does not need to send back the route error towards the sender node.

4.2 Signal to noise ratio

4.2.1 Introduction

Signal to noise ratio (SNR) is the ratio of the signal power to the noise level of a transmission medium, and it is used to categorize the quality of a transmission [35]. The cross layer concept is also related with SNR, since upper layers use the channel information obtained from the physical layer to improve the overall performance of the network.

4.2.2 Operation

Consider figure 10 [43]. The SNR of a link is obtained in the physical layer. The SNR_i helps the sender to choose an adequate transmission rate R_i , where this rate affects, as well, the packet delay D_i of the link. Additionally, the rate is considered by the routing protocol in its decisions, which affect the overhead of the network, and influence the values D_i, R_i, and SNR_i on individual links.

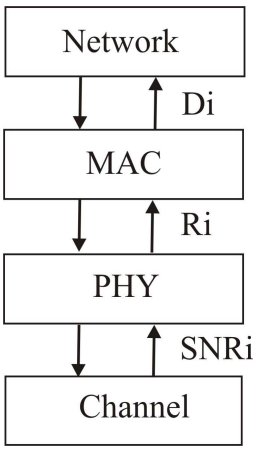


Figure 7: Cross layer communication

An example of how to improve the network throughput in wireless networks using SNR is described by Velayos&Karlsson in techniques to reduce the 802.11 handoff time [40], which is explained as follows. Link-layer handoff is the change of the access point (AP) to which a station is connected. However, in wireless IEEE 802.11, the handoff refers to the actions that interrupt the transmission of data, such as change of radio channel or exchange of signalling messages. The signalling to perform the handoff is specified by the 802.11 MAC protocol Signal strength is an important measure to care about before launching the handoff, since the search phase of the handoff could start when the strength of the received signal radio is below a certain threshold.

The detection phase of the handoff can be launched by the network, consisting of a single disassociation message sent by an access point to the station. However, normally the handoff is launched by a station, upon detecting a lack of radio connectivity based on weak signal received from the physical layer.

In the analysis of the detection of failed frames, the reasons can be collision, radio signal fading, or station out of range. The stations firstly consider collision, but after several unsuccessful retransmissions, radios signal fading is assumed and probe requests are sent to

check the link. Only after several unanswered request, out of range status is assumed and the start phase is called.

Once the detection is completed, the search phase of the handoff is performed. In the search phase, the main purpose of the station is to find the access point in the range. Afterwards, the last step is the execution of the handoff, where the station sends a re-association request to the new access point. Afterwards, the access point confirms the reassociation sending back a successful request.

Additional improvements are achieved from the signal to noise ratio in cellular wireless networks [41]. In IEEE 802.11, an access point serves an area of a limited radio range. When the capacity of an access point is not enough for the covered area, more access points can be installed. However, it does not mean that the capacity available of the network increases automatically, because the stations usually select the access point with the strongest signal to noise ratio. As the stations tend to concentrate on the access point closest, a load balancing mechanism is necessary

The signal strength is also an important feature in the range estimation techniques [39]. These techniques were extensively used in outdoor wireless, such as GPS. In indoor wireless, the range estimation techniques are less developed due to the radio propagation, which provokes a limitation in the range estimation techniques. The received signal strength is the principal option for range estimation in wireless networks. The energy of the signal tends to decrease from the access point towards the receiving station. The range to the AP can be deduced if a station measures the received signal strength, and knowing how the signal strength reduces with distance (i.e. the propagation model).

Signal strength is also helpful to predict link quality in ad hoc wireless networks [6]. It is observed that the successful reception of a packet depends on SNR at the receiver. Shortest links with high signal strength in the receiver node produces higher delivery rates than longer links, which achieves low signal strength.

4.3 Extensions to MAC 802.11 for cross layer design

4.3.1 Introduction

In ad hoc wireless networks, we deal with nodes that have different power capabilities; hence, there is a considerable likelihood to transmit with different power levels. The link asymmetry arises when a node with high power transmits to a lower power node, and the high power node cannot sense an ongoing transmission triggered by the low power node. Because of the asymmetry, traditional protocols that assume links as bidirectional perform poorly. Concretely, the hidden terminal problem is exacerbated.

IEEE 802.11 MAC protocol offers a lower throughput in the presence of link asymmetry, which influences on routing protocols primarily designed for wireless ad hoc networks with bidirectional links, such as DSR. The principal cause of this problem is that high power nodes cannot sense the RTS/CTS dialog among low power nodes. Thereby, the hidden terminal problem is exacerbated, which provokes more false link failures, increasing the times that route discovery process is launched [33].

4.3.2 Challenges and concept

The link asymmetry problem is a good example concerning bad interaction among layers. Similarly, other parameters can improve the performance in the cross layer design from the MAC layer. Using RTS/CTS communication among nodes, a new problem arises: a sender node attempting to establish a communication with other nodes receives no answer, because the destination node is communicating with other node. Then, the sender node stops trying to communicate after a while, and the MAC layer sends to the upper layer that the communication is not possible. The main objective in this master's thesis project for the MAC layer is to extend that information, clarifying to the upper layer why the communication is not possible.

4.4 Extensions to DSR for cross layer design

4.4.1 Theoretical Viewpoint

In the cross layer design, the modifications in the MAC layer affect to the routing protocols as well. Moreover, direct modifications in the routing protocol affect to the MAC layer, improving the whole performance of the network if these modifications are good enough. For example, the behaviour at MAC layer for a proactive routing protocol, sending hello messages periodically, will be completely different from for a reactive routing protocol, since proactive routing protocols keep routes updated in each node, even if the routes are not required and reactive routing protocols update routes on demand, when the path is needed. In the DSR protocol, a better interpretation of the information obtained from the MAC layer may achieve several improvements in the throughput, delay of packets to the destination, and packet delivery ratio [43].

4.4.2 Challenges and concept

A bad interaction between DSR protocols and the MAC layer could decrease the throughput in wireless 802.11 networks [28]. Once the MAC layer has been improved with a table of received powers of neighbouring nodes with the purpose tracking their distances to the sender node, our goal in this thesis is to handle that information in the routing layer to avoid that DSR triggers a route error process or route maintenance if it is not required.

DSR protocol triggers a route error when the receiver node of the communication did not reply after several attempts of RTS in the MAC layer. Consequently, DSR assumes ‘link error’, manages that link error as broken link and triggers the route maintenance process. However, as we discussed earlier in chapter 4.2, unsuccessful communication among nodes may arise because of different reasons than broken links. In such case, the route maintenance process is not necessary when a neighbouring node is still reachable. A cross layer design should identify when a link error was due to broken link (node not reachable), triggering the route maintenance process only in such case, or other reasons such as increased contention.

4.5 Implementation

The implementation and later results presented in the next section are referred to:

- DSR-AR: it refers to the results using the modification proposed in this dissertation from DSR simulations. 'A' comes from Alonso and 'R' from Rocha (our surnames).
- DSR β1: it refers to the original implementation of DSR.
- DSR β2 (Dampen policy): it refers to the improvement proposed by Nahm et al. [28][29].

Nahm et al. use a new parameter to control the stability of the connections. DSR Routing maintenance process is called only after the number of successive link failure L exceeds a certain limit β, which indicates the toleration limit of successive link failures.

- If transmission fails and L < β, the current packet is dropped and the current route is kept. Thereafter, L is increased by 1.
- If transmission fails and L = β, DSR routing maintenance process can be launched and L is reset to 0.
- If transmission is successful, L is reset to 0.

Usually, the routing protocol responds to link failure after RTS packet is retransmitted up to 7β times in the 802.11 MAC layer. Therefore, β=1 refers to the original policy of DSR protocol.

4.5.1 Extension to ns-2 - MAC layer

The main reasons for lost packets are high levels of congestion (high traffic), equipment failures (power problems) or errors due to noise or low signal (mobility). It would be beneficial for DSR the ability to distinguish these reasons, adjusting transmission rate in case of congestion, or deleting old routes in case of mobility.

Normal DSR interprets a link failure (in MAC layer) as a broken link, even when it was caused by congestion at receiver. The sender node should know why communication was not possible. In this master's thesis project, we implemented an approach that tracks the received signal strength of each neighbouring node in order to know when a neighbouring node is near enough for a successful transmission. If lost packets are due to congestion and high traffic, normal DSR triggers route error but this is counterproductive because it adds more. If lost packets is due to low signal quality or misrouted packets, then route error is needed because receiver is not reachable. Then, the signal strength of neighbouring nodes can be used to detect the reason for lost packets, distinguishing between congestion and broken links due to mobility, because in broken links due to mobility, the receiver is not reachable and its signal strength is now available. The implementation is divided into two parts; the first one keeps the last twenty received signals from a node in an array, and the second part decides the kind of

message (link failure, either due to errors or due to congestion using signal strength of neighbouring nodes) to be sent to the upper layer, whenever the communication is not possible but the destination node is in the transmission range of the sender.

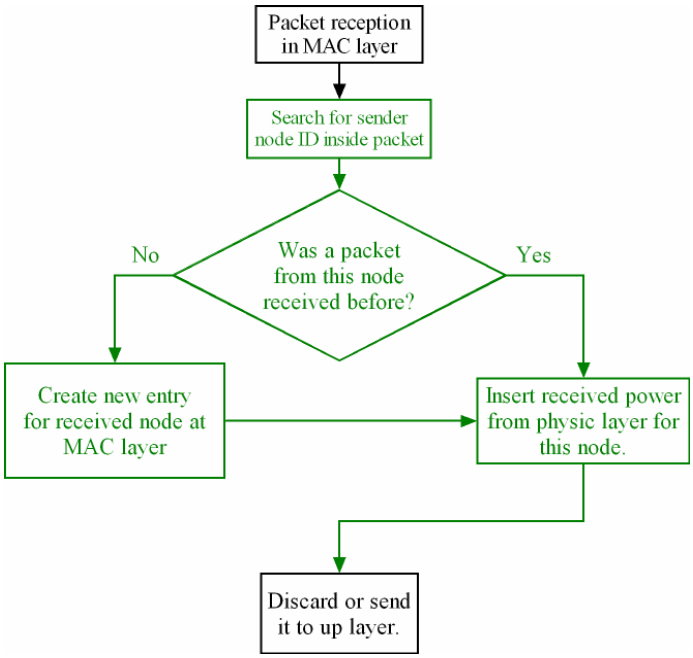


Figure 8: MAC layer at receiver node. A packet is received even when current node is not the destination node. See Code 1.

In the graphic above, the received power of neighbouring nodes is tracked with the purpose of using it later for distinguishing if neighbouring nodes are reachable or not. These modifications are made at the sender.

```
void
Mac802_11::recv(Packet *p, Handler *h)
{
. . .
   if(tx_active_ && hdr->error() == 0) {
      hdr->error() = 1;
   }

   hdr_mac802_11 *mh = HDR_MAC802_11(p);
   u_int32_t idNode = ETHER_ADDR(mh->dh_ta);
   double power = p->txinfo_.RxPr;
   insertNode (idNode,power,nodesPower);
. . .
}
```

Code 1: Modifications at "mac802-11.cc" when a packet is received from physic layer.

```
struct time_power {
   int pos;
   double power[20];
   u_int32_t idNode;
};
```

Code 2: Modifications at MAC layer (in the file mac802-11.h) where the last 20 received signal strength from neighbouring nodes are stored.

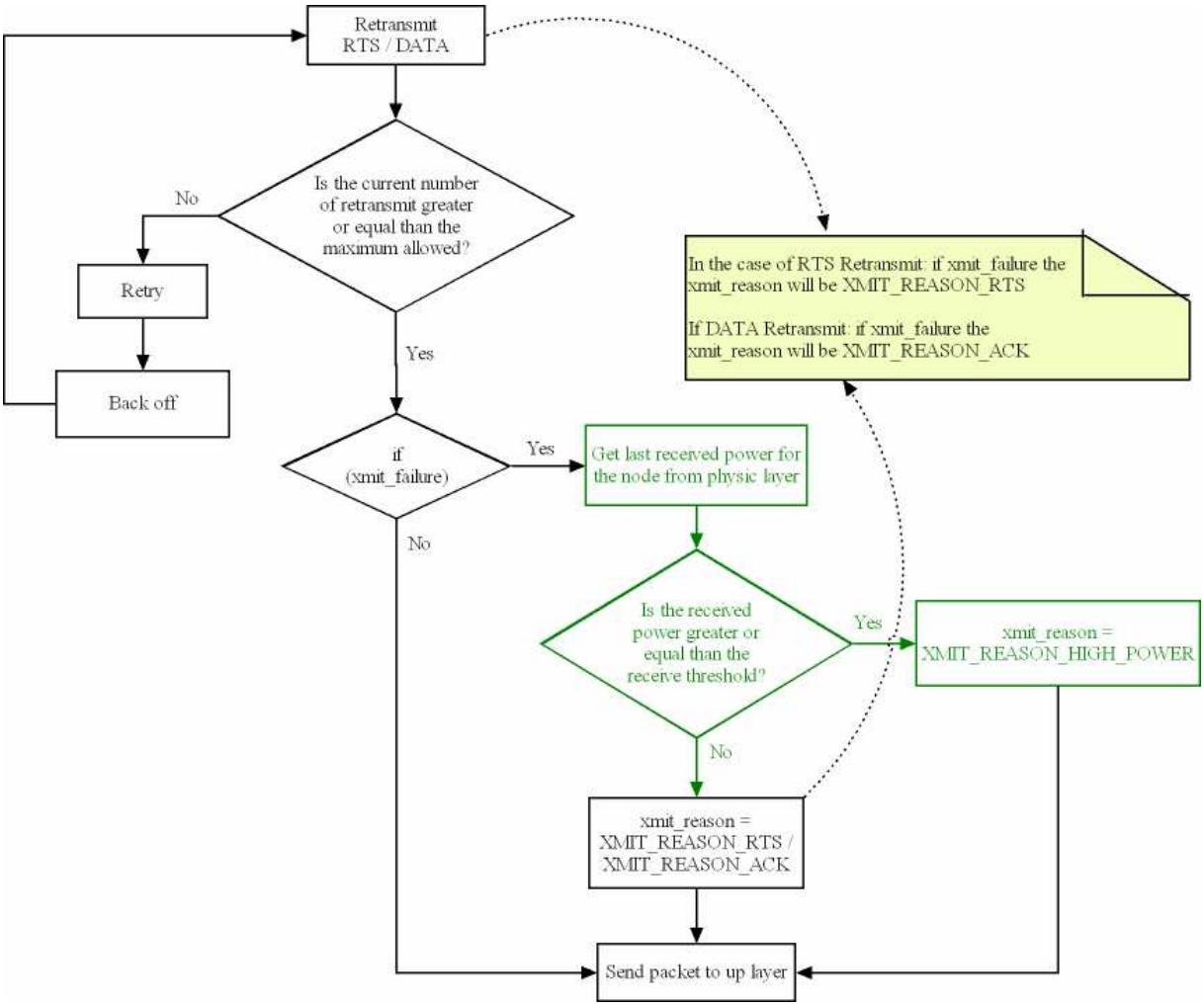


Figure 9: This diagram shows how MAC layer informs to the routing layer, when several attempts to communicate to the receiver node failed. See Code 3.

The normal behaviour of MAC layer in order to transmit information to a neighbouring node is to send a Request To Send (RTS). If this communication fails, the MAC layer waits (back off time) and tries it again later. After several and unsuccessful attempts, the MAC layer informs to the routing layer that communication was unsuccessful. In our dissertation, the reason for that unsuccessful communication is sent to the routing layer, that is, if the last received power of the destination node indicates that it is reachable, the routing layer is

informed, using the variable xmit_reason with the value XMIT_REASON_HIGH_POWER (See Code 3). In this case, the routing layer should interpret that communication to destination was not possible, not because of a broken link but rather congestion, therefore route maintenance is not needed. If that is not the reason delivered to the routing layer, a route maintenance process is required.

```
if (ch->xmit_failure_) {
      struct hdr_mac802_11* dh = HDR_MAC802_11(pktTx_);
      uint32_t idNode = ETHER_ADDR(dh->dh_ra);
      double received_Power = pktTx_->txinfo_.RxPr;
      int i = findNode(idNode, nodesPower);
      if (i != -1)
         received_Power = nodesPower[i].power[nodesPower[i].pos];
      if (received_Power > RxThreshold )
      {
         ch->size() -= phymib_.getHdrLen11();
         ch->xmit_reason_ = XMIT_REASON_HIGH_POWER;
         ch->xmit_failure_(pktTx_->copy(),ch->xmit_failure_data_);
      }
      else {
         ch->size() -= phymib_.getHdrLen11();
         ch->xmit_reason_ = XMIT_REASON_RTS;
         ch->xmit_failure_(pktTx_->copy(),
         ch->xmit_failure_data_);
      }
```

Code 3: Modifications in the file "mac802-11.cc", link error notification. See Figure 9.

```
(...)
  int     xmit_reason_;
#define XMIT_REASON_RTS 0x01
#define XMIT_REASON_ACK 0x02
// ktnahm add for DAMPEN policy
#define XMIT_REASON_CONFIRM 0x03
// Alonso - Rocha
#define XMIT_REASON_HIGH_POWER 0x04
(...)
```

Code 4: Modifications in the file "packet.h", definitions of link error.

The proposed approach for future work that we designed adds new functionalities for every node in the network. Since each neighbouring node is tracked via its received signal strength, a sender node is able to discern if neighbouring nodes are moving away or not. In addition, we

propose to calculate and use the average signals of nodes with the purpose of stop retransmitting packets when destination is not reachable because it moved away.

The reason of using an average value, instead of only the last received value from neighbouring nodes, is to adapt this approach to more realistic scenarios, where objects such as furniture, may interfere temporally in communication among mobile nodes.

```
if (average_selected){
         u_int32_t idNode = ETHER_ADDR(rf->rf_ra);
         int pos =findNode (idNode, nodesPower);
         if (pos!=-1){
            float av=0; int movAw=0;
            average(nodesPower[pos],av,movAw);
            if ( (movAw >= 18) && (av < RxThreshold) ) {
                  discard(pktRTS_, DROP_MAC_RETRY_COUNT_EXCEEDED); pktRTS_ = 0;
                  hdr_cmn *ch = HDR_CMN(pktTx_);
                  if (ch->xmit_failure_) {
                     ch->size() -= phymib_.getHdrLen11();
                     ch->xmit_reason_ = XMIT_REASON_RTS;
                     ch->xmit_failure_(pktTx_->copy(),ch->xmit_failure_data_); }
                  discard(pktTx_, DROP_MAC_RETRY_COUNT_EXCEEDED);
                  pktTx_ = 0; ssrc_ = 0;rst_cw();
                  return;}
         }
}
```

Code 5: File "mac802-11.c": use of average for retransmission of packets.

An accurate study for the number of repeated movements of neighbouring nodes moving away and for the value of the average are needed, in order to achieve an optimal performance in the network. See also source code in Appendix C.

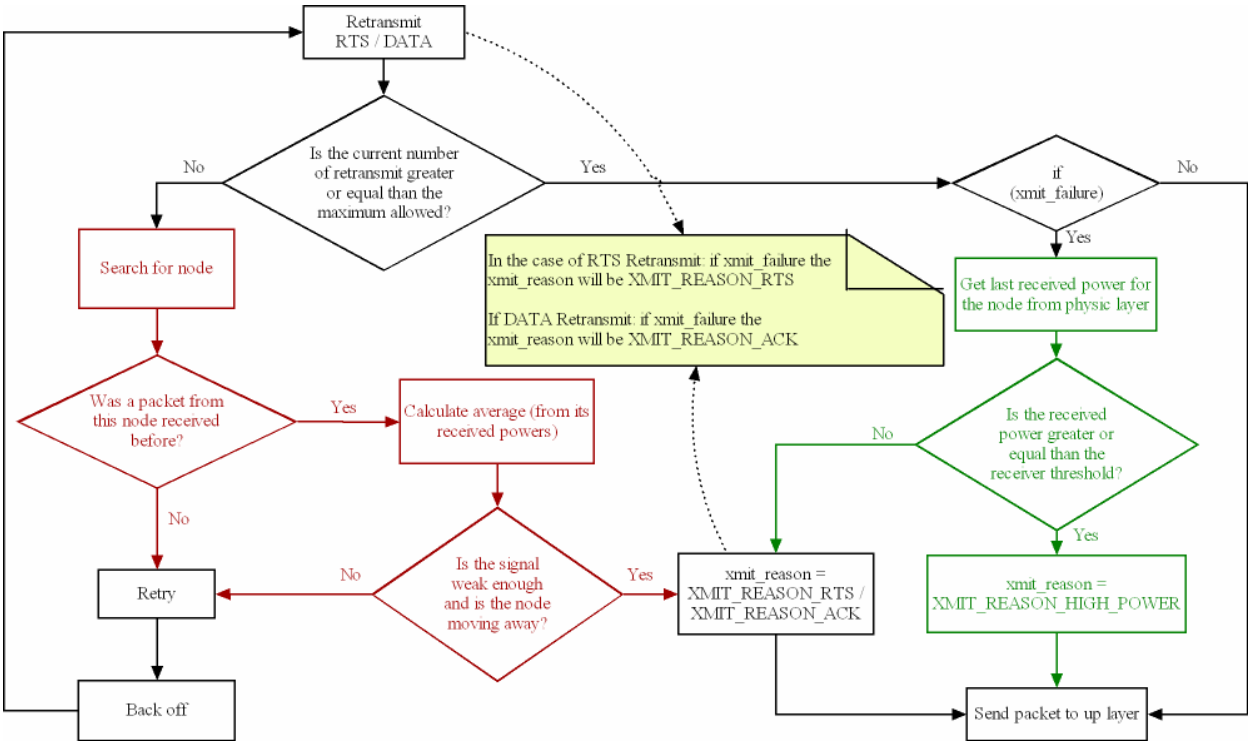


Figure 10: The proposed approach that uses an average value is shown here, to improve performance with object scenarios.

4.5.2 Extension to ns-2 - DSR

When a node tries to communicate to a neighbouring node and this communication is not successful (after several attempts, MAC layer sends an error to the routing layer), the normal behaviour of DSR is to interpret that the neighbouring node is not present anymore, that is, DSR assumes that the communication failure was due to mobility. Once the DSR protocol at sender interprets that link error as a broken link, which information is delivered for each intermediate node back to the sender node, which receives that information as well. A route error may trigger a route request process (processBrokenRouteError or handlePacketWithoutSR), if the sender does not have an alternative route towards the destination, or there is an alternative route towards the destination, which will be used. In both cases, the route containing the broken link is removed from the caches of the nodes involved in the route error.

In a scenario without mobility communication failures may arise, but DSR will interpret that it was due to mobility, when actually it was due to congestion. Therefore, the process of route error should not be performed since it increases even more the congestion, decreasing the overall performance in the network. The proposed modifications will make DSR able to distinguish between both situations, avoiding the route error process when the link error at

MAC layer was due to congestion and not due to mobility of nodes causing broken links. In the proposed approach, when MAC layer is not able to communicate to a neighbouring node, MAC layer informs to the routing layer not only that there was a problem, it is also included if the neighbouring node is still reachable (see Code 4). Therefore, DSR at the sender realizes about the real situation and do not perform a route error if the error received from MAC layer informs that the node is still reachable.

The DSR modification is showed as follows:

```
void DSRAgent::xmitFailed(Packet *pkt, const char* reason)
{
(...)
        if ( cmh->xmit_reason_ == XMIT_REASON_HIGH_POWER
                && !strcmp(reason, "DROP_RTR_MAC_CALLBACK")) {
                Packet::free(pkt);
                pkt = 0;
                return;
        }
 (…)
  /* send out the Route Error message */
  sendOutPacketWithRoute(p, true);
}
```

Code 6: File "mac802-11.c": use of average for retransmission of packets.

5 Simulation Results

5.1 Scenario Description

In this chapter, we describe the number and spatial location of nodes and their configuration parameters as such transmission range, carrier sense threshold, etc. that we utilized in our simulations. The time of the simulation, the transport layer protocol and number of flows are also described.

In our simulations, we utilized four different scenarios, with the purpose of evaluating diverse behaviours of mobile ad hoc networks. First, we distinguish between static and mobile scenarios. We used static scenarios to analyze and to show communication in the network, avoiding changes of routes or disconnections due to mobility of nodes. Later, we use mobile scenarios to verify the adaptability of our approach in mobility and high mobility scenarios. The first scenario we deal with is called “Chain” and all static nodes are positioned describing a beeline. The communication from a source node to a destination node can use intermediate nodes and alternative routes are not possible due to the separation among nodes. Therefore, in this scenario the number of hops is an important parameter. We utilized Grid scenario to evaluate how collisions affect overall network performance. In this scenario traffic is generated with the purpose of create collisions, while number of hops does not change. For mobility scenarios, we used “Random way point” scenario where the speed and direction of nodes is randomly set and “Manhattan” scenario, where nodes move along streets and speed set to a specific value for each simulation of this scenario.

5.1.1 Static Scenarios

Chain scenario

The chain scenario defines 22 nodes without mobility forming a chain with a separation of 200 meters. The carrier sense threshold for each node is 550 meters and the transmission range is set to 250 meters.

The simulation results were obtained from two, four and eight simultaneous flows for both TCP and UDP protocols. The number of hops to reach the destination varies among 4, 5, 6, 7, 8, 10, 12, 14, 16 and 20 hops. The source is always the same, where the destination depends on the number of hops to simulate. For instance, four TCP flows and 7 or 12 hops means that

there were simulated four TCP flows from node 0 (source) to node 7 or node 12 respectively (see figures 11, 12, 13). This scenario was designed to demonstrate the improvements in TCP when the fractional window [29] is used. Simulation time for this was set to 120 seconds.

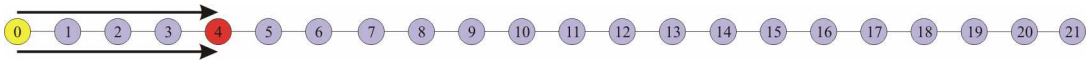


Figure 11: Two flows from node 0 to 4, example for 4 hops

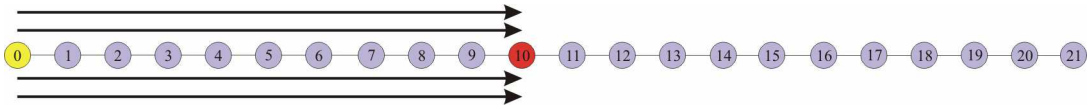


Figure 12: Four flows from node 0 to 10, example for 10 hops

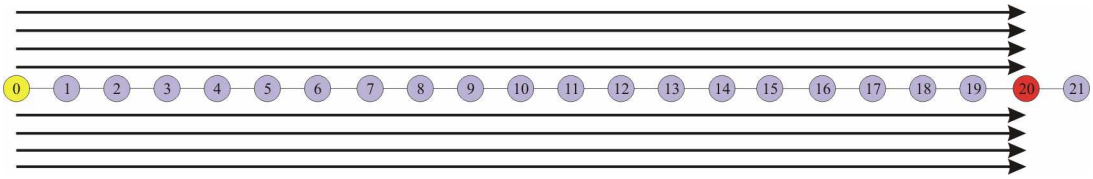


Figure 13: Eight flows from node 0 to 20 example for 20 hops

Grid7x7 scenario

The grid scenario is composed by 49 nodes situated in a square forming 7 rows and 7 columns. Each node is separated 200 meters from its neighbours. The carrier sense threshold is 550m and the transmission range is 250m for each node. As the chain scenario described above, this is a static scenario where the nodes have no mobility.

We have been simulated two parallel flows, TCP and UDP, from node 21 through node 27 with this scenario. In order to check how collisions were managed, they were induced on purpose, with intersected flows, from node 3 to 45 and from 21 to 27 at the same time. We decided to increase the number of flows until reach the maximum of intersections in the grid, from two vertical flows intersected with two horizontal flows until seven flows intersected with seven flows. Simulation time for this scenario was set to 120 seconds.

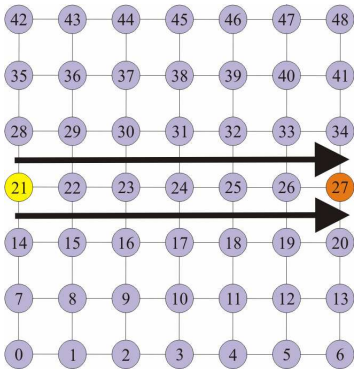


Figure 14:two parallel flows

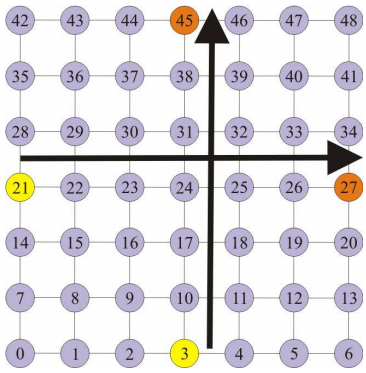


Figure 15:two flows

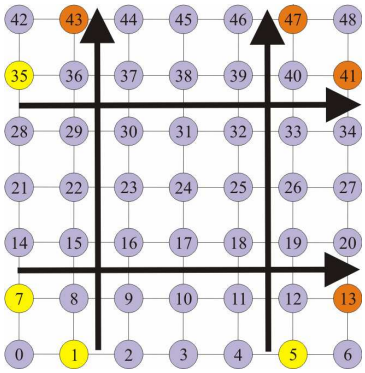


Figure 16:2x2 flows

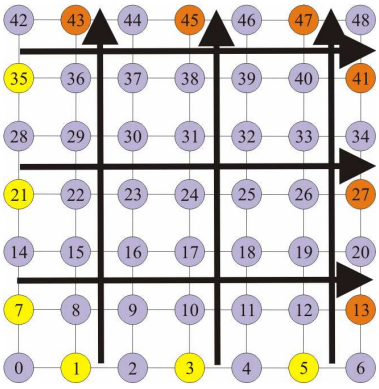


Figure 17:3x3 flows

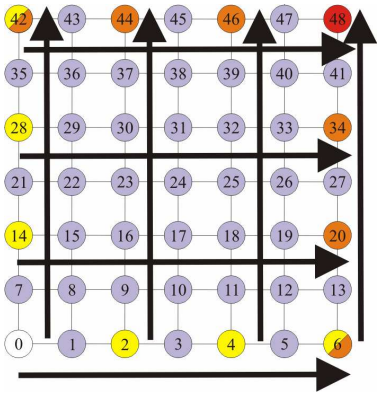


Figure 18:4x4 flows

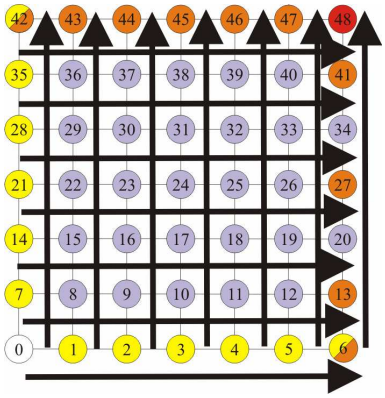


Figure 19:7x7 flows

5.1.2 Mobility Scenarios

Random waypoint

Random way point scenario was created using the “scengen” tool provided in the ns-2.29 package. The scengen tool creates n nodes and places them randomly. The nodes are moving during the simulation with random speeds, from 1m/s to 10m/s. In mobility scenarios there is a special parameter called pause time, which means that a node stops for a while after a movement. In this scenario, the pause time value was set to five seconds, used mainly for simulation purposes. The size of this scenario is set to 2000x2000 meters, therefore it is quite probable the existence of, at least, one route from a sender to a receiver node. The carrier sense threshold is 550m and the transmission range is 250m for each node. Simulation time for this scenario was set to 120 seconds.

Manhattan

In Manhattan scenario 200 nodes are used. Each node is defined with a carrier sense threshold of 130m and a transmission range of 60m. The main purpose of this scenario is simulating different speeds of the nodes, where there is a different scenario file for each maximum speed of the nodes. The scenarios used are taken from low mobility: 0m/s, 1m/s, 2m/s; to high mobility: 5 m/s, 10 m/s, 20 m/s.

The Manhattan scenario consists of a square of 500m x 500m where six streets are defined, 3 vertical and 3 horizontal, placed at equal distance. The nodes move along 12 lanes, 2 lanes for each street. This scenario was taken from the simulations of Nahm et al., and the main goal was to compare the obtained results in static and mobile scenarios for TCP and UDP in a different environment with controlled speeds. Simulation time for this scenario was set to 200 seconds.

5.2 Traffic Source

In order to perform our simulation, a scenario defining position and number of nodes, physical interfaces, routing protocol and a transport protocol (TCP/UDP) are needed. Besides this, it is also needed to attach some traffic to such transport protocol. In this section, the traffic sources are defined for the different transport protocols simulated, FTP traffic for TCP and CBR for UDP.

5.2.1 File Transfer Protocol (FTP)

FTP is a commonly used protocol for exchanging files over any network that supports the TCP/IP protocol. There are two parts involved in an FTP transfer (usually computers): a server and a client. The FTP server, running FTP server software, listens on the network for connection requests from other computers. The client, running FTP client software, initiates a connection to the server. Once connected, the client may perform a number of file manipulation operations such as uploading files to the server, download files from the server, rename or delete files on the server and so on [53]. In mobile ad hoc wireless networks, any node has the capability of being client and server. All nodes are potentially client and server at the same time.

The ftp packet size for all simulations was 1024 bytes for TCP. Unfortunately, the only way to configure parameters of ftp traffic is to adapt a free implementation of FTP to a new Application/FTP source and sink or use real FTP traffic. Therefore, we used the default FTP parameters (i.e. frequency of sent packets), due to the scope and goal of this dissertation.

5.2.2 Constant Bit Rate (CBR)

CBR is useful for streaming multimedia content such as audio. Therefore, we evaluated performance of CBRtraffic [54].

The packet size for all simulations was 512 bytes. The rate can be set in two ways, defining packets per second and the rate in KB. For all of our simulations we used five packets per second. However, we also simulated a 300kb rate in order to probe the mobile ad hoc wireless network for capacity constraints.

5.3 Performance parameters and graphical representation

This chapter aims to elucidate the following results in this master's thesis project. These graphics are composed of DSR and DSR-AR results, that is, results using normal DSR and results using our proposed approach, that we called DSR-AR.

In chain scenario, the horizontal axis shows the number of hops between the sender and receiver nodes. For Grid and Random Waypoint scenarios, it means the number of TCP or UDP flows. Finally, in Manhattan scenario the horizontal axis indicates the maximum speed of nodes in meters per second.

5.3.1 Throughput

These graphics present the overall rate of transfer for normal DSR (red line) and for DSR-AR (green line). The vertical axis shows the throughput, measured in kilobytes per second, which should be maximized.

$$Throughput = \frac{received\_bytes}{Time\_of\_simulation}$$

5.3.2 Routing overhead

These graphics show the number of the routing packets per second for DSR (red line) and for DSR-AR (green line), which should be minimized. In the vertical axis appears the number of packets per second, whereas in the horizontal axis the number of hops or flows.

$$Routing\_overhead = \frac{number\_routing\_bytes}{number\_of\_(routing + data)bytes}$$

5.3.3 Lost packets

These graphics show for normal DSR (red line) and DSR-AR (green line) the percentage of packets that were sent by the sender nodes, but were not received by receiver nodes along all simulation time. This value should be minimized.

$$Lost\_packets = (sentPackets - receivedPackets)$$

5.3.4 MAC errors

These graphics show in the vertical axis the dropped packets from the MAC layer for DSR (red column) and DSR-AR (green column). There were found 4 different type of MAC errors: collision (these errors usually represent up to 99% of the total errors), retry exceed count, MAC busy or duplicate packet. These errors should be minimized.

5.3.5 Route errors

Each time DSR or DSR-AR performs a route error process at sender, it is registered and showed in these graphics. A route error in DSR triggers a route maintenance process provoking more control traffic in the network. Usually these kinds of errors are due to broken links because of the mobility of nodes, but they may arise from collision of packets, as well. These errors should be minimized.

5.3.6 Route changes

If the sender and the receiver nodes are not near enough to communicate one to each other, they may use intermediate nodes to make a successful transmission. Due to the mobility of nodes, this route may change several times.

A route change is a new route used to reach the receiver node. If one of the links between sender and receiver nodes is broken and is feasible a new route, then an optional route is used, registering the route change and showed in these graphics. These changes should be minimized.

5.3.7 Route changes between two nodes along simulation time.

These graphics explain how the length of the route between a pre-selected sender and receiver nodes changes along the simulation time. For example, the optimal length in grid scenario for nodes 21 to 27 is 6 hops along all simulation time. Because of packet collisions, this route changes many times even if no mobility and therefore decreasing the overall throughput.

5.4 Chain scenario

We obtained improvements in throughput and routing overhead for two, four and eight flows from node 0 to 4, 5…20. Since chain is a static scenario, broken links should not appear like in mobility scenarios. Therefore, the routing overhead is also a consequence of the MAC layer, which informs to DSR about link errors due to collisions of packets instead of the mobility. Once the routing overhead is reduced, the throughput increases.

5.4.1 TCP

Using both DSR – AR and β2 [28] described in section 4.5, all route errors were removed. The more flows we use, the more MAC errors we obtain, because all flows travel though a unique path.

MAC errors for eight TCP flows between DSR – AR is almost the double than using normal DSR, although these results are because of the notorious reduction of routing packets and there is only one path for all flows. DSR–AR reduces routing packets, by identifying the reason of link errors at MAC layer and avoiding the route error process. Therefore, the routes that can still be used are removed from the caches at the intermediate nodes, leading to a better performance for data packets. Then, MAC layer must manage additional data packets, achieving a higher overall throughput, although in this scenario more packets are dropped from the MAC layer.

An important usage of DSR-AR is the reduction of route errors compared with normal DSR. This is because using normal DSR route errors were mistakenly detected. In normal DSR, all link errors from MAC layer are treated as broken links, leading to route errors. Sometimes this is wrong, because the congestion may cause link errors as well. If a link error was due to congestion, DSR increases that congestion with a route error process. It should be noticed that in a static scenario, such as “chain”, the route errors should be minimized.

A significant improvement of DSR-AR is related to changes of the length of routes (see changes of the length of routes). In this case, those changes become lower and stable. This is directly derived from the reduction of route errors. Once a route error is detected, routing protocol DSR triggers “route maintenance”, increasing the possibility of finding a new route towards the destination node, therefore, the length of the route may change.

The following graphics are related to the simulation of eight flows during 120 seconds, results for other number of flows can be found in the appendix.

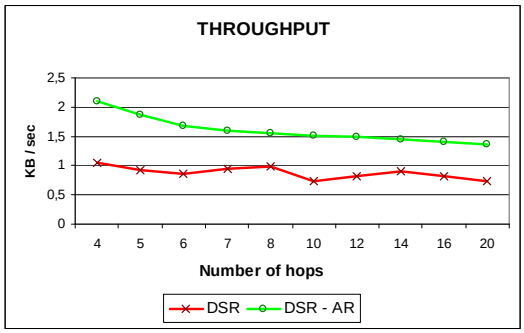


Figure 20:Throughput

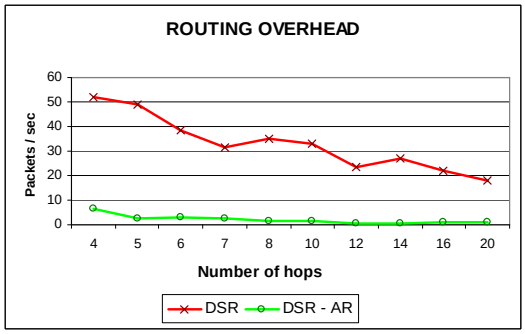


Figure 21:Routing overhead

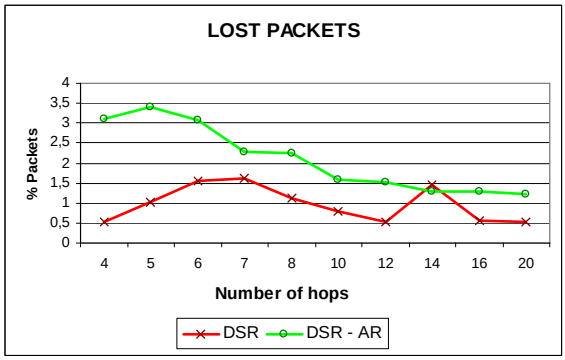


Figure 22: Lost packets

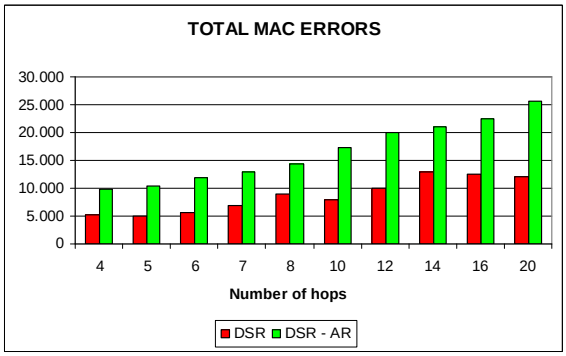


Figure 23: MAC errors

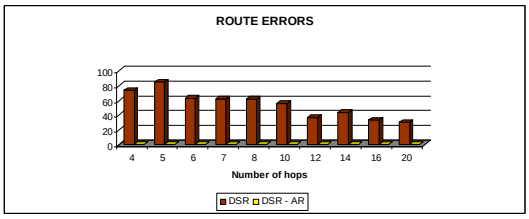


Figure 24: Route errors

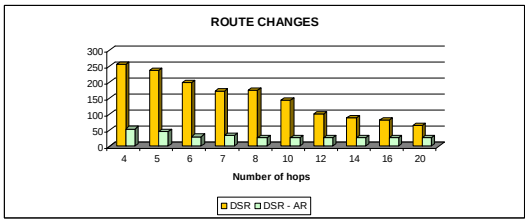


Figure 25: Route changes

Routing overhead, percentage of lost packets and number of route changes decrease in normal DSR when the number of hops grows. All route errors were removed using DSR-AR, which means that they exist due to congestion. Since there is no alternative route, all packets must use the same route, which leads to higher congestion when the number of hops is smaller. However, the MAC errors increases in DSR-AR. Collisions are due to a higher traffic of data because there are no route error processes, which avoid transmission of data. Finally, it is noticed that the throughput decreases when path length is increasing in chain topology. This behaviour was explained in the section 2.4.2 (effects of path length in TCP).

5.4.2 UDP

This simulation was based on CBR (five packets per second), whereas TCP simulation was based on FTP. Hence, the obtained values cannot be compared directly. However, in the following graphics the strong relationship among all of them is shown. While simulating normal DSR and UDP, some troubles were found in the core of the NS-2 related to the memory management in Linux. However, these results were included, for 14 the number of hops for instance, and commented, because after several simulation sessions the errors remained. The same simulation session was performed successfully using DSR-AR.

Ns-2 simulated for 14 hops shows a high number of “retry exceed count”, which means that in the MAC layer some packets were not delivered after all possible retransmissions. That explains the higher number of packets lost (figure 28). Because of this, DSR detects a route error and triggers the route maintenance process, which provokes more routing overhead (figure 27), decreasing the overall throughput of the network.

The results are according to the amount of traffic simulated, that is, there was not much data simulated, therefore we obtained similar results using normal DSR and DSR-AR. The following graphics are related to the simulation of 8 UDP flows during 120 seconds where the number of hops to reach the destination varies, results for other number of flows can be found in the appendix.

Even in a low traffic scenario, some route errors were found simulating for normal DSR. Using DSR-AR, these route errors were removed and as a consequence of that, the routing overhead was reduced. This is because normal DSR misinterprets the information received from the MAC layer and triggers a route error when a neighbouring node is not reachable. This route error tries to find a route to reach the destination that could lead to increase the routing overhead. Since chain is a static scenario, route errors should not be triggered. Therefore, DSR was misinterpreting the information from MAC layer, which leaded to an increment of the routing overhead.

Because of the low traffic generated, improvements in the throughput are not significant, but it is observed that the number of route errors and routing overhead was reduced. It is observed that the number of lost packets for such a low traffic scenario is low. Some problems were experienced simulating normal DSR for 14 and 20 number of hops; therefore, we did not take into account these results because they are not 100% reliable.

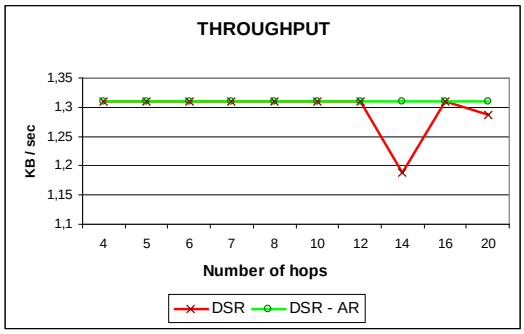


Figure 26:Throughput

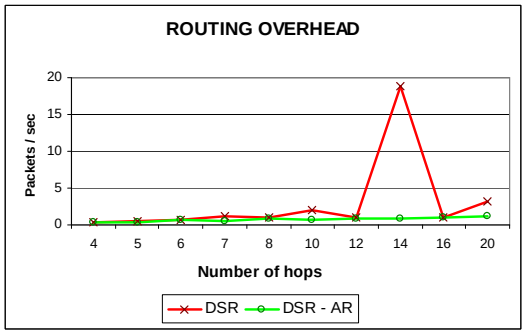


Figure 27:Routing overhead

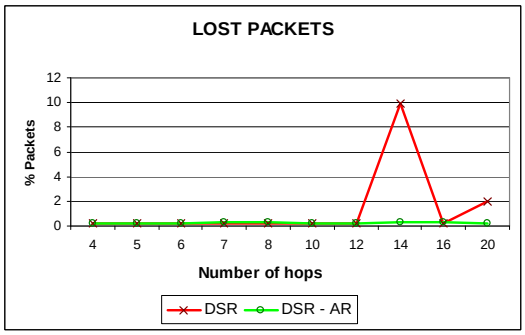


Figure 28:Lost packets

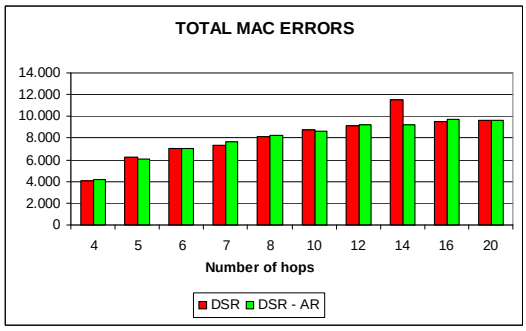


Figure 29: Total MAC errors

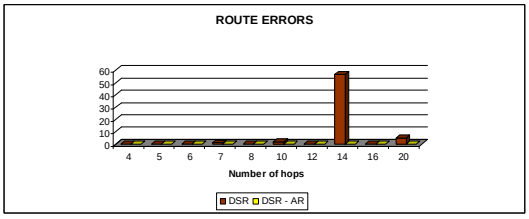


Figure 30:Route errors

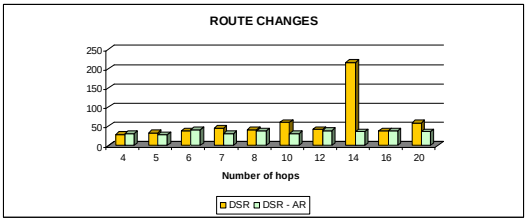


Figure 31:Route changes

5.5 Grid scenario

Several improvements were achieved in throughput and routing overhead for 2 (two parallel flows from node 21 to node 27), 2 (two cross flows, from node 21 to node 27 and from node 3 to node 46), 4, 6, 8, 14 flows, see also figures 14 to 19. Since grid is a static scenario, there should not be broken links like in mobility scenarios. The simulation time for grid scenario was 120 seconds for both TCP and UDP. The throughput is obtained as an aggregate and not per flow. Since the grid topology has much more redundancy than chain topology, it allows alternative routes and back-up resources; therefore, all the graphics shown below perform higher values than the results obtained in chain topology.

5.5.1 TCP

Using both DSR–AR and β2 [28] described in section 4.5, all route errors were removed. The number of MAC errors in DSR – AR is decreased compared to normal DSR, because of the high reduction of routing packets (up to 94% less). Since DSR–AR reduces routing packets, MAC layer must manage additional data packets, achieving higher overall throughput (up to 45% more). Furthermore, in this scenario, there is a significant reduction of dropped packets from MAC layer, because there are alternative routes between sender and receiver nodes.

An important usage of DSR-AR is the reduction of route errors compared with normal DSR, since route errors were correctly detected using DSR-AR, like in chain scenario.

In DSR, the length of the route fluctuates between high and optimal values, due to a big amount of route errors and the consequent new routes found. By reducing route errors in DSR-AR, the length of the routes becomes optimal and constant. DSR-AR improves the results obtained using DAMPEN Policy [28] as well.

The lost packets for normal DSR and DSR-AR are presented in figure 34. They are mainly due to congestion and mobility of nodes. Lost packets in DSR-AR are higher than normal DSR because routes are not re-established, during this time more congestion was created, because more data was transmitted. In normal DSR, the data could not be transmitted because the route errors were mistakenly detected.

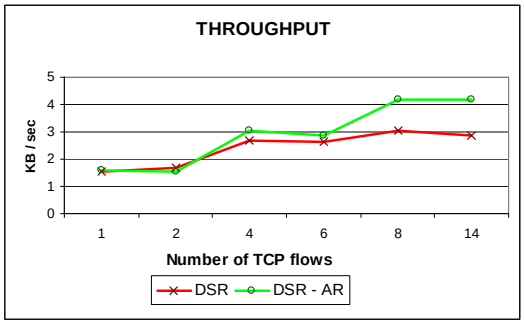


Figure 32:Throughput

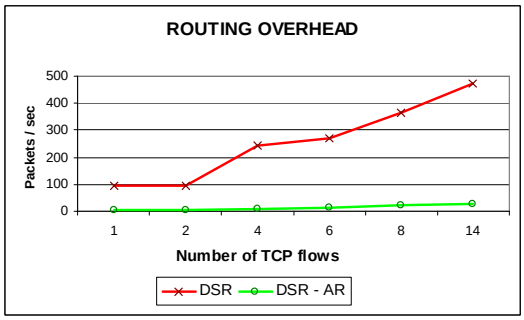


Figure 33:Routing overhead

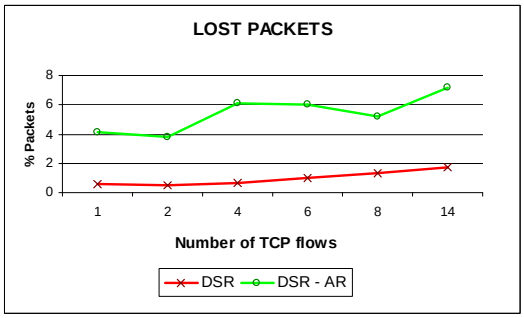


Figure 34:Lost packets

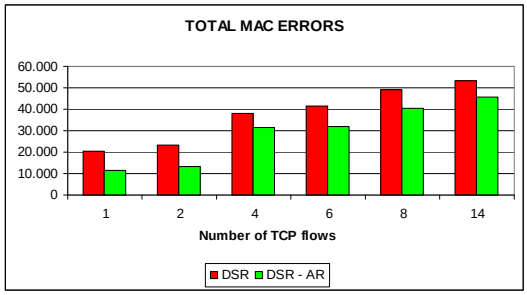


Figure 35:Total MAC errors

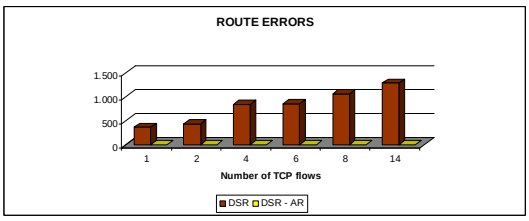


Figure 36: Route errors

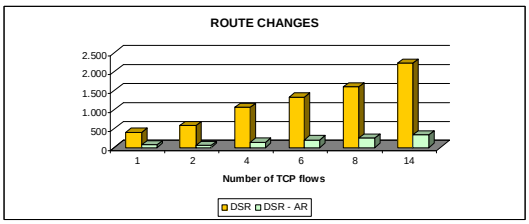


Figure 37:Route changes

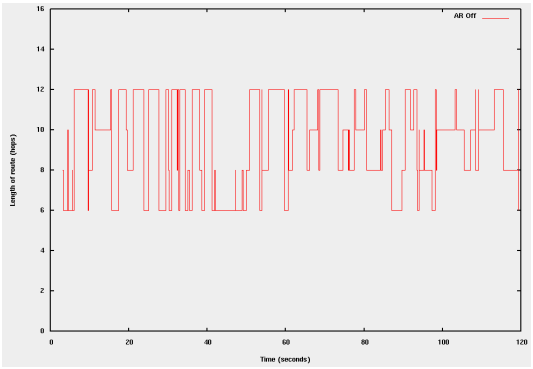


Figure 38: Length of the routes (number of hops), from node 21 to node 27.Normal DSR: Two parallel TCP flows

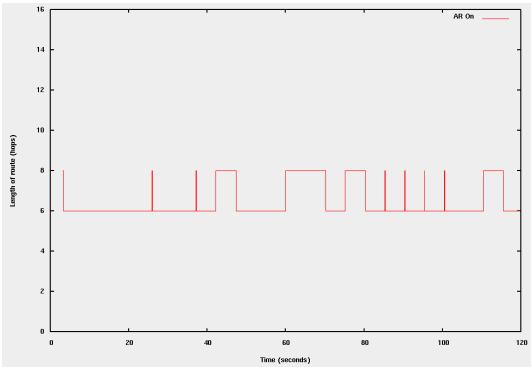


Figure 39: Length of the routes (number of hops), from node 21 to node 27. DSR – AR: Two parallel TCP flows

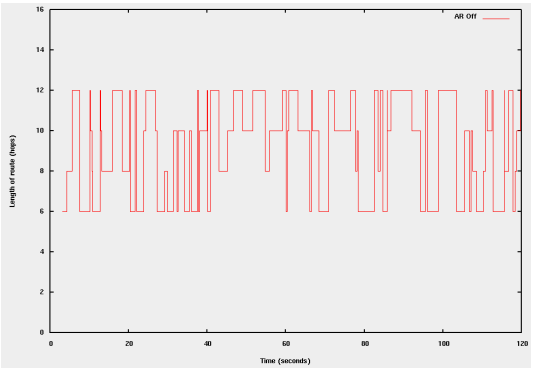


Figure 40: Length of the routes (number of hops), from node 21 to node 27. Normal DSR: Two cross TCP flows

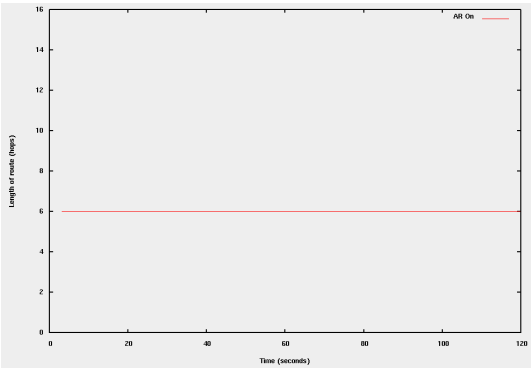


Figure 41:Length of the routes (number of hops), from node 21 to node 27. DSR – AR: two cross TCP flows

DSR triggers a route error process when the communication among nodes to reach the destination fails. This process looks for an alternative route and the length of the route changes. This process increases routing overhead that might decrease throughput.

These graphics show the length of the route (in number of hops) from node 21 to node 27 during the simulation. The number of hops to reach the destination was extracted from the trace file and analyzed for two parallel and two cross flows, figures 14 and 15 respectively, with the purpose of showing the differences between normal DSR and DSR-AR.

5.5.2 UDP

On one hand, we may notice a reduction of the routing overhead in figure 43 for all flows using DSR-AR. This is because normal DSR triggers a route error process when communication among nodes to reach the destination fails, increasing the routing overhead as was explained in the section above. The improvement is not so good as using TCP, since the amount of data injected in the network is higher using TCP than UDP.

On the other hand, there are not important improvements for throughput or MAC errors when low UDP traffic is generated in grid and chain scenarios.

Besides Fig. 46, for each graphic it can be noticed that when more TCP flows are simulated, higher results are obtained. This is rather usual, since the number of flows directly affects to the amount of data and therefore, to the results obtained.

Finally, it is observable that some results at flow 14 are quite high. As we explained in the section above, this behaviour may be a consequence of a result magnification made by ns-2 in some Linux environments.

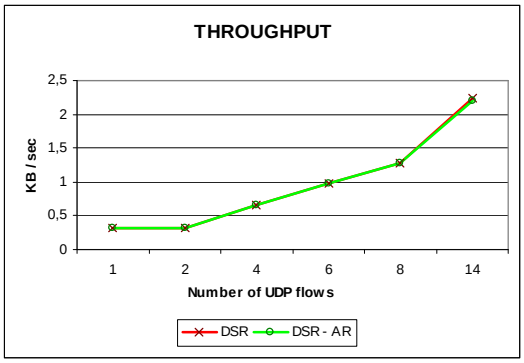


Figure 42:Throughput

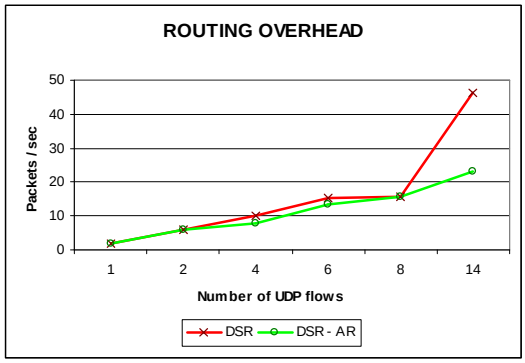


Figure 43:Routing overhead

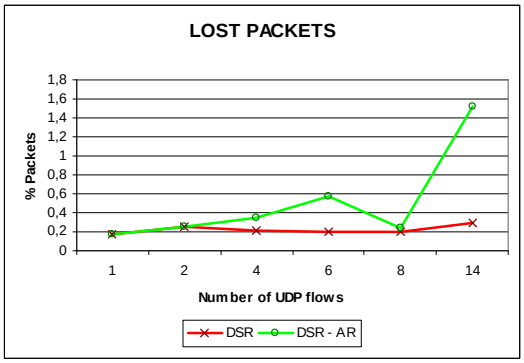


Figure 44: Lost packets

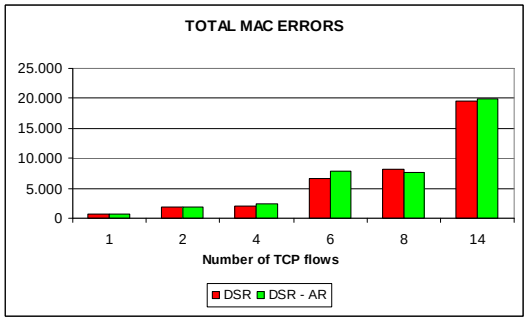


Figure 45:Total MAC errors

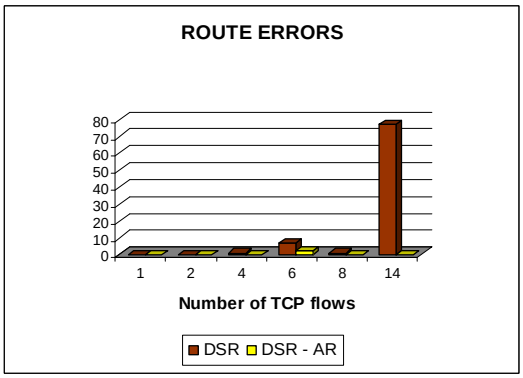


Figure 46:Route errors

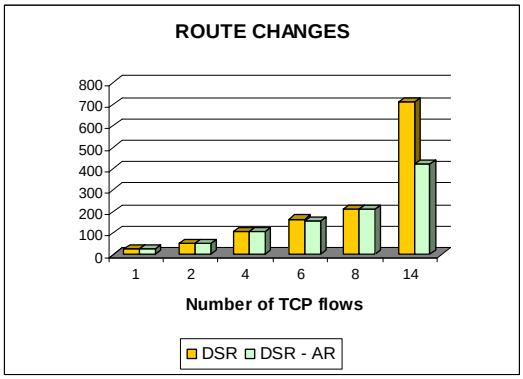


Figure 47:Route changes

5.6 Random Waypoint scenario

5.6.1 TCP

In TCP random waypoint scenario, MAC errors are increasing using normal DSR (figure 51), because of the mobility of nodes (from a minimum of 0m/s to a maximum of 10m/s). This mobility, as well, causes a high number of route errors and therefore more routing overhead and packet loss. Moreover, the throughput decreases meanwhile the number of flows increases.

Figure 48 shows better results for throughput using DSR-AR for all flows (except for 8 flows). Like in grid scenario, the throughput increases upon skipping unnecessary routing maintenance process at routing layer. Related to lost packets, figure 50 shows that more lost packets are achieved using DSR-AR. If we compare these results to figure 48 we realize that the more throughput is achieved, the more lost packets are obtained. For 8 flows, the throughput of normal DSR is better, but in this case, lost packets is lower.

Figure 49 shows how routing overhead is severely reduced, since there is less routing activity at routing layer using DSR-AR. We see from Figure 51 that MAC errors are decreased for all flows (up to 66% errors less for 50 flows). We can see that the number of MAC errors is higher than those errors in static scenarios. Usually, in mobility scenarios there are more MAC errors. This is because the link failure with congestion over multiple hops is more usual, due to node mobility.

Figure 52 and figure 53 show route errors and route changes respectively. Although a route error not always provokes a route change, it is the regular behaviour. Using DSR-AR, the more number of flows are used, the more improvements are obtained compared to normal DSR.

As conclusion, it is worthwhile to point out that the graphics are related between themselves, since less MAC errors, less route errors, and less route changes provokes lower routing overhead in the network. As the routing overhead is decreasing, the nodes are able to transmit more data packets; therefore, a higher throughput is obtained (up to 300% for 50 flows). Furthermore, we can see how the improvement of DSR-AR decreases when the flow number is 8. This can be explained as follows. Less aggressive TCP traffic leads to less routing activity. Therefore, the improvement of DSR-AR avoiding routing activity is not so good as simulating 16 or 32 TCP flows.

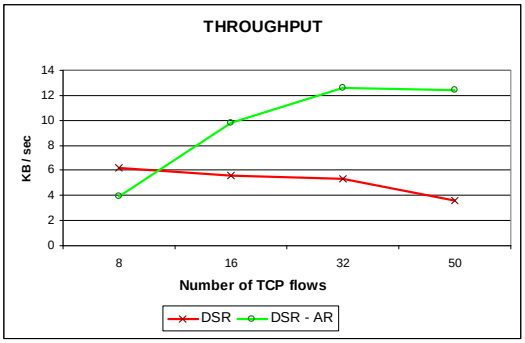


Figure 48:Throughput

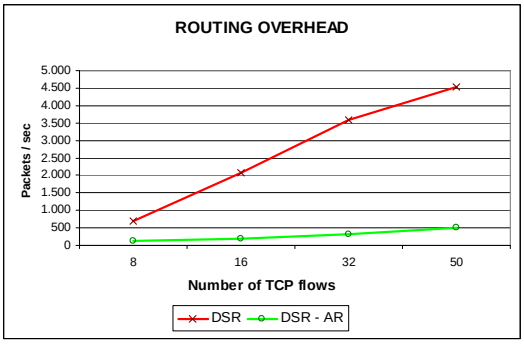


Figure 49:Routing overhead

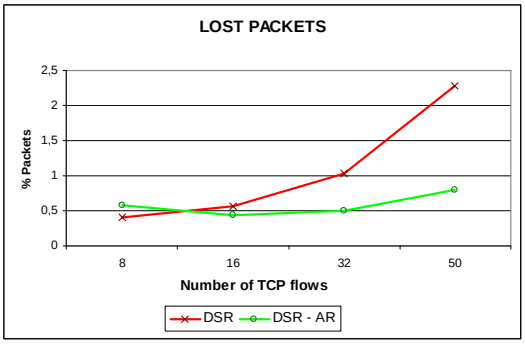


Figure 50:Lost packets

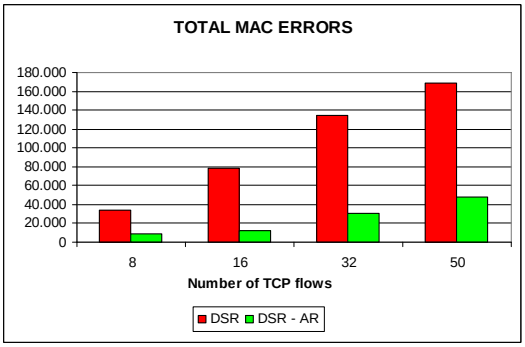


Figure 51:Total MAC errors

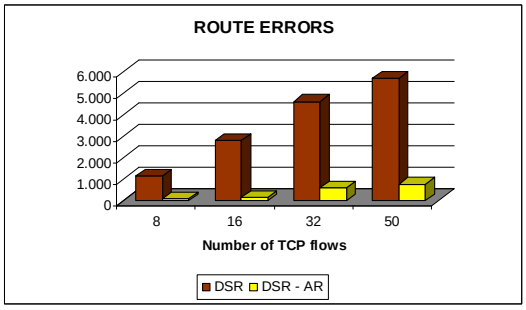


Figure 52:Route errors

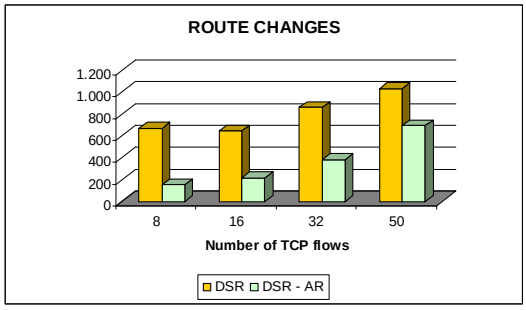


Figure 53:Route changes

5.6.2 UDP

Figure 54 shows an improvement for all flows related to the throughput. The main improvement is achieved for 16 flows up to 128% of throughput performance. However, for 32 flows the improvement is not so high.

If we compare these graphics with those got in chain and grid scenarios, we may notice that better results are achieved. For instance, throughput using DSR-AR for 8 UDP flows reaches about 2 KB/s (figure 54), while for 8 UDP flows in grid scenario DSR-AR reaches 1.25 KB/s approximately. In this case, DSR-AR works better with node mobility, where more link failures occur; therefore, DSR-AR is able to identify higher number of correct links that should not be removed, achieving better results.

On the other hand, mobility of nodes and the large amount of traffic (300KB/s for each flow) also leads to higher percentage of packets loss (figure 56). However, these lost packets can be reduced using DSR-AR up to 4.5% for 8 flows. Again, this graphic shows better results using DSR-AR for mobile scenario compared to a static scenario (figure 44).

In any case, the rest of the flows perform well. A 55% less of MAC errors (figure 57) were obtained using DSR-AR. Additionally, using DSR-AR, there was achieved between 47% and 58% less number of route changes (figure 59) than normal DSR and between 49% and 62% less route errors (figure 58).

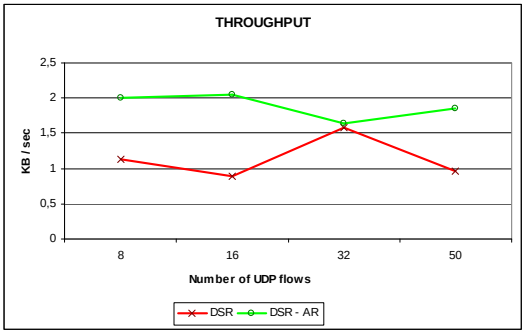


Figure 54:Throughput

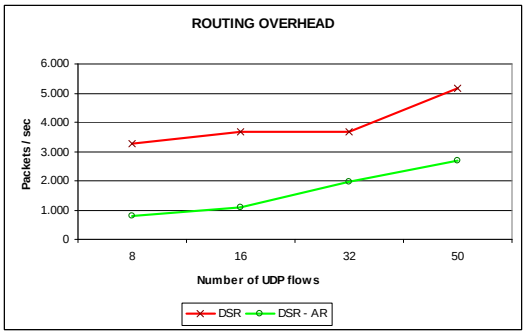


Figure 55: Routing overhead

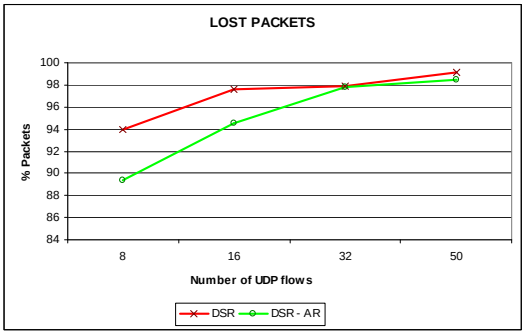


Figure 56:Lost packets

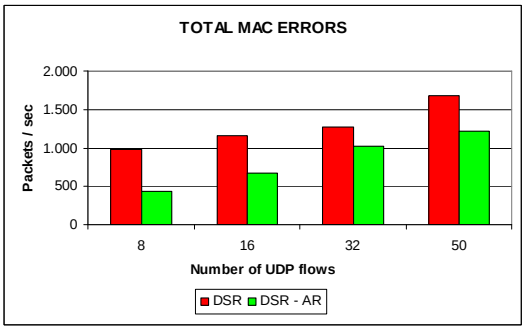


Figure 57: Total MAC errors

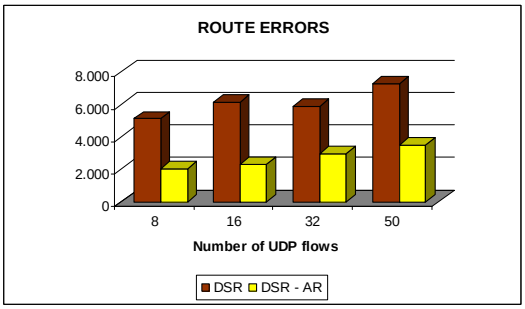


Figure 58: Total errors

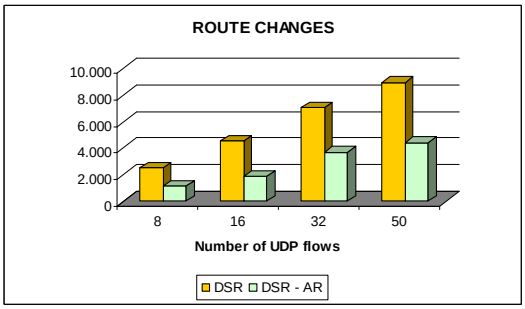


Figure 59: Route changes

5.7 Manhattan scenario

5.7.1 TCP

In this scenario, 20 flows were simulated. Figure 60 shows the throughput in Manhattan. We can notice that with high mobility (10, 20 m/s) the throughput drastically decreases for both DSR-AR and normal DSR.

Figure 61 shows how routing overhead increases for both DSR-AR and normal DSR when the mobility is higher (5, 10, 20 m/s). This behaviour is a bit different compared to random mobile scenario, where the routing overhead also increases at higher number of TCP flows (figure 49). On the other hand, the throughput was increased for DSR-AR. It is meaningful to point out that higher speed simulations negatively affects DSR-AR throughput.

The relationship among the graphics can be analyzed from more MAC errors (figure 63). As long as nodes speed increases, MAC collisions happen frequently, thus, it causes higher number of route errors, therefore more routing overhead and more lost packets (figures 61, 62). However, the number of route changes (figure 65) decreases at high speed of the nodes, since they are not able to create new routes quickly. Because of this, the throughput is negatively affected and decreases.

Even when high speed affects DSR-AR, we see that using DSR-AR better results were achieved compared to normal DSR. Since MAC errors were properly identified (up to 83% less errors for 5 m/s), there are less route errors (figures 63, 64). Furthermore, the number of route changes is also decreased (figure 65), unless the speed goes up to 10 m/s, where DSR-AR and normal DSR perform similar. Hence, lower routing overhead in the network. Moreover, the nodes are able to transmit more data packets and a higher throughput is obtained (figure 60) in all scenarios (up to 107% for 5m/s). However, the improvement with 10, 20m/s is not so good, since route errors are due to mobility and not to congestion.

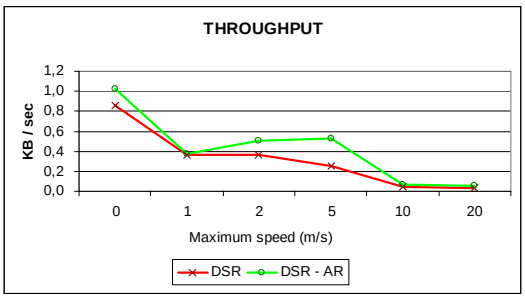


Figure 60:Throughput

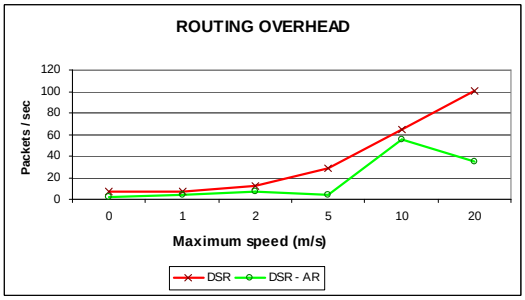


Figure 61:Routing overhead

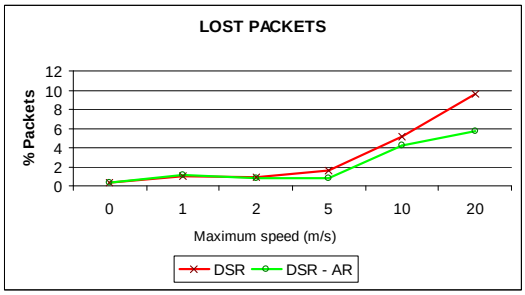


Figure 62:Packet loss

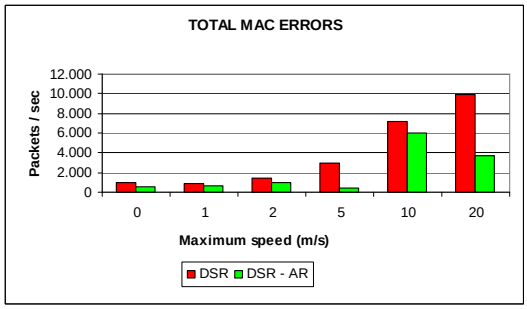


Figure 63:Total MAC errors

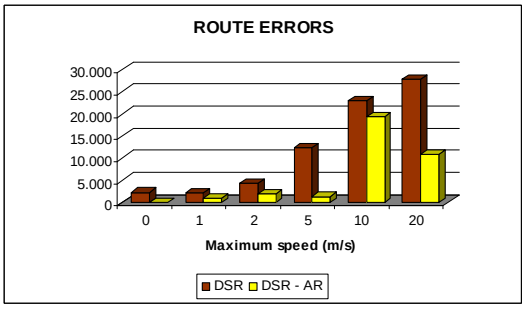


Figure 64: Route errors

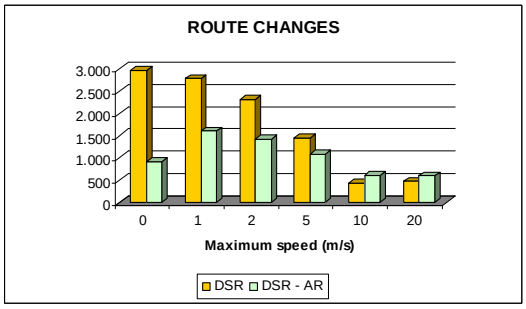


Figure 65: Route changes

5.7.2 UDP

In this scenario, 20 flows were simulated. Figure 66 shows similar throughput results for both DSR-AR and normal DSR. In this case, DSR-AR in Manhattan UDP cannot improve as well as Random mobile scenario (Figure 54). This is a consequence of the big amount of traffic generated and the high number of lost packets. Figure 67 shows improvements in all speeds using DSR-AR compared to normal DSR. We can see as with high speed, more node mobility, and more routing overhead. If we look at figure 69, we can see as MAC errors increase for both DSR-AR and normal DSR when speed is higher. In any case, DSR-AR shows better results for MAC errors. MAC errors are higher because at high node speed, MAC collisions happen more often. MAC collisions lead to higher number of route errors, increasing the routing overhead as well.

Using DSR-AR in Manhattan UDP, similar results were achieved compared to normal DSR. Since MAC errors (figure 69) were properly identified (up to 54% errors less for 5 m/s), there are less route errors and thus, lower routing overhead in the network. Because of the mobility of nodes, there are many route changes even with DSR-AR, which makes DSR-AR more suitable for static scenarios but not decreasing performance for mobility scenarios.

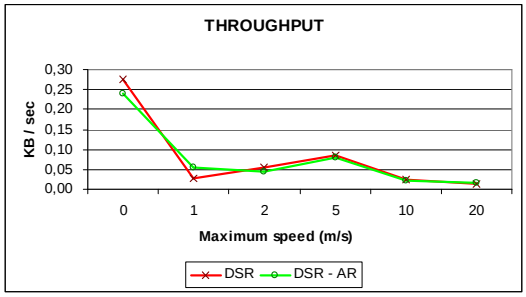


Figure 66:Throughput

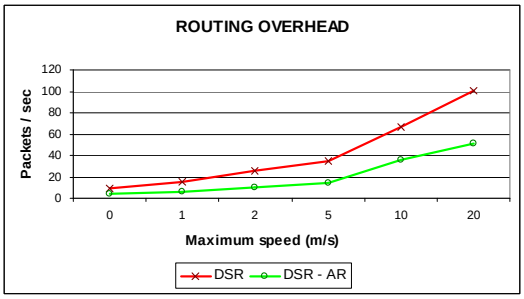


Figure 67:Routing overhead

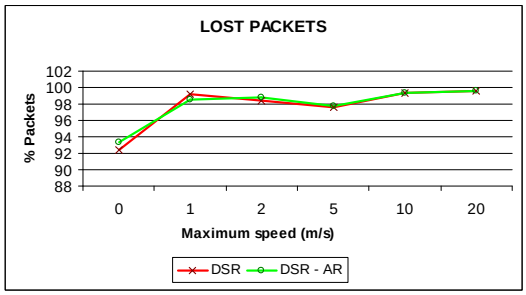


Figure 68: Lost packets

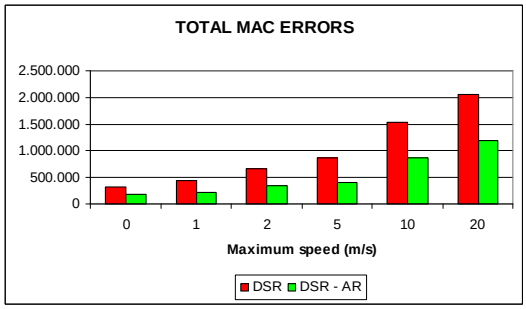


Figure 69: Total MAC errors

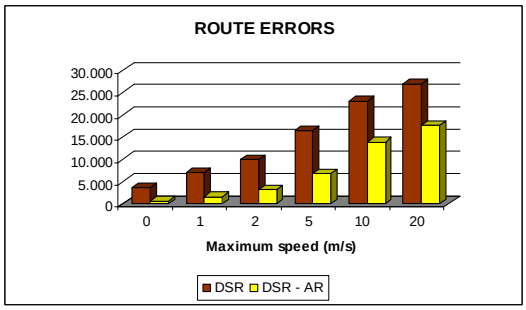


Figure 70: Route errors

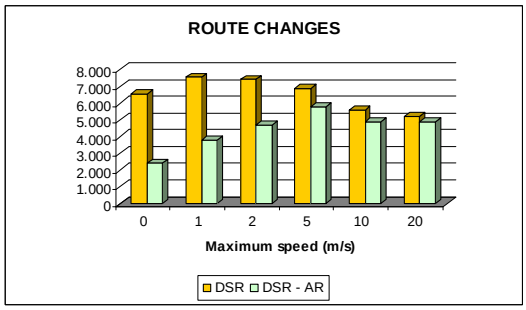


Figure 71: Route changes

6 Conclusions and Future Work

The purpose of this chapter is to summarize the outcome of the master's thesis. The results have been obtained using the proposed goals as reference. Afterward, an additional improvement in DSR protocol is proposed as future work.

6.1 Conclusions

The key issue treated in this master's thesis project has been the improvement of the throughput and reduction of routing overhead in mobile ad hoc networks using the ns-2 network simulator. We performed several simulations in both static and mobile scenarios with two transport protocols, TCP and UDP, in order to check the adaptability of proposed approach for different contexts. We focused our work on 2 layers; these are MAC 802.11 protocol in the data link layer and DSR protocol in the routing layer.

In the situation of congestion, the MAC 802.11 protocol does not perform well enough using the RTS/CTS/DATA/ACK dialog because RTS packets from a sender to a receiver may collide and communication fails, leading to bad interaction with the routing layer. The MAC layer could mistakenly inform the routing layer about broken links when the communication among nodes is still possible, when, however, communication failed due to the collisions of RTS packets caused by congestion. Thus, the routing protocol interprets broken links in the MAC layer as route errors, triggering the route maintenance process, therefore increasing the overhead in the network. Our main purpose in this thesis is to avoid this misinterpretation by determining the cause of the broken links. To deal with this, the signal strength of each node is tracked. This information is used to notify the routing protocol if the node is still reachable but the communication was dropped. Then, at routing layer it is possible to distinguish if the route exists and do not trigger a route error process, avoiding routing packets which would increase more the congestion in the network.

For every received packet in the MAC layer, the signal strength of the node is stored (the last 20 receptions). After attempting to send a RTS packet to a neighbouring node seven times without receiving CTS [48], a node concludes that communication is not possible. The last signal strength of the receiver node is compared to the transmission threshold. If the signal strength determines that the node is near enough, the MAC layer informs the routing protocol that it is not necessary to trigger a route maintenance process. At this moment, the routing

protocol knows that the error was due to collisions in the MAC layer and the node is within the transmission threshold. The routing protocol does not interpret this link error at MAC layer as a broken link due to mobility and does not trigger a route error process because the route still exists.

In static scenarios, such as chain and grid, superfluous routing overhead is reduced, and therefore, the throughput of the network increases. In chain scenario, the length of the route varies between high and normal values, due to a fair number of route errors and the consequent new routes found. By reducing route errors in DSR-AR, the length of the routes becomes optimal.

In mobility scenarios, we may distinguish among different node speeds. For low and medium speeds of nodes (up to 10 m/s), the DSR-AR approach achieves optimal results, obtaining up to 80 % less routing overhead in random way point and 80% in Manhattan. Additionally, the throughput is improved by up to 90 % in Manhattan.

The performance for high mobility scenarios is not significant, since from 20m/s network performance does not improve. However, using DSR-AR the network performance does not decrease, either. Interestingly, in a crowded network as Manhattan, DSR-AR performance is not as good as Random mobile scenario.

Ideally, DSR-AR should be more effective in static scenarios, where misinterpretation of broken links is more common than in mobility scenarios. This can be deduced independently from the transport protocol used or the traffic generated.

Finally, it should be noticed that a real system network might use more network resources than the simulations. Thus, the results tend to be lower for both DSR-AR and normal DSR in a real environment.

6.2 Future work

Besides the DSR-AR scheme proposed, we have developed a new feature with the objective of avoiding disconnections in scenarios with objects that interfere with the communication among nodes. The main idea makes sense in mobility scenarios, for example an office with mobile nodes. Here the connection can be interrupted easily due to furniture obstructing the communication.

If the receiver node is not moving away from the transmitter node, it is possible to avoid disconnection by making an average of the received signals. This new approach makes a

prediction based on the last twenty movements and interprets if the receiver node is not reachable anymore. We should be careful choosing what method of average (arithmetic mean, geometric mean, harmonic mean, etc.) offers a better behaviour according to the different speeds or scenarios. Moreover, it is important to decide on the value of the average which determines if the protocol continues trying to retransmit or not, and on the number of times that this node was moving away.

References

[1] Abolhasan, M. & Wysocki, T. & Dutkiewicz, E. (2003). *A review of routing protocols for mobile ad hoc networks*

[2] Akintola, A.& Aderounmu, A. & Owojori, A. & Adigun,M.O.(2006) *Performance modelling of UDP over IP-Based wireline and Wireless Networks*

[3] Bharghavan, V. *MACAW, a media access protocol for wireless LAN's*

[4] Chandran, K. & Raghunathan, S. & Venkatesan, S. & prakash, R. (2001). *A feedback-based scheme for improving TCP performance in ad hoc netwoks.*

[5] Chandramouli, R. & Rose, S. *Secure domain name system (DNS) deployment guide*

[6] Douglas, S. & Aguayo, D. & Benjamin, A.& Morris, R. (2003) Performance of multihop wireless networks; shortest path is not enough

[7] Dube, R. & Rasi, C & Wang, K. & Tripathi, K. (1997) *Signal stability-based adaptive routing for ad hoc mobile networks*

[8] Estrin, D & Govindan, R. & Heidenman, J. & Kumar, S. *Next century challenges: scalable coordination in sensor networks*

[9] Fahmy, S. & Prabhakar, V. & Avasarala, S. & Younis, O. *TCP over wireless links, mechanisms and implications*

[10] Fall, K. & Floyd, S. *Simulation-based comparisons of Tahoe, Reno, and SACK TCP*

[11] Fall, K. & Varadhan, K. *The ns manual (formerly ns Notes and Documentation).* Retrieved April 20, 2006 from http://www.isi.edu/nsnam/ns/ns-documentation.html

[12] Garcia-Luna-Aceves, J.J. & Spohn, M. (1999). *Source-tree routing in wireless networks*

[13] Grilo, A. & Nunes, M. *Performance evaluation of IEEE 802.11e*

[14] Holland, G. & Vaidya, N. (1999). *Analysis of TCP performance over mobile ad hoc networks*

[15] Hu, Y. & Johnson, D. *Implicit source routes for on-demand ad hoc network routing*

[16] Ilyas, M. (2003). *The Handbook of Ad Hoc Wireless Networks*

[17] Jacobson, V. (1999). *Modified TCP Congestion Avoidance Algorithm.* Retrieved October 20, 2006 from ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt

[18] Johnson, D. & Maltz, D. & Broch, J.DSR: *The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks*

[19] Karn, P. *MACA, a new channel access method for packet radio*

[20] Kim, D. & Toh, K. & Choi, Y. (2001). TCP-Bus: improving TCP performance in wireless ad hoc networks

[21] Klark, N. Perl. Retrieved April, 20, 2006 from http://search.cpan.org/dist/perl/pod/perldsc.pod

[22] Ko, Y. & Vaidya, H. (1998) *Location-aided routing (LAR) in mobile ad hoc networks*

[23] Kopparty, S. & Krishnamurthy, & V. T Faloutsos, M. (2002). *Split TCP for mobile ad hoc networks*

[24] Kurkowsky, S. & Camp, T. & Colagrosso, M. *MANET Simulation Studies: The incredibles*

[25] Liu, J. & Sigh, S. (2001). *ATCP: TCP for mobile ad hoc networks*

[26] Macker,J. & Corson, S. (1999) *Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and evaluation considerations*. Retrieved October, 06, from http://www.ietf.org/rfc/rfc2501.txt

[27] Murty, S. & Garcia-Luna-Aceves, J.J.(1996). *An efficient routing protocol for Wireless networks*

[28] Nahm, K, & Helmy, A. & Kuo, J. *TCP over multihop 802.11 networks: Issues and performance enhancement*

[29] Nahm, K & Helmy, A. & Jay Kuo, C. *Improving Stability and Performance of Multihop 802.11 Networks*

[30] Park, V. & Corson, S. (1997) *Temporally ordered routing algorithm (TORA)*

[31] Perkins, C. & Bhagwat, P. (1994) *Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers.*

[32] Rom,R. & Sidi, M. *Multiple access protocols, performance and analysis*

[33] Shah, V. & Krishnamurthy, S. *Handling asymmetry in power heterogeneous ad hoc networks: a cross layer approach*

[34] Shroff, N. . *Cross-Layer Design for Multihop Wireless Networks*

[35] Siva Ram, C. & Murthy, B.S. Manoj (2004). *Ad Hoc Wireless Networks, Architectures and Protocols*

[36] Su, W. & Gerla, M. (1999) *IPv6 flow handoff in ad hoc wireless networks using mobile prediction*

[37] Tanenbaum, A. (2003). *Computer Networks 4th edition*

[38] Vaidya, N. (2004). Mobile Ad Hoc Networks, Routing, MAC and Transport Issues

[39] Velayos, H. (2005) *Autonomic wireless networking*

[40] Velayos, H. & Karlsson, G. *Techniques to reduce the IEEE 802.11b handoff time cells*

[41] Velayos, H. & Karlsson, G. & Aleo, V. *Load Balancing in Overlapping Wireless LAN*

[42] Wang,Y. (2004) *Medium Access Control in ad hoc wireless networks with omni-directional and directional antennas*

[43] Yuen, W.& Lee, H. & Andersen, T. *A simple and effective cross layer networking system for mobile ad hoc networks*

[44] Zhou, L. & Haas, Z. *Securing ad hoc networks*

[45] 802.11 MAC code in NS-2.28. Retrieved April 20, 2006 http://www-ece.rice.edu/~jpr/ns/docs/802_11.html

[46] AMS glossary. Retrieved October 06, 2006 from http://amsglossary.allenpress.com/glossary/browse

[47] Bryan's ns2 DSR faq. Retrieved May 12, 2006 from http://www.geocities.com/b_j_hogan/

[48] IEEE www.ieee.org

[49] The data link layer. Retrieved, October 10, 2006 from http://mia.ece.uic.edu/~papers/ece333/spring04/ast_ch3.ppt

[50] The dynamic source routing protocol. Retrieved April 25, 2006 from http://www.cs.cmu.edu/~dmaltz/dsr.html

[51] The network simulator. Retrieved May 05, 2006 from http://www.isi.edu/nsnam/ns/

[52] The NS by example. Retrieved May 01, 2006 from http://nile.wpi.edu/NS/

[53] File Transport Protocol. Retrieved May 01, 2006 from http://en.wikipedia.org/wiki/File_Transfer_Protocol

[54] Constant Bit Rate. Retrieved May 01, 2006 from http://en.wikipedia.org/wiki/Constant_bit_rate

[55] The Click DSR Router Project. Retreived October 10, 2006 from http://pecolab.colorado.edu/DSR.html

[56] The Microsoft Research Mesh Connectivity Layer. Retreived October 10, 2006 from http://research.microsoft.com/mesh/

[57] The Monarch Project implementation. Retreived October 10, 2006 from http://www.monarch.cs.rice.edu/dsr-impl.html

[58] Piconet II mobile router, implementing an ad hoc routing protocol. Retreived October 10, 2006 from http://piconet.sourceforge.net/thesis/main.html

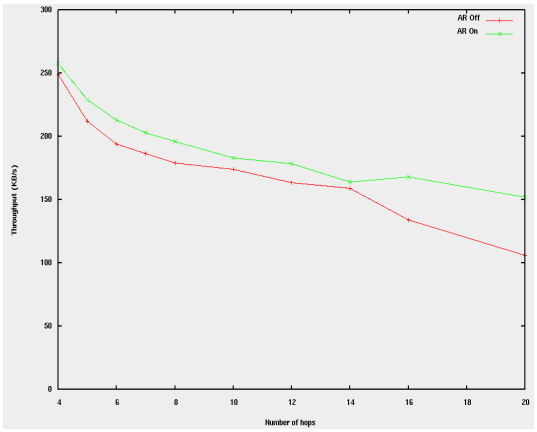
[59] OPNET, Retreived October 10, 2006 from http://w3.antd.nist.gov/wctg/prd_dsrfiles.html

Appendix A

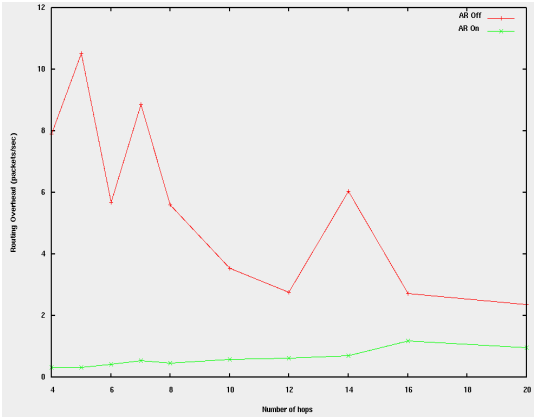
A.1 Chain scenario

A.1.1 TCP

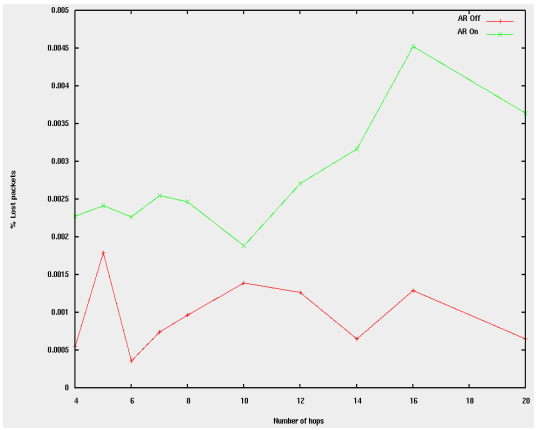
Two TCP flows



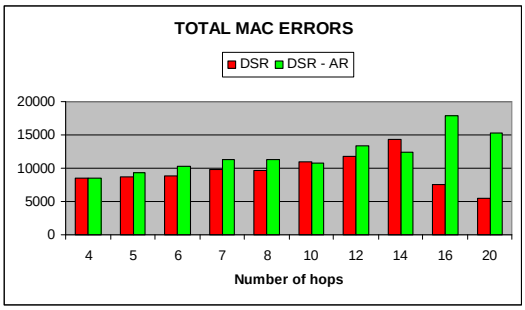
Throughput



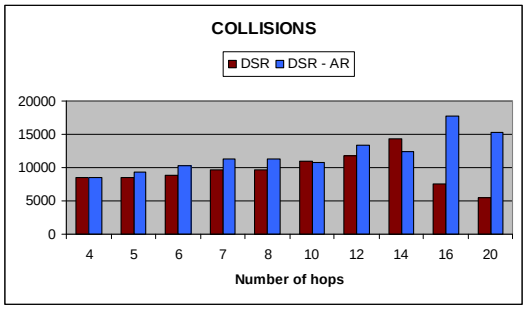
Routing overhead



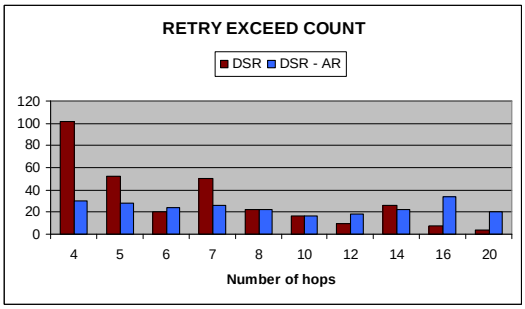
Lost packets (1 = 100%)



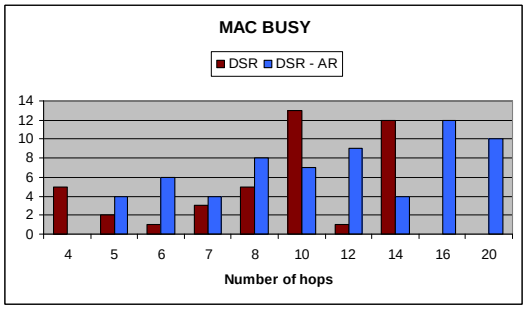
Dropped packets at MAC layer



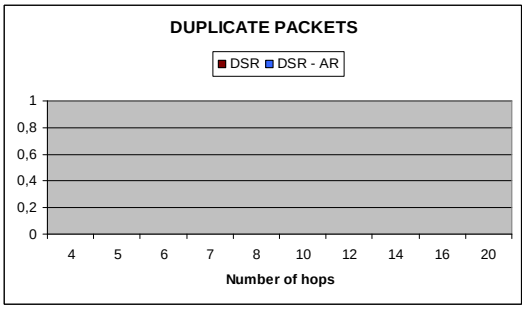
Number of collisions at MAC layer



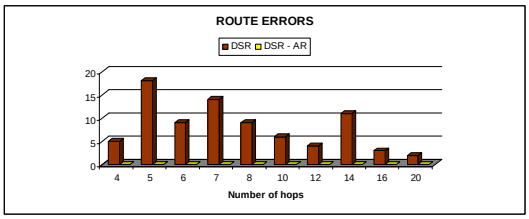
Number of dropped packets due to retry exceed count



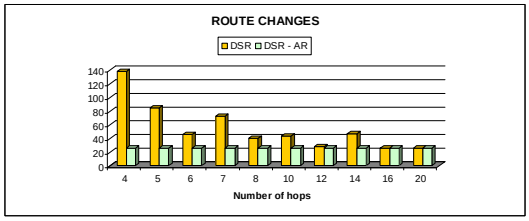
Packets dropped because of MAC busy



Duplicate packets at MAC

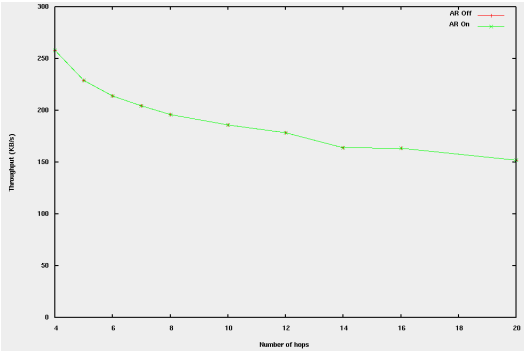


Number of route errors in DSR

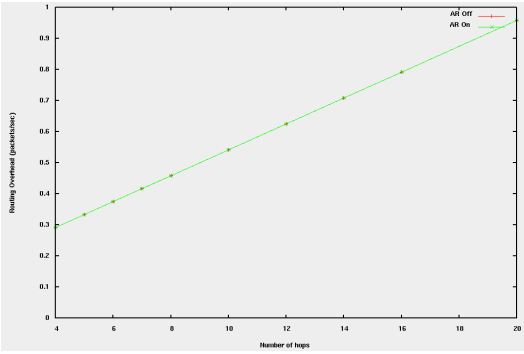


Aggregate of the number of changes in the length of the route

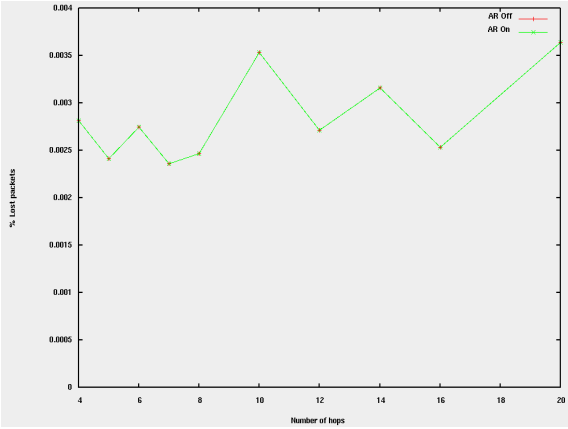
Dampem policy by Nahm β2 (AR off) and DSR-AR in collaboration with β2 (AR on)



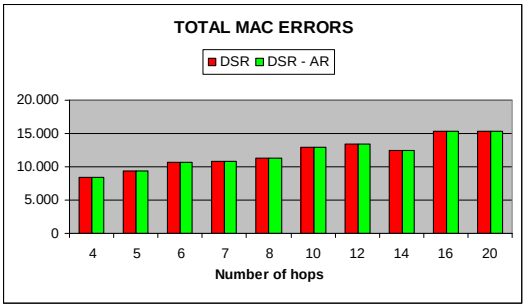
Throughput



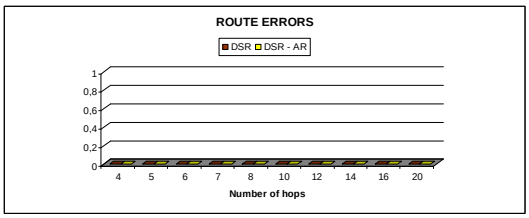
Routing overhead



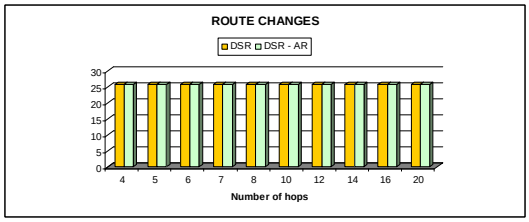
Lost packets



Total dropped packets at MAC

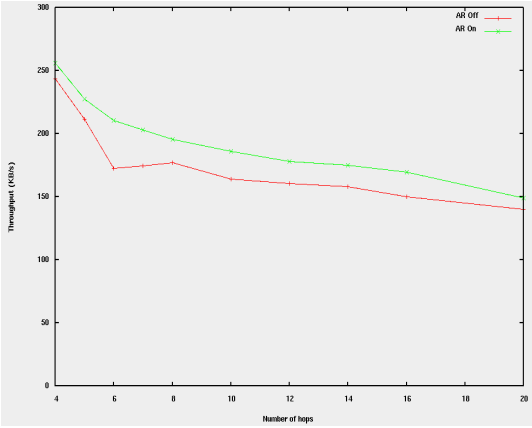


Number of route errors in DSR

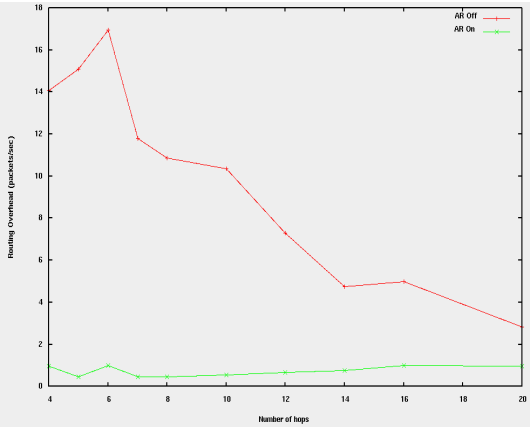


Aggregate of the number of changes in the length of the route

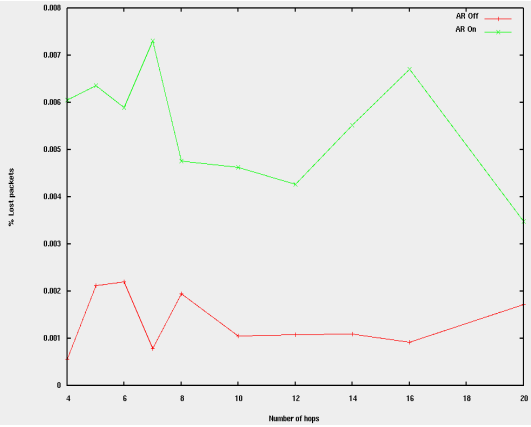
Four TCP flows



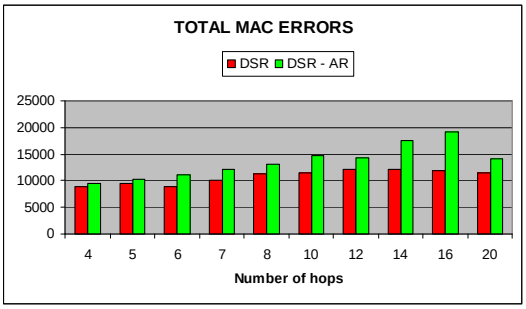
Throughput



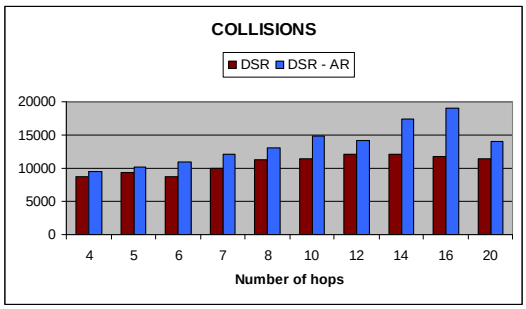
Routing overhead



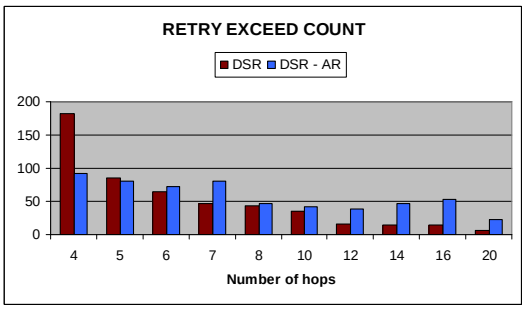
Lost packets



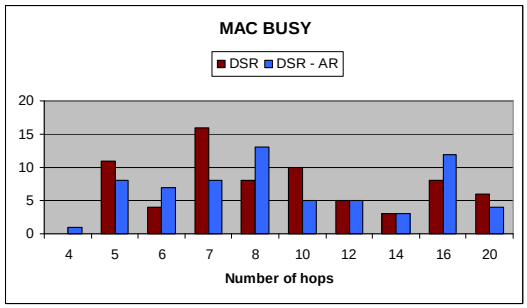
Total of MAC errors



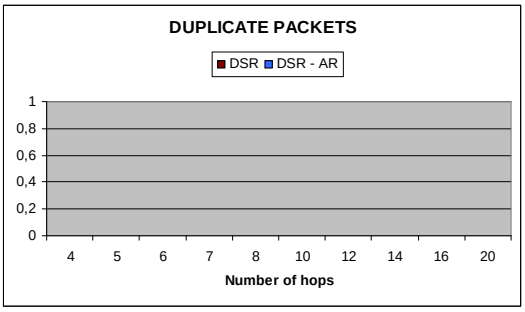
Number of collisions at MAC layer



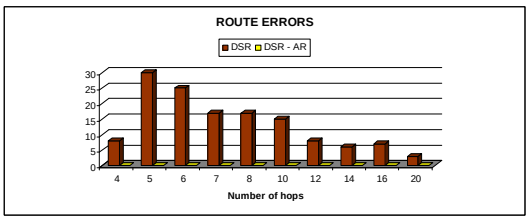
Number of errors due to retry exceed count



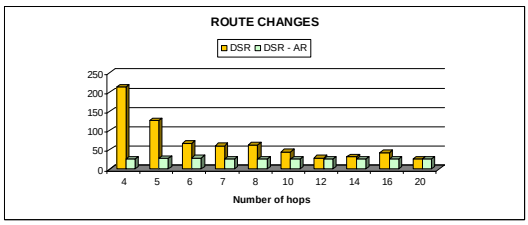
Dropped packets because of MAC busy



Number of dropped packets due to duplicate packets

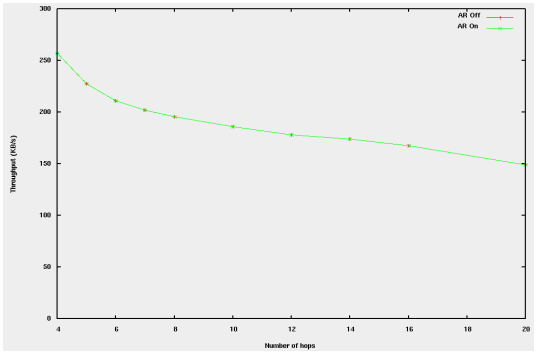


Number of route errors in DSR

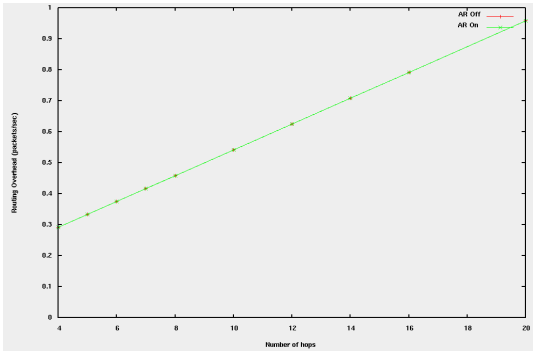


Aggregate of the number of changes in the length of the route

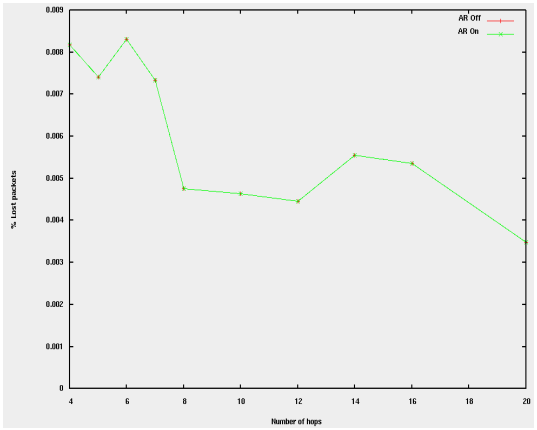
Dampem policy by Nahm β2 (AR off) and DSR-AR in collaboration with β2 (AR on)



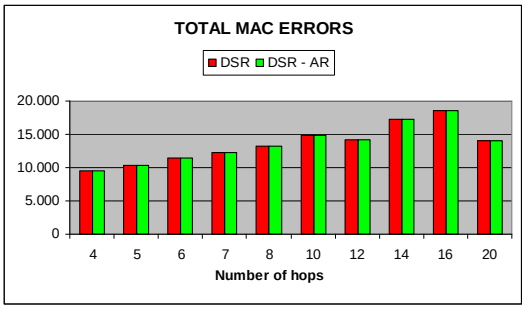
Throughput



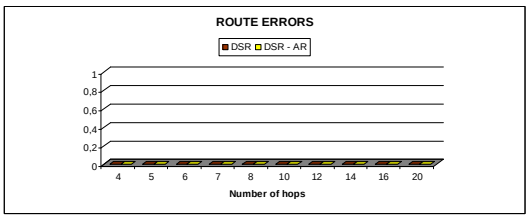
Routing overhead



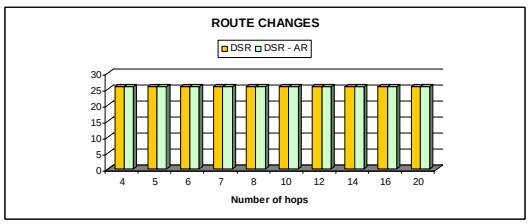
Lost packets



Total of MAC errors



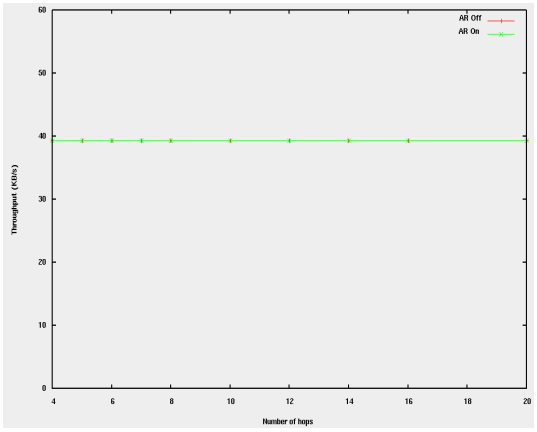
Number of route errors in DSR



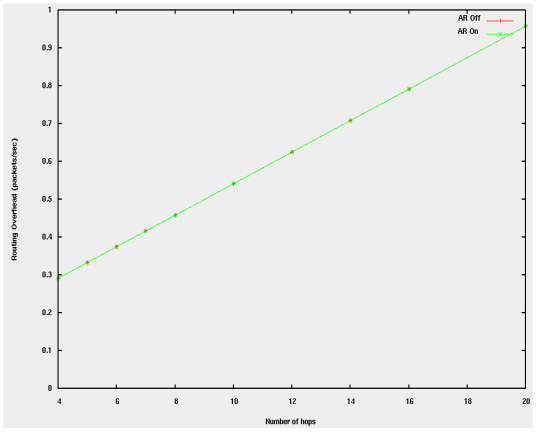
Aggregate of the number of changes in the length of the route

A.1.2 UDP

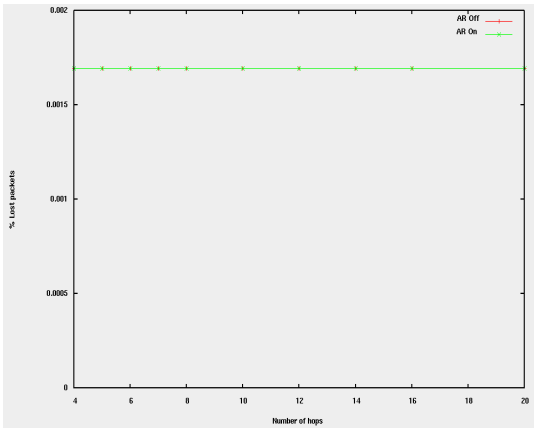
Two UDP flows



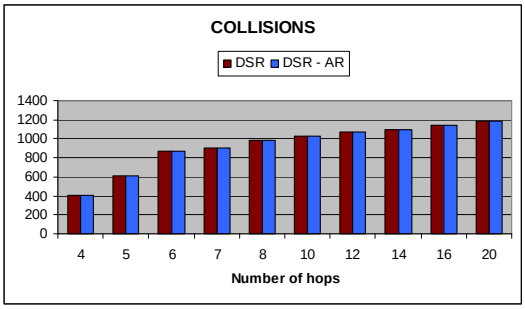
Throughput



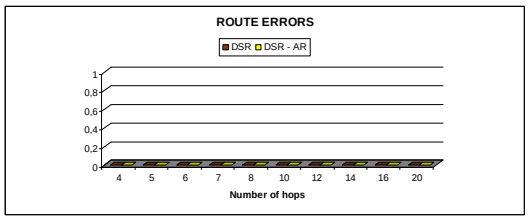
Routing overhead



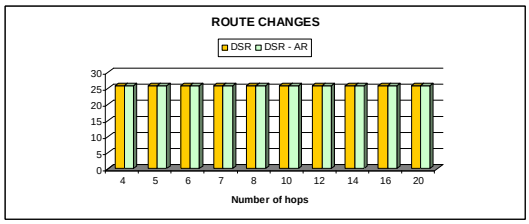
Lost packets



Dropped packets due to collisions

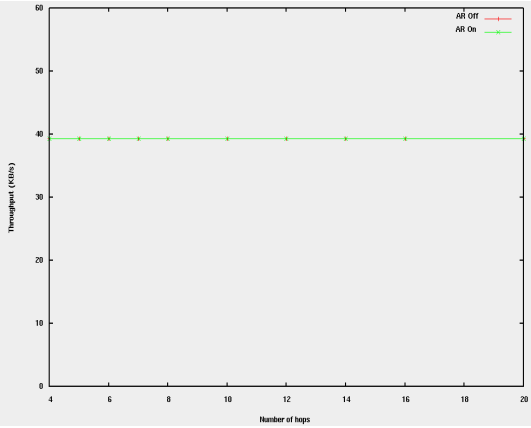


Number of route errors

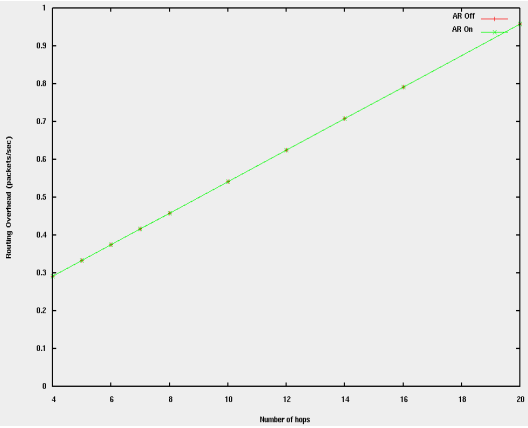


Number of route changes

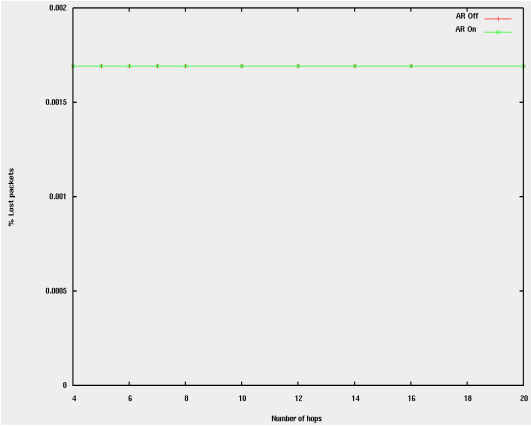
Dampem policy by Nahm β2 (AR off) and DSR-AR in collaboration with β2 (AR on)



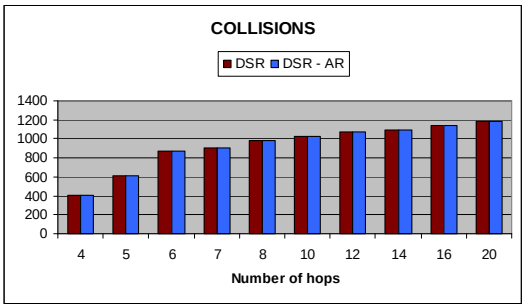
Throughput



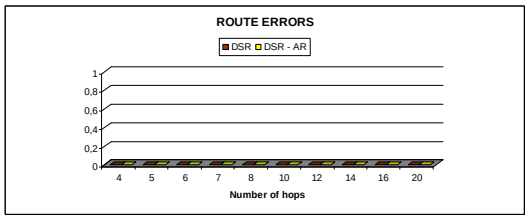
Routing overhead



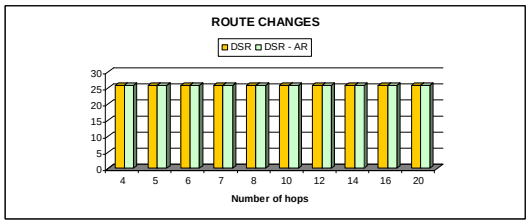
Lost packets



Dropped packets due to collisions

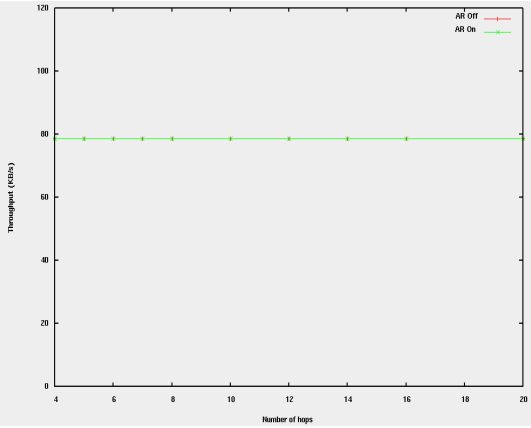


Number of route errors

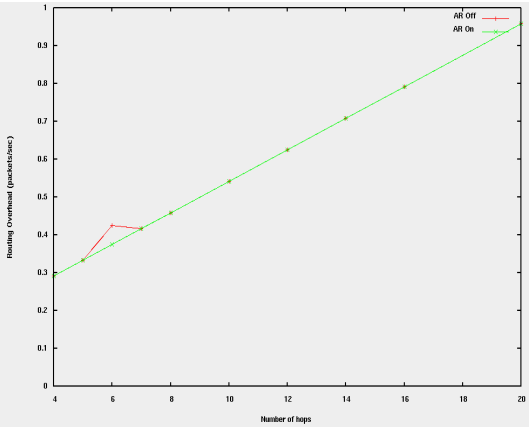


Number of route changes

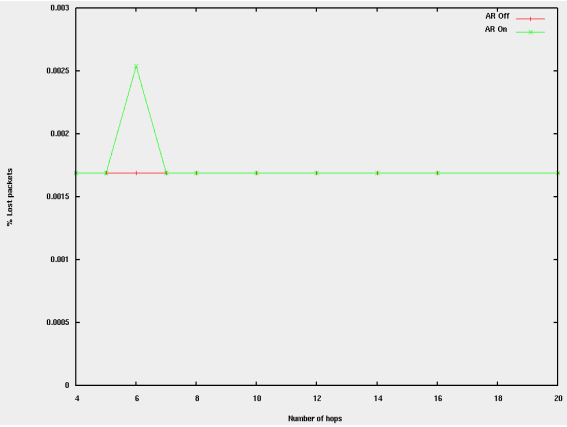
Four UDP flows



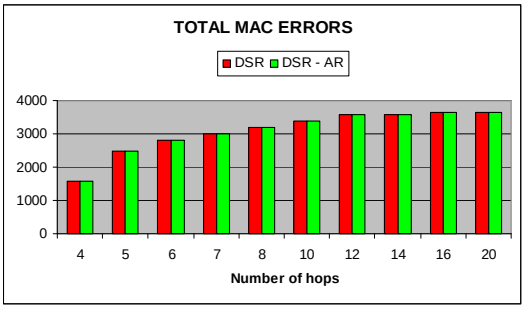
Throughput



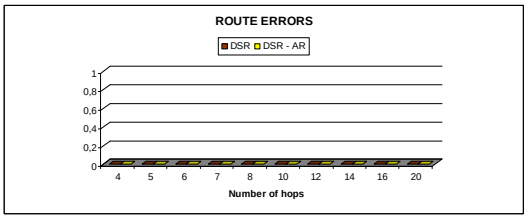
Routing overhead



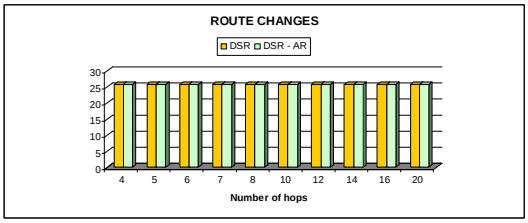
Lost packets



Number of MAC error



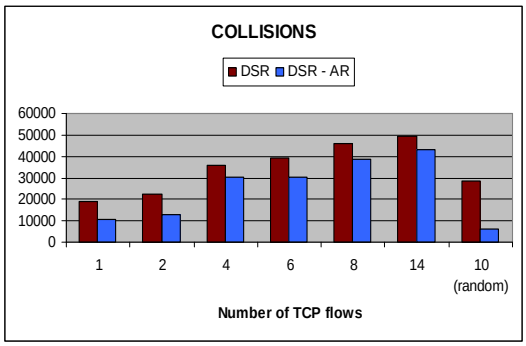
Number of route errors



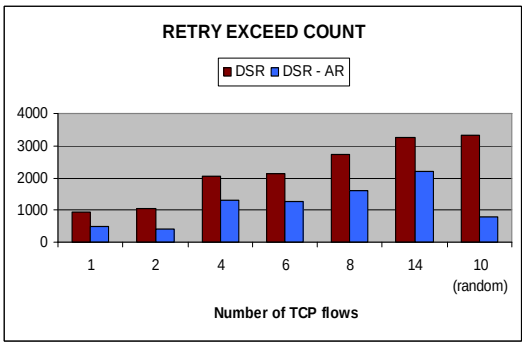
Number of route changes

A.2 Grid 7x7 scenario

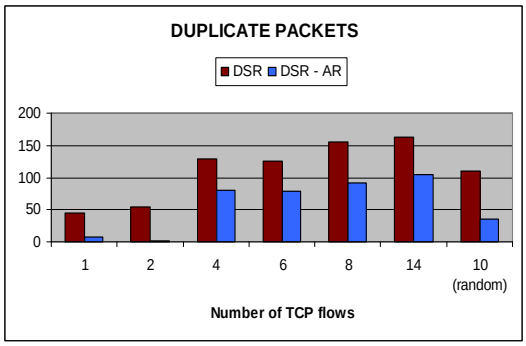
A.2.1 TCP



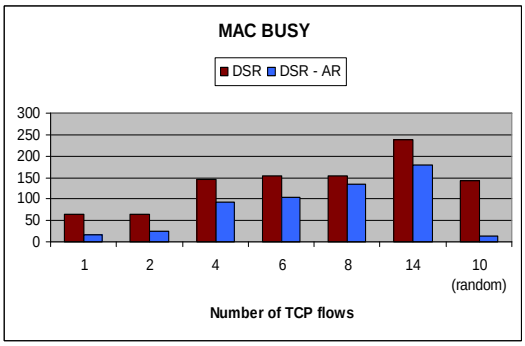
Dropped packets due to collisions



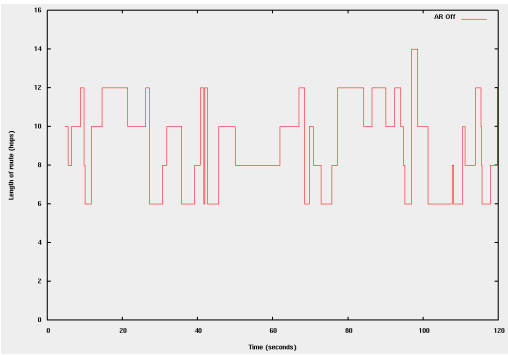
Dropped packets due to retry exceed count



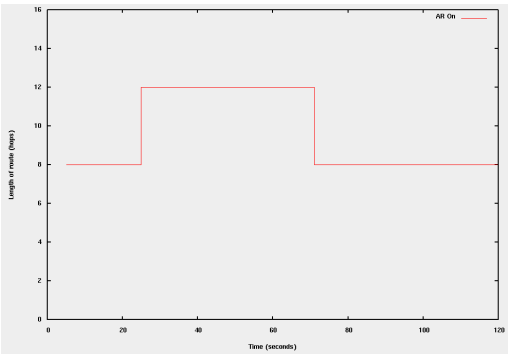
Dropped packets due to duplicate packets



Dropped packets due to MAC busy

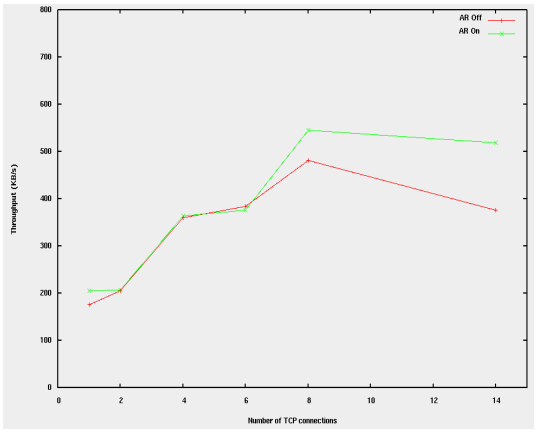


Length of route (number of hops) for six cross flows normal DSR

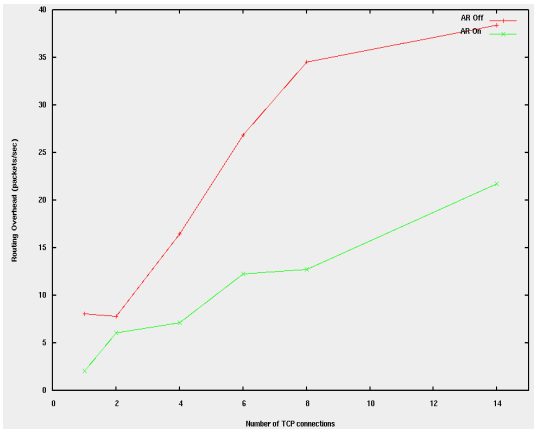


Length of route (number of hops) for six cross flows DSR-AR

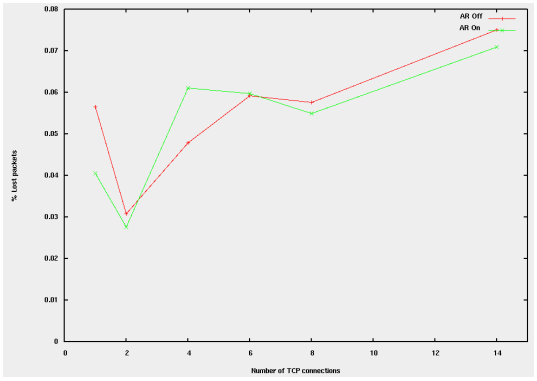
Dampem policy by Nahm β2 (AR off) and DSR-AR in collaboration with β2 (AR on)



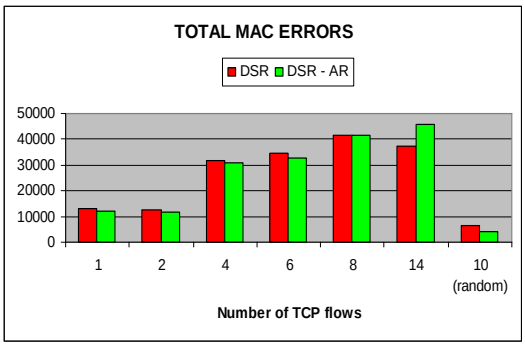
Throughput



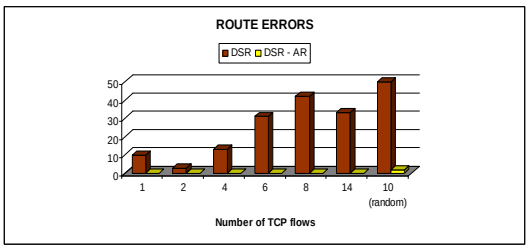
Routing overhead



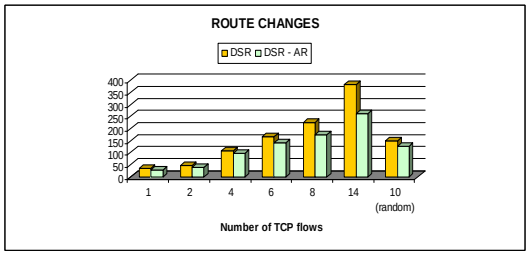
Lost packets



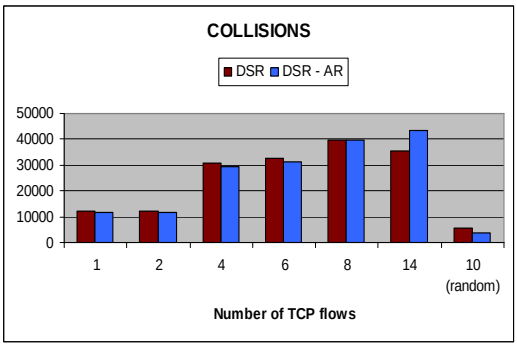
Dropped packets due to MAC errors



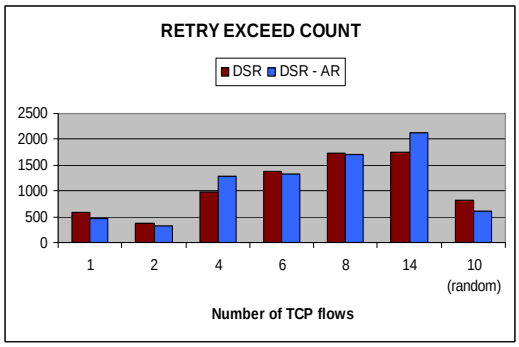
Number of route errors



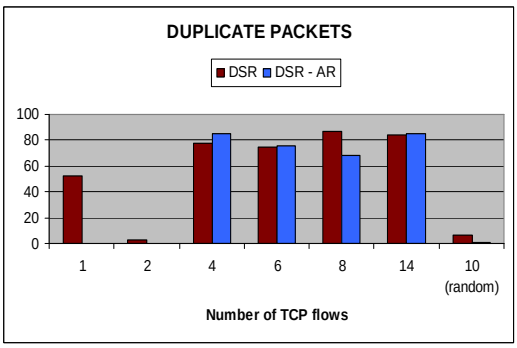
Number of route changes



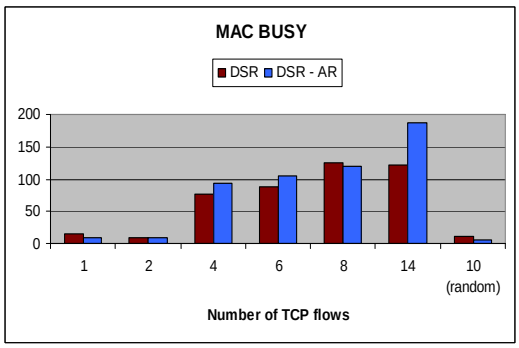
Dropped packets due to collisions



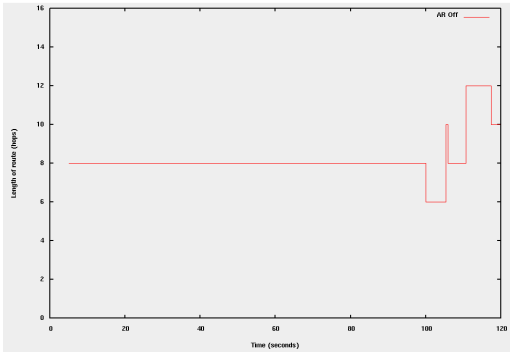
Dropped packets due to retry exceed count



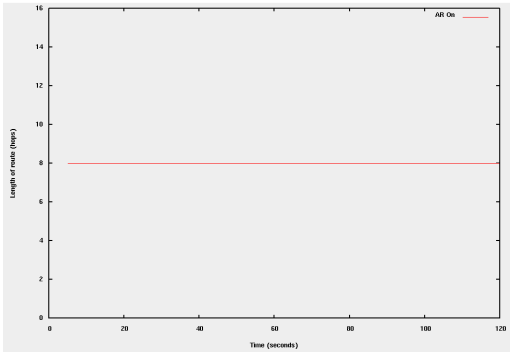
Dropped packets due to duplicate packets



Dropped packets due to MAC busy



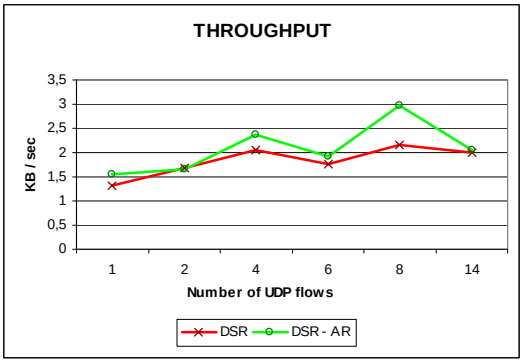
Length of the route (number of hops) for six cross flows DSR β2



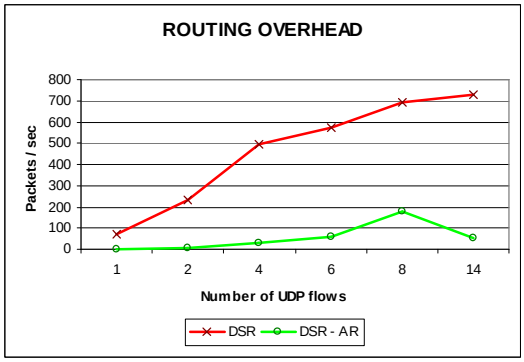
Length of the route (number of hops) for six cross flows DSR-AR β2

A.2.2 UDP

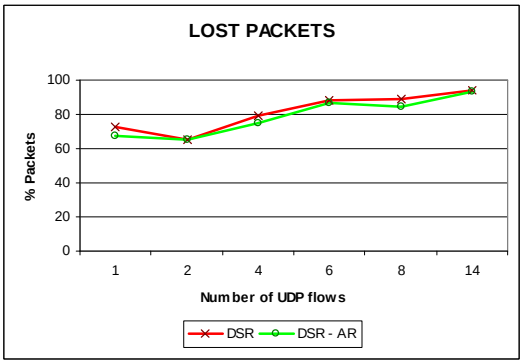
(300kb CBR)



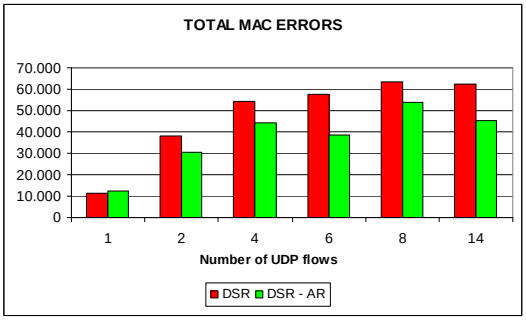
Throughput



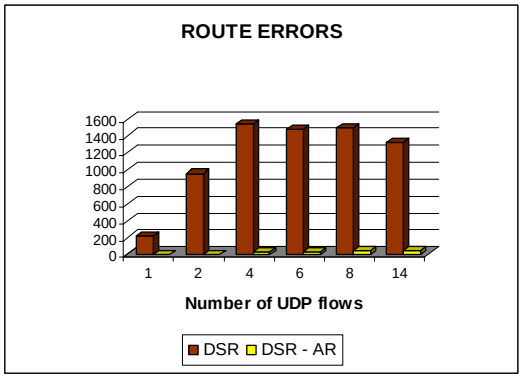
Routing overhead



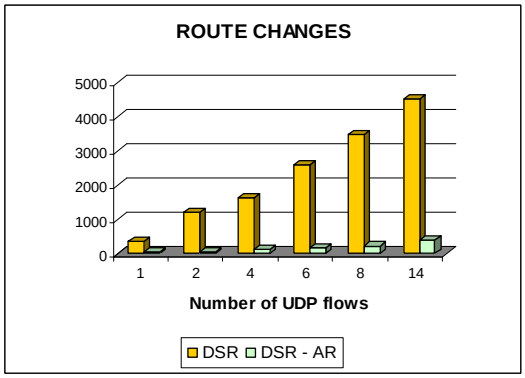
Lost packets



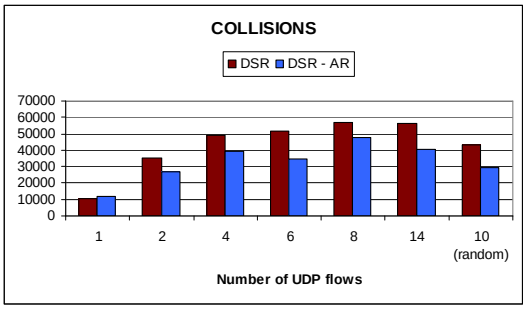
Total MAC errors



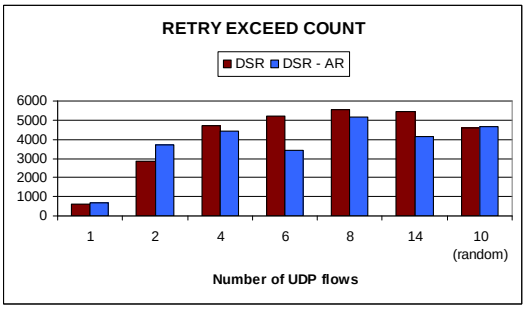
Number of route errors



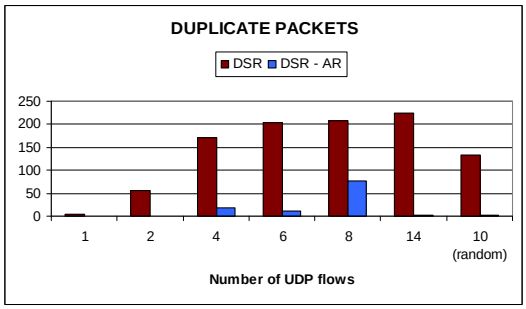
Number of route changes



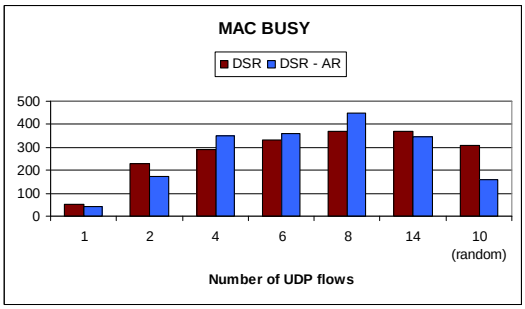
Dropped packets due to collisions



Dropped packets due to retry exceed count

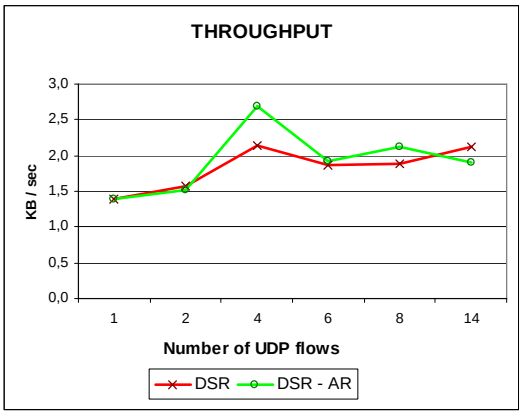


Dropped packets due to duplicate packets

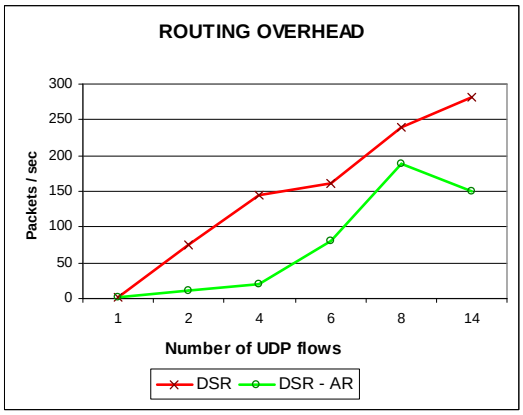


Dropped packets due to MAC busy

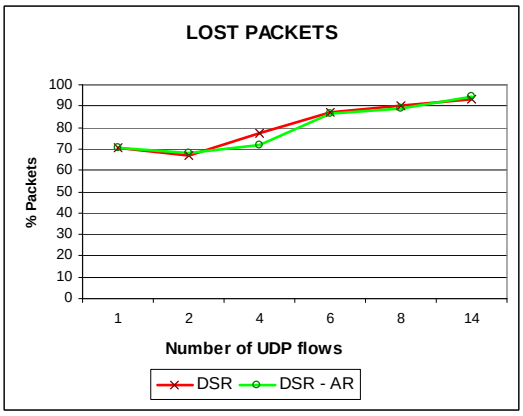
Dampem policy by Nahm β2 (AR off) and DSR-AR in collaboration with β2 (AR on)



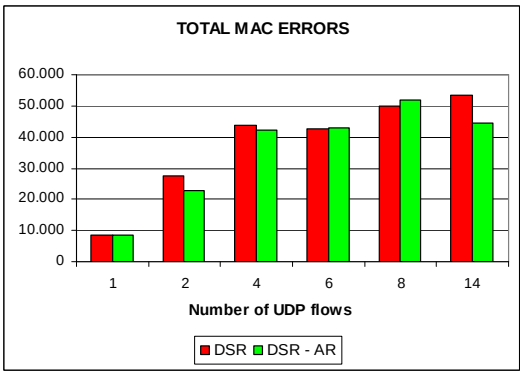
Throughput



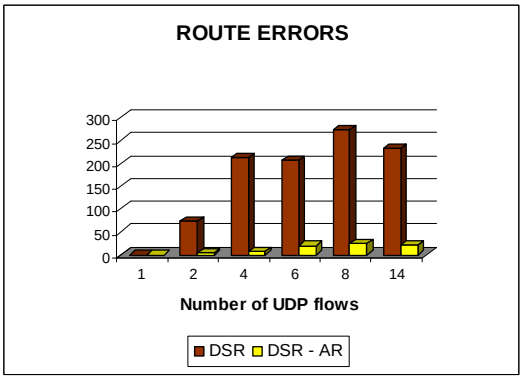
Routing overhead



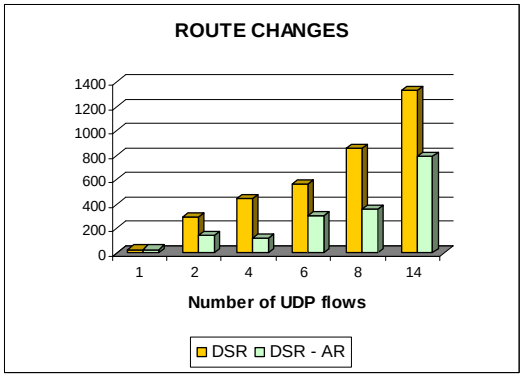
Lost packets



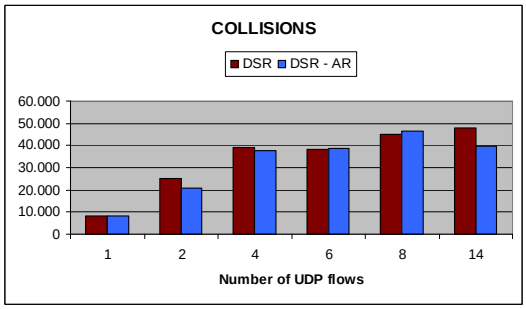
Total dropped packets at MAC layer



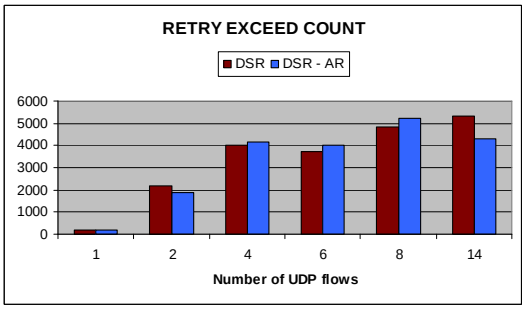
Number of route errors



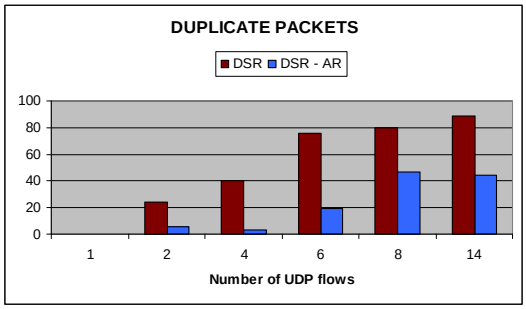
Number of route changes



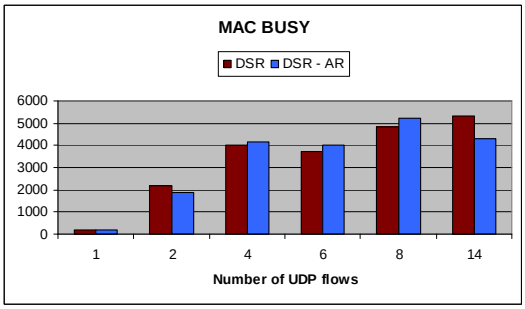
Dropped packets due to collisions



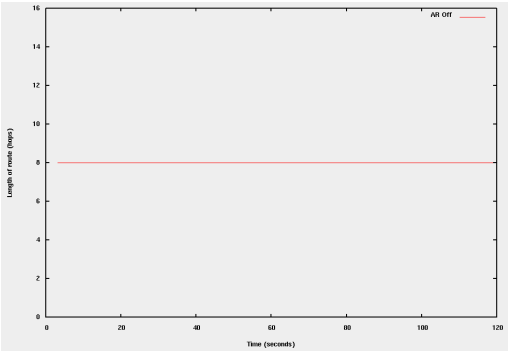
Dropped packets due to retry exceed count



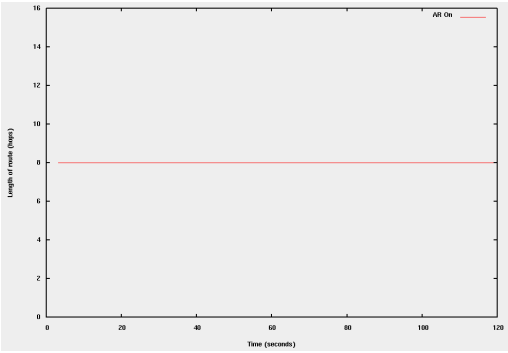
Dropped packets due to duplicate packets



Dropped packets due to MAC busy



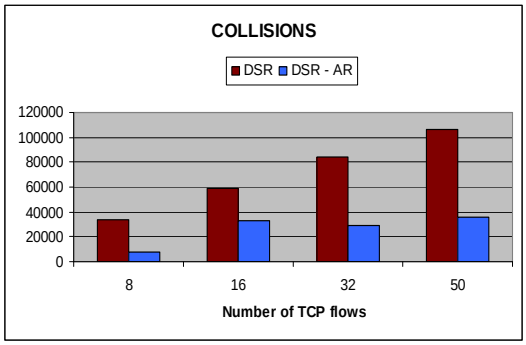
Two cross flows DSR β2



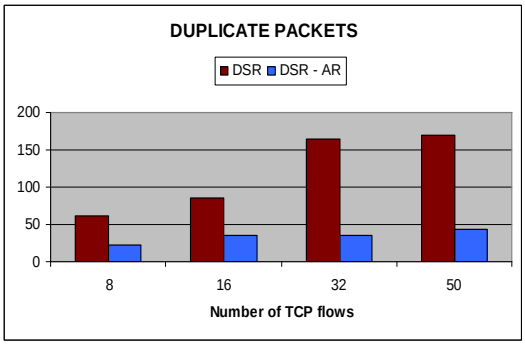
Two cross flows DSR-AR

A.3 Random waypoint scenario

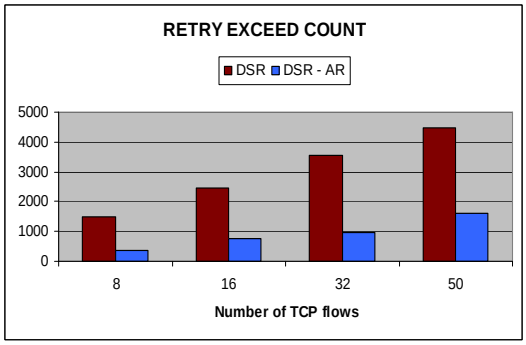
A.3.1 TCP



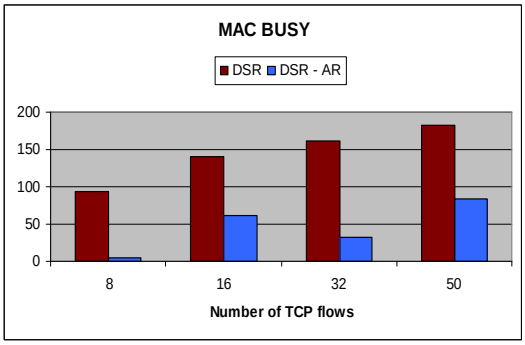
Dropped packets due to collisions



Dropped packets due to duplicate packets

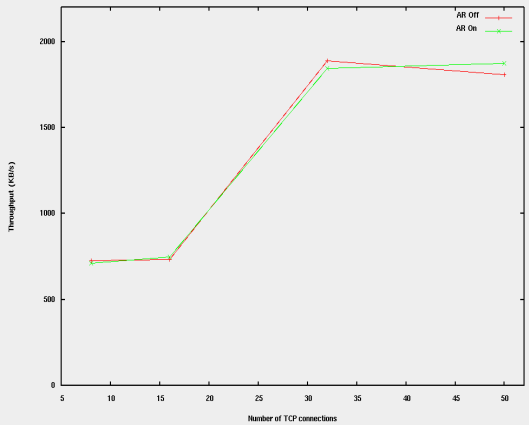


Dropped packets due to retry exceed count

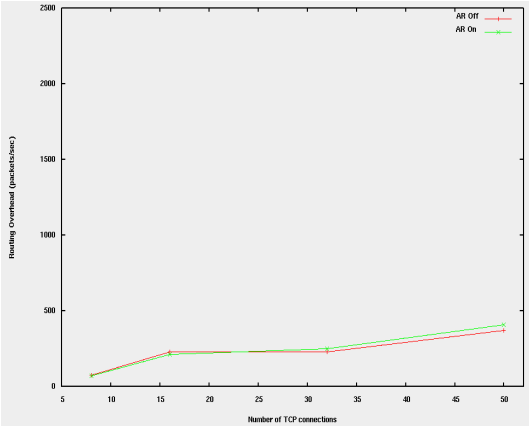


Dropped packets due to MAC busy

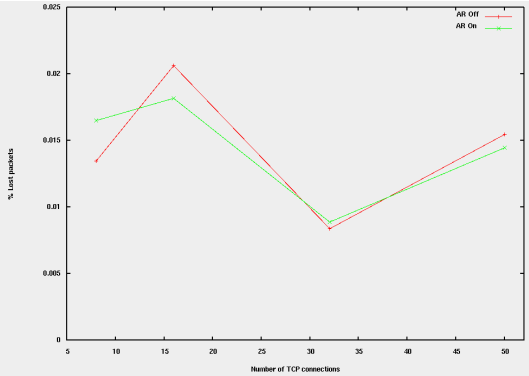
Dampem policy by Nahm β2 (AR off) and DSR-AR in collaboration with β2 (AR on)



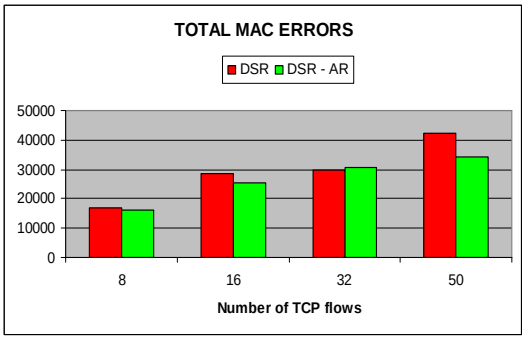
Throughput



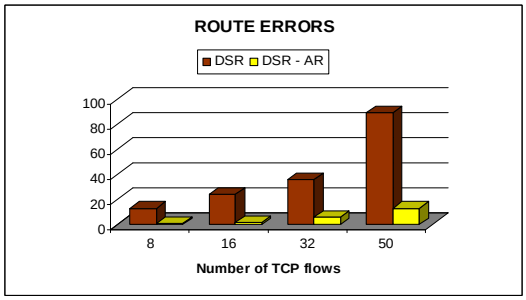
Routing overhead



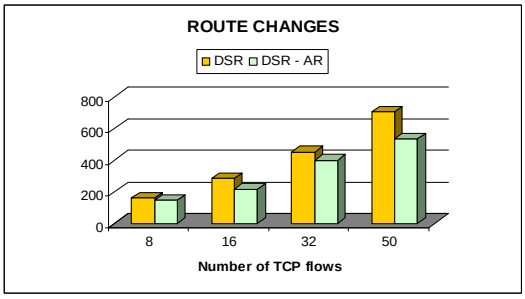
Lost packets



Total dropped packets



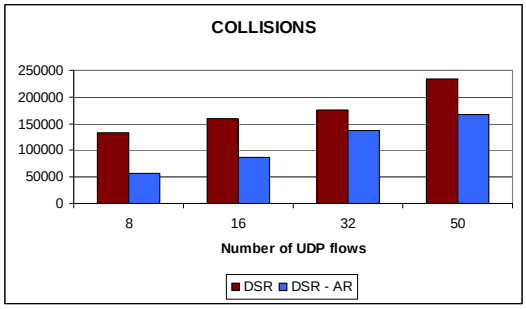
Number of route errors



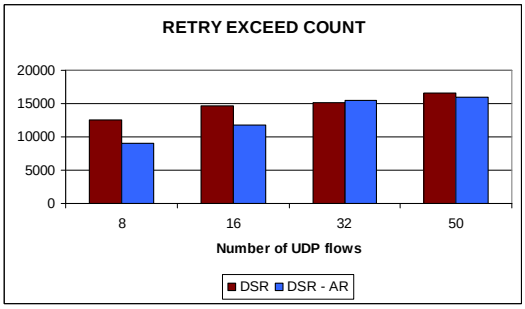
Number of route changes

A.3.2 UDP

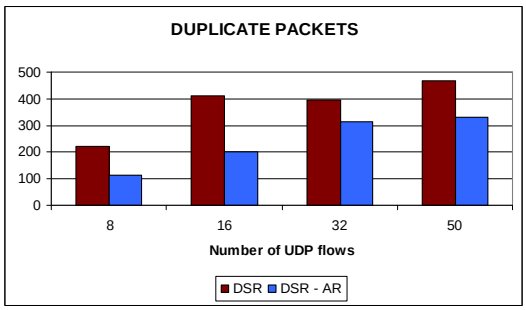
(300Kb)



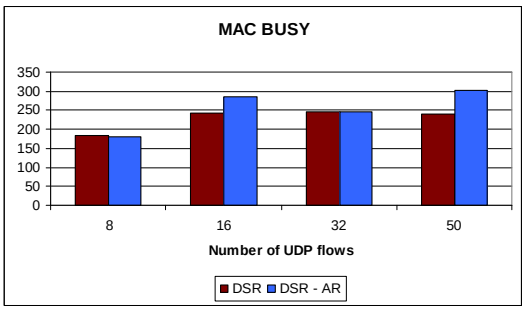
Dropped packets due to collisions



Dropped packets due to retry exceed count

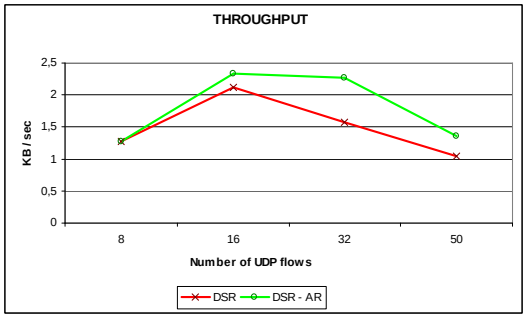


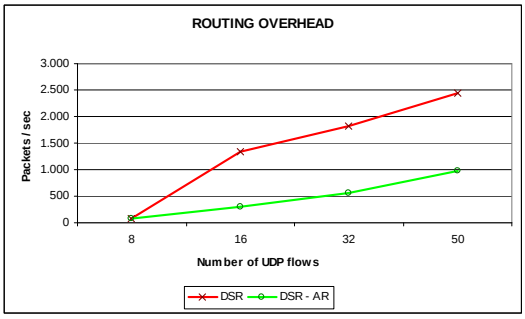
Dropped packets due to duplicate packets

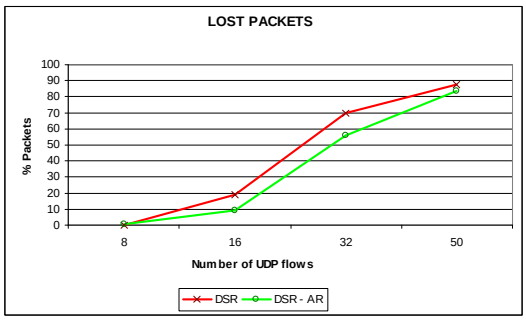


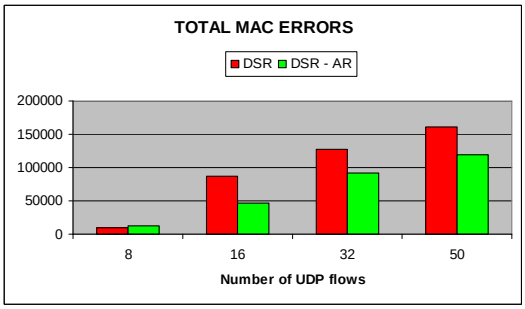
Dropped packets due to MAC busy

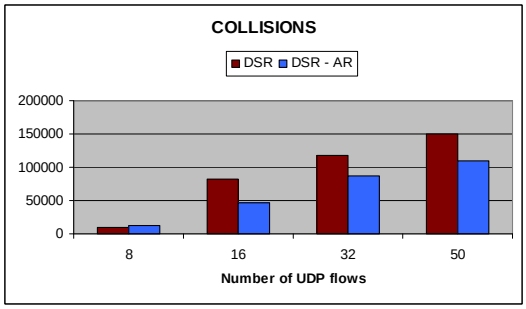
Another random waypoint scenario (300Kb)

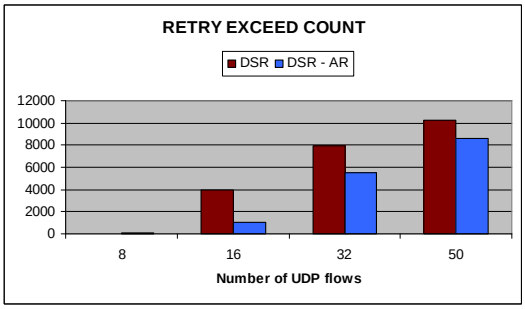


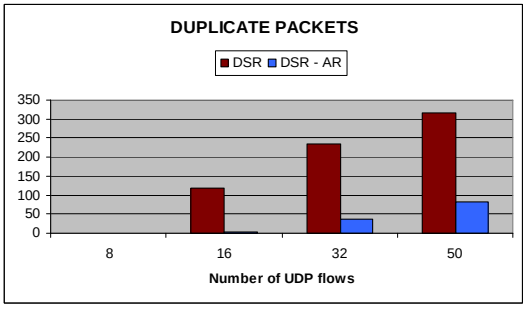


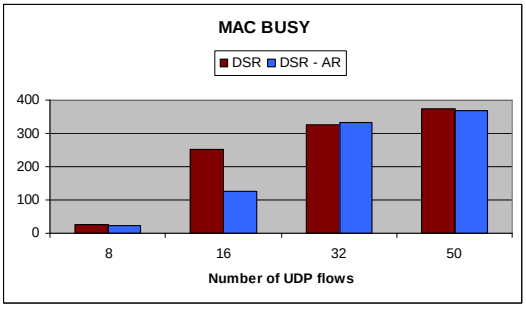




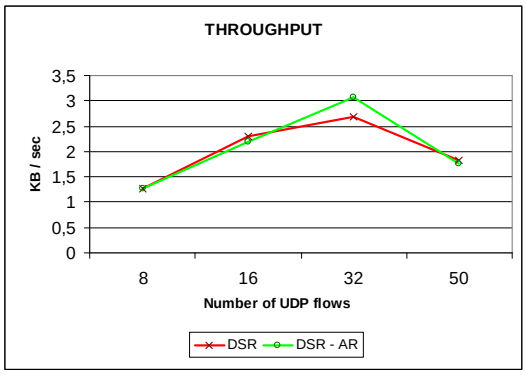




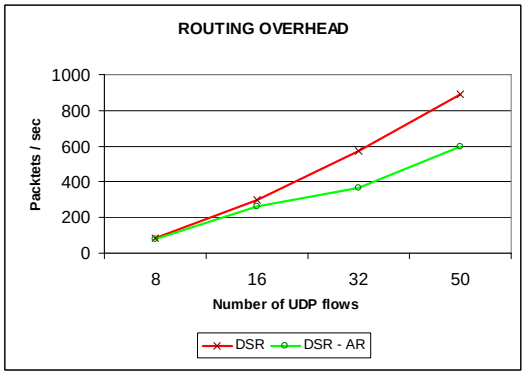




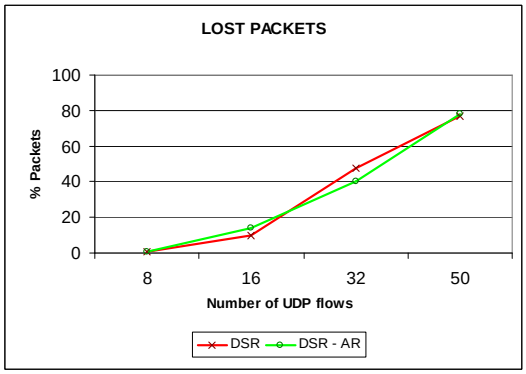
Dampem policy by Nahm β2 (AR off) and DSR-AR in collaboration with β2 (AR on)



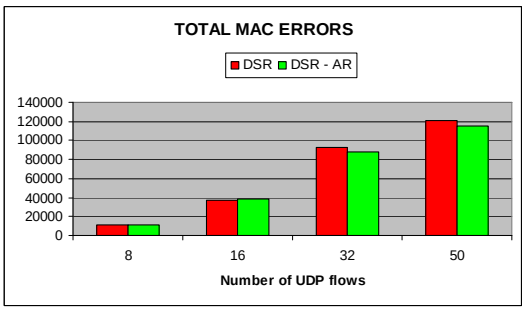
Throughput



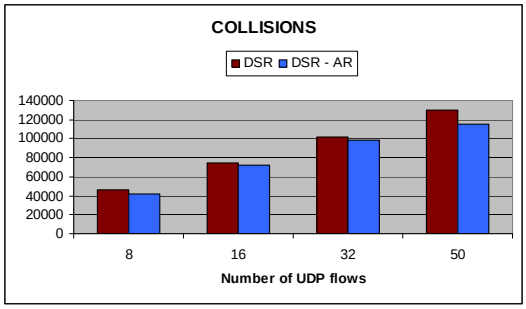
Routing overhead



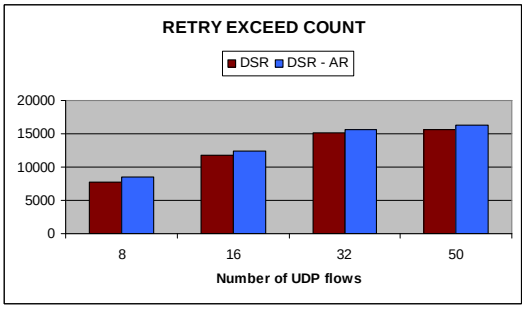
Lost packets



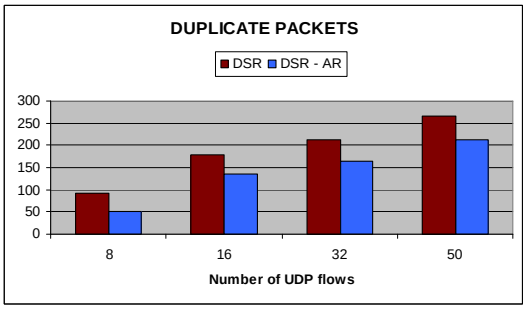
Total dropped packets



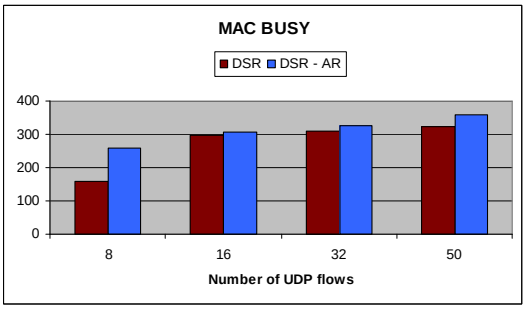
Dropped packets due to collisions



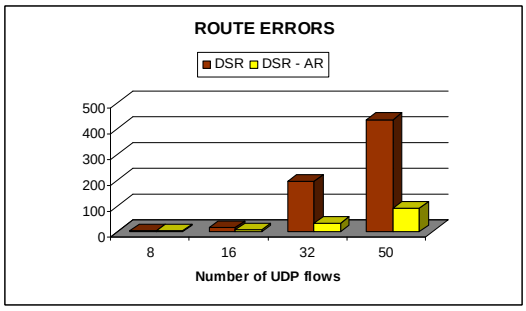
Dropped packets due to retry exceed count



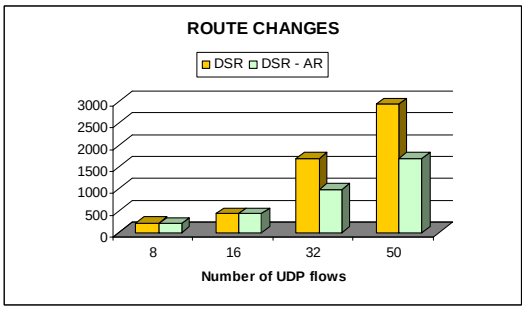
Dropped packets due to duplicate packets



Dropped packets due to MAC busy



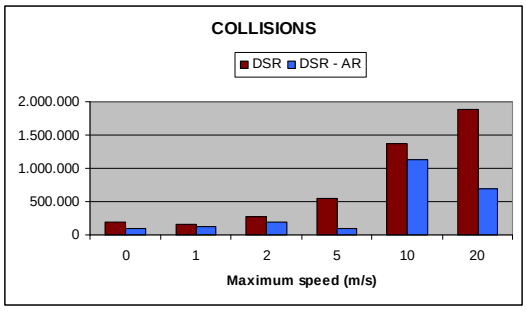
Number of route errors



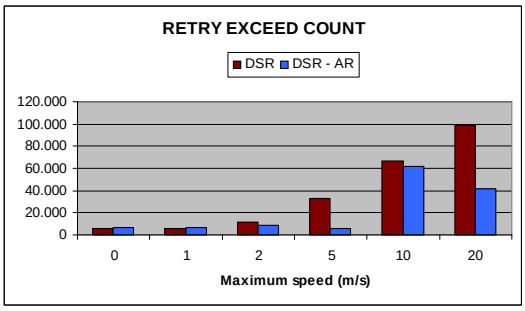
Number of route changes

A.4 Manhattan scenario

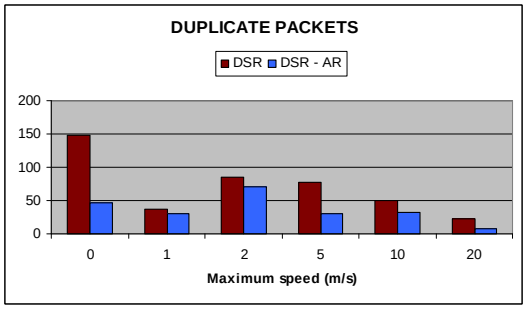
A.4.1 TCP



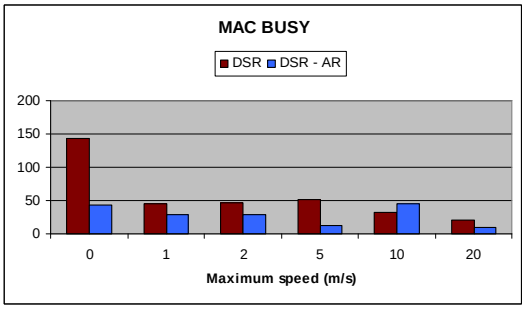
Dropped packets due to collisions



Dropped packets due to retry exceed count

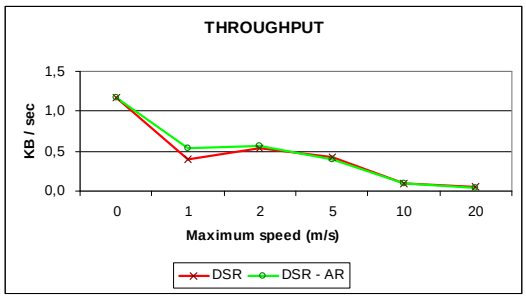


Dropped packets due to duplicate packets

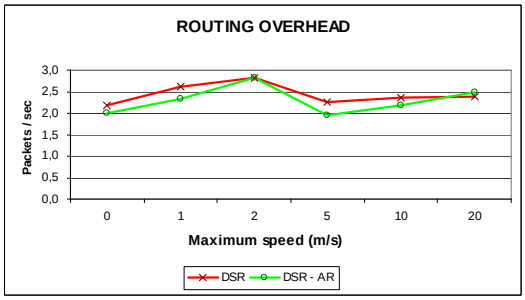


Dropped packets due to MAC busy

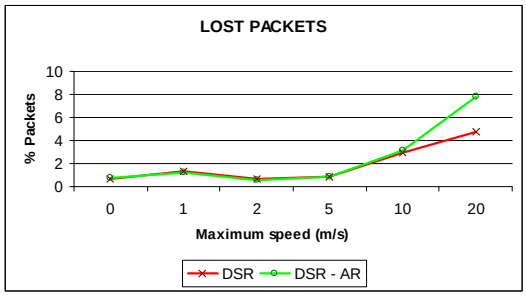
Dampem policy by Nahm β2 (AR off) and DSR-AR in collaboration with β2 (AR on)



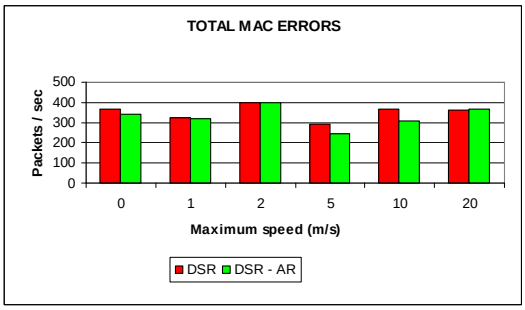
Throughput



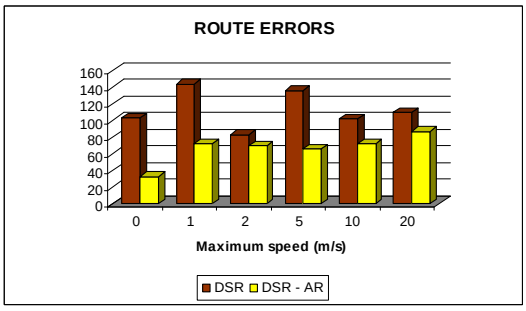
Routing overhead



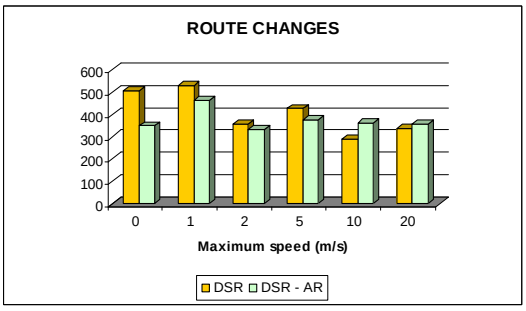
Lost packets



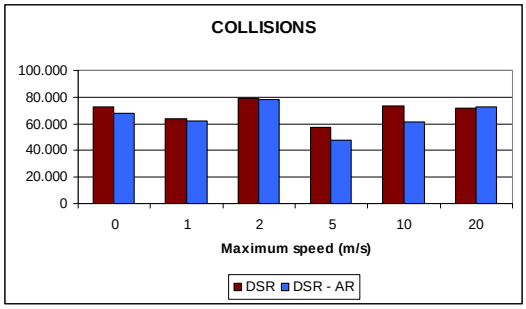
Total dropped packets



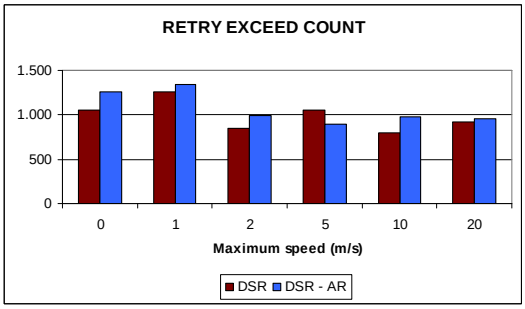
Number of route errors



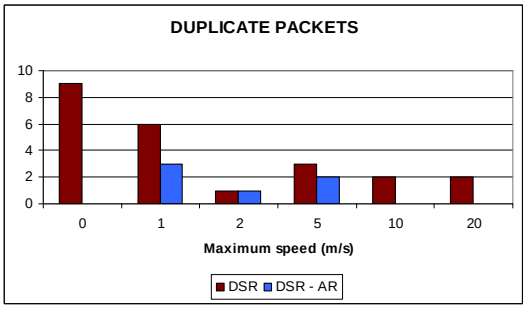
Number of route changes



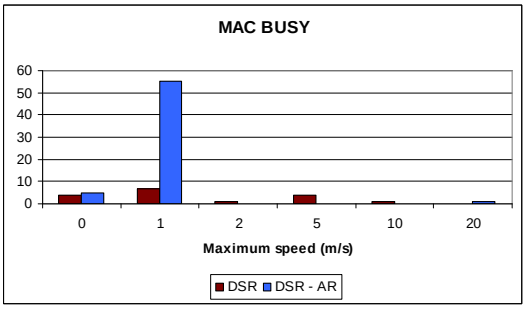
Dropped packets due to collisions



Dropped packets due to retry exceed count

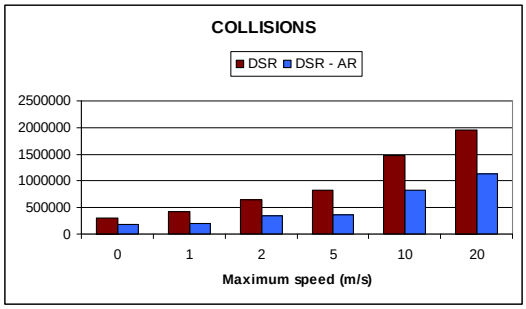


Dropped packets due to duplicate packets

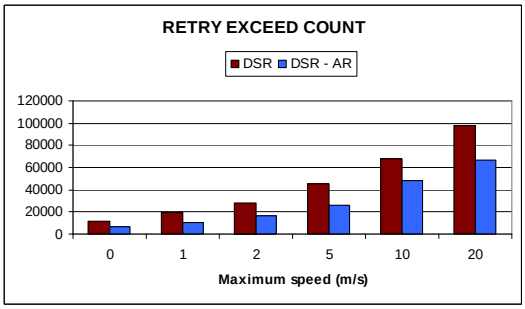


Dropped packets due to MAC busy

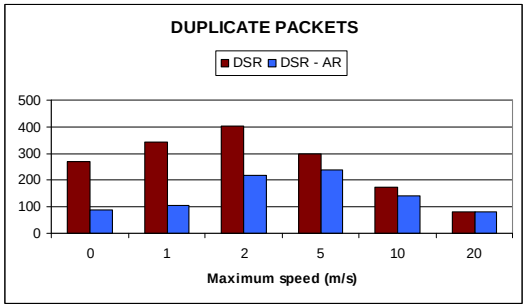
A.4.2 UDP



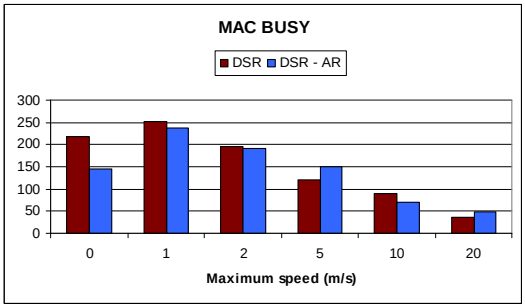
Dropped packets due to collisions



Dropped packets due to retry exceed count

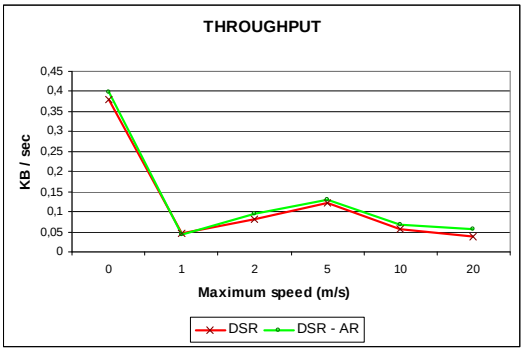


Dropped packets due to duplicate packets

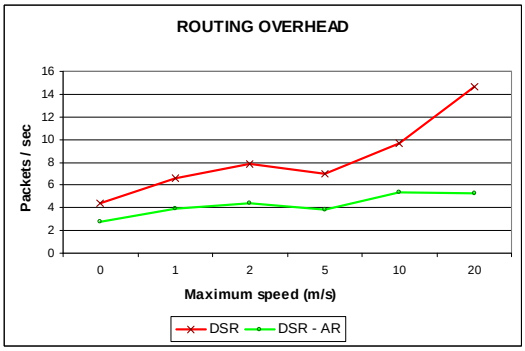


Dropped packets due to MAC busy

Dampem policy by Nahm β2 (AR off) and DSR-AR in collaboration with β2 (AR on)

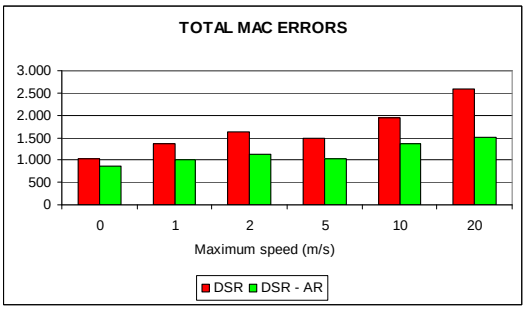


Throughput

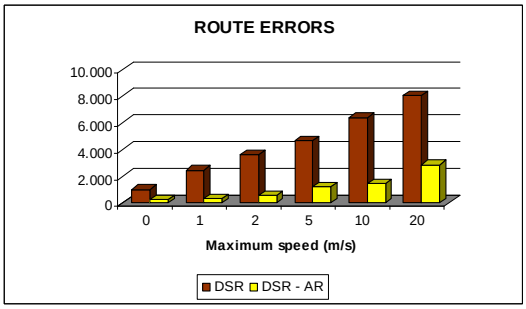


Routing overhead

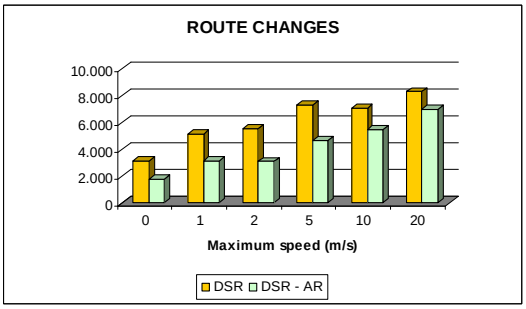
Lost packets



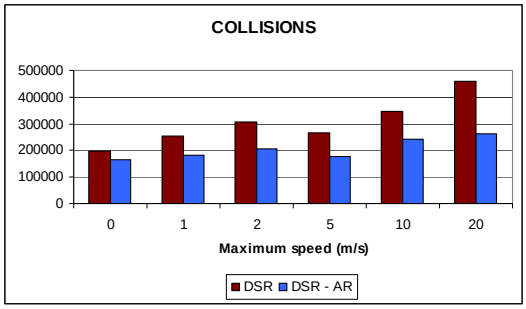
Total dropped packets



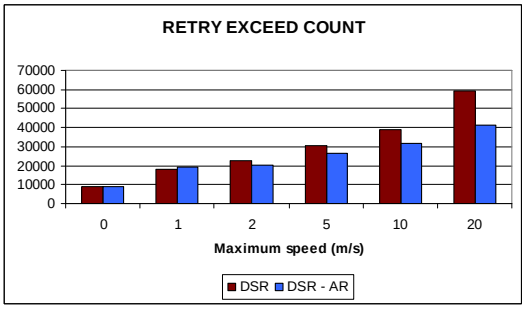
Number of route errors



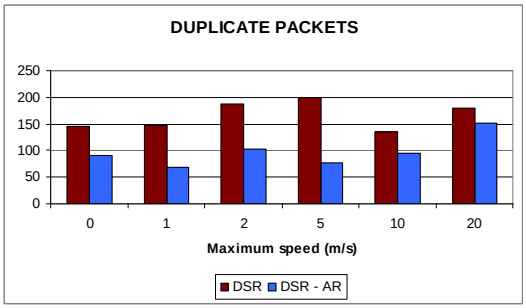
Number of route changes



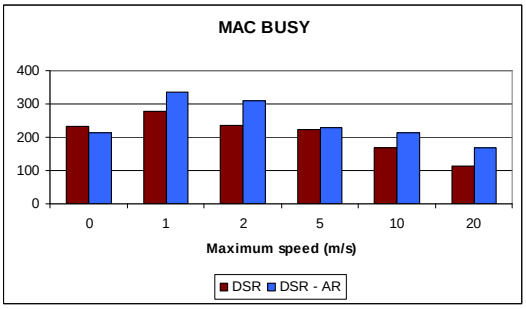
Dropped packets due to collisions



Dropped packets due to retry exceed count



Dropped packets due to duplicate packets



Dropped packets due to MAC busy

Appendix B

Setup files common for all scenarios

- **TCP** (main-tcp.tcl)

```
# ======================================================================
# Default Script Options
# ======================================================================
set opt(chan)       Channel/WirelessChannel
set opt(prop)       Propagation/TwoRayGround
set opt(netif)      Phy/WirelessPhy
set opt(mac)        Mac/802_11
set opt(ifq)        CMUPriQueue ;# for dsr
set opt(ifqlen)     50
set opt(ll)     LL
set opt(ant)            Antenna/OmniAntenna

set opt(x)      6000        ;# X dimension of the topography
set opt(y)      3000        ;# Y dimension of the topography
set opt(cp)     ""; # traffic TCP/TFRC/etc
set opt(sc)     ""; # topology scenario

set opt(nn)         0      ;# number of nodes -- changes according to opt(sc)
set opt(seed)       0.0
set opt(stop)       200.0       ;# simulation time
set opt(tr)         out.tr      ;# trace file
set opt(rp)             dsr     ;# routing protocol script (dsr or dsdv)
set opt(lm)             "off"       ;# log movement

set opt(err) ""             ;# if needed, check MyErrorProc below
set opt(alpha)    1.0
set opt(beta)       1       ;# DSR beta (0: static, 1: DSR, >2:DSR++DAMPEN)
set opt(gamma)    0        ;# DSR gamma (0 without signal power, >0 using signal power to
differentiate retry count exceed from link error)
set opt(rate)  2e6
set opt(dist)  250   ;# receiving range

set opt(ntcp)       0       ;# opt(cp) related parameter: number of TCPs
set opt(hops)       1       ;# opt(cp) related parameter: E2E hops
# ======================================================================

set AgentTrace          ON
set RouterTrace         ON
set MacTrace            ON
CMUTrace set newtrace_  1
```

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

Agent/Null set sport_       0
Agent/Null set dport_       0
Agent/CBR set sport_        0
Agent/CBR set dport_        0
Agent/TCPSink set sport_    0
Agent/TCPSink set dport_    0
Agent/TCP set sport_        0
Agent/TCP set dport_        0
Agent/TCP set packetSize_   1024
Queue/DropTail/PriQueue set Prefer_Routing_Protocols    1

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

# Initialize the SharedMedia interface with parameters to make
# it work like the 914MHz Lucent WaveLAN DSSS radio interface
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11; # 550m
Phy/WirelessPhy set RXThresh_ 3.652e-10; # 250m
Phy/WirelessPhy set Pt_ 0.281838
Phy/WirelessPhy set freq_ 2.4e+9
Phy/WirelessPhy set L_ 1.0

ErrorModel set enable_ 1
ErrorModel set markecn_ false
ErrorModel set bandwidth_ 2Mb


# ======================================================================

#source [lindex $argv 0]; # override the default settings

# ======================================================================

proc usage { argv0 }  {
    puts "Usage: $argv0"
    puts "\tmandatory arguments:"
    puts "\t\t\[-x MAXX\] \[-y MAXY\]"
    puts "\toptional arguments:"
    puts "\t\t\[-cp conn pattern\] \[-sc scenario\] \[-nn nodes\]"
    puts "\t\t\[-seed seed\] \[-stop sec\] \[-tr tracefile\]\n"
```

```
}

proc getopt {argc argv} {
    global opt
    lappend optlist cp nn seed sc stop tr x y ntcp hops gamma beta alpha qlen rate
    for {set i 0} {$i < $argc} {incr i} {
        set arg [lindex $argv $i]
        if {[string range $arg 0 0] != "-"} continue

        set name [string range $arg 1 end]
        set opt($name) [lindex $argv [expr $i+1]]
   puts "opt($name) = $opt($name)"
    }
}

proc create_tcp_connection {src dst log_file_prefix start} {
    global ns_ tcp_ node_ iiik tcptrace_ opt
    if ![info exists iiik] {
   set iiik 0
    }
    set tcp_($iiik) [new Agent/TCP/Newreno]
    set tcpsink_($iiik) [new Agent/TCPSink]
    $ns_ attach-agent $node_($src) $tcp_($iiik)
    $ns_ attach-agent $node_($dst) $tcpsink_($iiik)
    $ns_ connect $tcp_($iiik) $tcpsink_($iiik)

    $tcp_($iiik) set fid_ 1
    $tcp_($iiik) set window_ 64
    $tcp_($iiik) set nodeid_ [$node_($src) id]
    $tcpsink_($iiik) set nodeid_ [$node_($src) id]
    if { $log_file_prefix != "" } {
   $tcpsink_($iiik) set total_bytes_ 0
   $ns_ register_record $tcpsink_($iiik) $log_file_prefix$iiik.tr
   set tcptrace_($iiik) [open $log_file_prefix$iiik.log w]
   $tcp_($iiik) set trace_all_oneline_ false
   $tcp_($iiik) trace cwnd_
   $tcp_($iiik) trace rtt_
   $tcp_($iiik) trace srtt_
   $tcp_($iiik) trace ssthresh_
   $tcp_($iiik) attach $tcptrace_($iiik)
   $ns_ at $opt(stop) "flush $tcptrace_($iiik)"
   $ns_ at $opt(stop) "close $tcptrace_($iiik)"
    }
    set ftp_($iiik) [$tcp_($iiik) attach-source FTP]
    $ns_ at $start "$ftp_($iiik) start"
#    $ns_ rtmodel-at $start down $node_($src)
    incr iiik
}
proc MyErrorProc {} {
    set errObj [new ErrorModel]
    $errObj FECstrength 0
```

```
    $errObj set rate_ 0.001
    $errObj set unit pkt
    return $errObj
}



Simulator instproc register_record {agent file} {
    $self instvar agent_list log_file_list
    lappend agent_list $agent
    lappend log_file_list [open $file w]
}

set total_thr_file [open "thr.log" w]

Simulator instproc record {} {
    $self instvar agent_list log_file_list
    global total_thr_file opt
    set nnn [llength $agent_list]
    set now [$self now]
    set interval $opt(stop)
    set bw_tcp 0
    set bw_tfrc 0
    for {set i 0} {$i < $nnn} {incr i} {
        set agent [lindex $agent_list $i]
        set bytes [$agent set total_bytes_]
        set file [lindex $log_file_list $i]
        set bw [expr $bytes/$interval*8.0/1024.0]
   set bw_tcp [expr $bw_tcp + $bw]
        $agent set total_bytes_ 0
        puts $file "$now $bw"
    }
    if { $interval == $opt(stop) } {
   if { $now == $opt(stop) } {
       puts $total_thr_file "$bw_tcp"
   }
    } else {
   puts $total_thr_file "$now $bw_tcp"
    }
    $self at [expr $now+$interval] "$self record"
}

proc log-movement {} {
    global logtimer ns_ ns

    set ns $ns_
    source /tcl/timer.tcl
    Class LogTimer -superclass Timer
    LogTimer instproc timeout {} {
    global opt node_;
    for {set i 0} {$i < $opt(nn)} {incr i} {
```

```
        $node_($i) log-movement
    }
    $self sched 0.1
    }
    set logtimer [new LogTimer]
    $logtimer sched 0.1
}



proc finish {} {
   global ns_
   $ns_ flush-trace
   #close $nf

}

# ======================================================================
# Main Program
# ======================================================================

getopt $argc $argv

if { $opt(x) == 0 || $opt(y) == 0 } {
    usage $argv0
    exit 1
}

if {$opt(seed) > 0} {
    puts "Seeding Random number generator with $opt(seed)\n"
    ns-random $opt(seed)
}

#
# Initialize Global Variables
#
set ns_     [new Simulator]
set chan    [new $opt(chan)]
set prop    [new $opt(prop)]
set topo    [new Topography]
set tracefd [open $opt(tr) w]
#set nf     [open $opt(namfile) w]

#$ns_ namtrace-all $nf

$topo load_flatgrid $opt(x) $opt(y)

$prop topography $topo

#
# Create God
```

```
#
set god_ [create-god $opt(nn)]


#
# log the mobile nodes movements if desired
#
if { $opt(lm) == "on" } {
    log-movement
}

if { $opt(dist) == 60 } {
   Phy/WirelessPhy set CSThresh_ 1.65011e-9; # 130m #140!!! change
   Phy/WirelessPhy set RXThresh_ 7.74635e-9; # 60m
}

if { [string compare $opt(rp) "dsr"] == 0} {
    Agent/DSRAgent set beta_ $opt(beta)
    Agent/DSRAgent set gamma_ $opt(gamma)
    Mac/802_11 set basicRate_ $opt(rate)
    Mac/802_11 set dataRate_  $opt(rate)
    Phy/WirelessPhy set bandwidth_ $opt(rate)
    Phy/WirelessPhy set Rb_   $opt(rate)

    for {set i 0} {$i < $opt(nn) } {incr i} {
        dsr-create-mobile-node $i
    }
} elseif { [string compare $opt(rp) "dsdv"] == 0} {
    Mac/802_11 set basicRate_ $opt(rate)
    Mac/802_11 set dataRate_  $opt(rate)
    Phy/WirelessPhy set bandwidth_ $opt(rate)
    Phy/WirelessPhy set Rb_   $opt(rate)

    for {set i 0} {$i < $opt(nn) } {incr i} {
        dsdv-create-mobile-node $i
    }
}

#
# Source the Connection and Movement scripts
#
if { $opt(cp) == "" } {
    puts "*** NOTE: no connection pattern specified."
        set opt(cp) "none"
} else {
    Agent/TCP set increase_num_ $opt(alpha)
    puts "Loading connection pattern... $opt(cp)"
    source $opt(cp)
}

#
```

```
# Tell all the nodes when the simulation ends
#
for {set i 0} {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).000000001 "$node_($i) reset";
}
$ns_ at $opt(stop).00000001 "puts \"NS EXITING...\" ; finish ; $ns_ halt"



if { $opt(sc) == "" } {
    puts "*** NOTE: no scenario file specified."
        set opt(sc) "none"
} else {
    puts "Loading scenario file... $opt(sc)"
    source $opt(sc)
    puts "Load complete..."
}

$ns_ at 0.0 "$ns_ record"
puts "Starting Simulation..."
$ns_ run
```

- **UDP** (main-udp.tcl)

```
# ======================================================================
# Default Script Options
# ======================================================================
set opt(chan)       Channel/WirelessChannel
set opt(prop)       Propagation/TwoRayGround
set opt(netif)      Phy/WirelessPhy
set opt(mac)        Mac/802_11
set opt(ifq)        CMUPriQueue ;# for dsr
set opt(ifqlen)     50
set opt(ll)     LL
set opt(ant)            Antenna/OmniAntenna

set opt(x)      6000        ;# X dimension of the topography
set opt(y)      3000        ;# Y dimension of the topography
set opt(cp)     ""; # traffic TCP/TFRC/etc
set opt(sc)     ""; # topology scenario

set opt(nn)         0      ;# number of nodes -- changes according to opt(sc)
set opt(seed)       0.0
set opt(stop)       200.0       ;# simulation time
set opt(tr)         out.tr      ;# trace file
set opt(rp)             dsr     ;# routing protocol script (dsr or dsdv)
set opt(lm)             "off"       ;# log movement

set opt(err) ""             ;# if needed, check MyErrorProc below
set opt(alpha)    1.0
set opt(beta)       1       ;# DSR beta (0: static, 1: DSR, >2:DSR++DAMPEN)
set opt(gamma)    0        ;# DSR gamma (0 without signal power, >0 using signal power to
differentiate retry count exceed from link error)
set opt(rate)  2e6
set opt(dist)  250   ;# receiving range

set opt(ntcp)       0       ;# opt(cp) related parameter: number of TCPs
set opt(hops)       1       ;# opt(cp) related parameter: E2E hops
# ======================================================================

set AgentTrace          ON
set RouterTrace         ON
set MacTrace            ON
CMUTrace set newtrace_  1

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

Agent/Null set sport_       0
Agent/Null set dport_       0
Agent/CBR set sport_        0
Agent/CBR set dport_        0
```

```
Agent/TCPSink set sport_    0
Agent/TCPSink set dport_    0
Agent/TCP set sport_        0
Agent/TCP set dport_        0
Agent/TCP set packetSize_   1024
Queue/DropTail/PriQueue set Prefer_Routing_Protocols    1

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

# Initialize the SharedMedia interface with parameters to make
# it work like the 914MHz Lucent WaveLAN DSSS radio interface
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11; # 550m
Phy/WirelessPhy set RXThresh_ 3.652e-10; # 250m
Phy/WirelessPhy set Pt_ 0.281838
Phy/WirelessPhy set freq_ 2.4e+9
Phy/WirelessPhy set L_ 1.0

ErrorModel set enable_ 1
ErrorModel set markecn_ false
ErrorModel set bandwidth_ 2Mb


# ======================================================================
#source [lindex $argv 0]; # override the default settings
# ======================================================================

proc usage { argv0 }  {
    puts "Usage: $argv0"
    puts "\tmandatory arguments:"
    puts "\t\t\[-x MAXX\] \[-y MAXY\]"
    puts "\toptional arguments:"
    puts "\t\t\[-cp conn pattern\] \[-sc scenario\] \[-nn nodes\]"
    puts "\t\t\[-seed seed\] \[-stop sec\] \[-tr tracefile\]\n"
}

proc getopt {argc argv} {
    global opt
    lappend optlist cp nn seed sc stop tr x y ntcp hops gamma beta alpha qlen rate
    for {set i 0} {$i < $argc} {incr i} {
        set arg [lindex $argv $i]
        if {[string range $arg 0 0] != "-"} continue
        set name [string range $arg 1 end]
        set opt($name) [lindex $argv [expr $i+1]]
   # puts "opt($name) = $opt($log_file_prefix$num.loss$name)"
```

```
    }
}

proc create_udp_connection {src dst log_file_prefix start} {

   global ns_ node_ num opt udp_ udptrace_ sink_
   if ![info exists num] {
      set num 0
   }
   set udp_($num) [new Agent/UDP]
   $ns_ attach-agent $node_($src) $udp_($num)
   set cbr($num) [new Application/Traffic/CBR]
   $cbr($num) set packetSize_ 512
   $cbr($num) set interval_ 0.2

   #$cbr($num) set rate_ 300Kb # that is high CBR modification

   $cbr($num) attach-agent $udp_($num)
   #set null_($iiik) [new Agent/Null]
   set sink_($num) [new Agent/LossMonitor]
   $ns_ attach-agent $node_($dst) $sink_($num)
   $ns_ connect $udp_($num) $sink_($num)

   if { $log_file_prefix != "" } {
      $ns_ register_record $sink_($num) $log_file_prefix$num.tr
   }

   $ns_ at $start "$cbr($num) start"
   incr num

}

proc MyErrorProc {} {
    set errObj [new ErrorModel]
    $errObj FECstrength 0
    $errObj set rate_ 0.001
    $errObj set unit pkt
    return $errObj
}


Simulator instproc register_record {agent file} {
    $self instvar agent_list log_file_list
    lappend agent_list $agent
    lappend log_file_list [open $file w]
}

set total_thr_file [open "thr.log" w]

Simulator instproc record_udp {} {
```

```
    $self instvar agent_list log_file_list
    global total_thr_file opt sink_

    set nnn [llength $agent_list]
    set now [$self now]
    set interval $opt(stop)
    set bw_udp 0
    set bl_udp 0

    #set loss_ nlost_

    for {set i 0} {$i < $nnn} {incr i} {

   set btes [$sink_($i) set bytes_]
        set file [lindex $log_file_list $i]
        set bw [expr $btes/$interval*8.0/1024.0]
   set bw_udp [expr $bw_udp + $bw]
   $sink_($i) set bytes_ 0
        puts $file "$now $bw"

    }
    if { $interval == $opt(stop) } {
   if { $now == $opt(stop) } {
       puts $total_thr_file "$bw_udp"
   }
    } else {
   puts $total_thr_file "$now $bw_udp"
    }
    $self at [expr $now+$interval] "$self record_udp"

}



proc log-movement {} {
    global logtimer ns_ ns

    set ns $ns_
    source /tcl/timer.tcl
    Class LogTimer -superclass Timer
    LogTimer instproc timeout {} {
    global opt node_;
    for {set i 0} {$i < $opt(nn)} {incr i} {
        $node_($i) log-movement
    }
    $self sched 0.1
    }

    set logtimer [new LogTimer]
    $logtimer sched 0.1
```

```
}



proc finish {} {
   global ns_
   $ns_ flush-trace
   #close $nf

}

# ============================================================================
# Main Program
# ============================================================================

getopt $argc $argv

if { $opt(x) == 0 || $opt(y) == 0 } {
    usage $argv0
    exit 1
}

if {$opt(seed) > 0} {
    puts "Seeding Random number generator with $opt(seed)\n"
    ns-random $opt(seed)
}

#
# Initialize Global Variables
#
set ns_     [new Simulator]
set chan    [new $opt(chan)]
set prop    [new $opt(prop)]
set topo    [new Topography]
set tracefd [open $opt(tr) w]
#set nf     [open $opt(namfile) w]

#$ns_ namtrace-all $nf

$topo load_flatgrid $opt(x) $opt(y)

$prop topography $topo

#
# Create God
#
set god_ [create-god $opt(nn)]



#
# log the mobile nodes movements if desired
```

```
#
if { $opt(lm) == "on" } {
    log-movement
}

if { $opt(dist) == 60 } {
   Phy/WirelessPhy set CSThresh_ 1.65011e-9; # 130m
   Phy/WirelessPhy set RXThresh_ 7.74635e-9; # 60m
}

if { [string compare $opt(rp) "dsr"] == 0} {
    Agent/DSRAgent set beta_ $opt(beta)
    Agent/DSRAgent set gamma_ $opt(gamma)
    Mac/802_11 set basicRate_ $opt(rate)
    Mac/802_11 set dataRate_  $opt(rate)
    Phy/WirelessPhy set bandwidth_ $opt(rate)
    Phy/WirelessPhy set Rb_   $opt(rate)

    for {set i 0} {$i < $opt(nn) } {incr i} {
        dsr-create-mobile-node $i
    }
} elseif { [string compare $opt(rp) "dsdv"] == 0} {
    Mac/802_11 set basicRate_ $opt(rate)
    Mac/802_11 set dataRate_  $opt(rate)
    Phy/WirelessPhy set bandwidth_ $opt(rate)
    Phy/WirelessPhy set Rb_   $opt(rate)

    for {set i 0} {$i < $opt(nn) } {incr i} {
        dsdv-create-mobile-node $i
    }
}



#
# Source the Connection and Movement scripts
#
if { $opt(cp) == "" } {
    puts "*** NOTE: no connection pattern specified."
        set opt(cp) "none"
} else {
    Agent/TCP set increase_num_ $opt(alpha)
    puts "Loading connection pattern... $opt(cp)"
    source $opt(cp)
}

#
# Tell all the nodes when the simulation ends
#
for {set i 0} {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).000000001 "$node_($i) reset";
```

```
}
$ns_ at $opt(stop).00000001 "puts \"NS EXITING...\" ; finish ; $ns_ halt"



if { $opt(sc) == "" } {
    puts "*** NOTE: no scenario file specified."
        set opt(sc) "none"
} else {
    puts "Loading scenario file... $opt(sc)"
    source $opt(sc)
    puts "Load complete..."
}

$ns_ at 0.0 "$ns_ record_udp"
puts "Starting Simulation..."
$ns_ run
```

Chain scenario - Scripts

- Connections set up
  - TCP (tcp- chain.tcl)

```
set i 0
while { $i < $opt(ntcp) } {
    create_tcp_connection 0 $opt(hops) tcp 2.0
    incr i
}
```

  - UDP (udp-chain.tcl)

```
set i 0
while { $i < $opt(ntcp) } {
    create_udp_connection 0 $opt(hops) udp 2.0
    incr i
}
```

- Execution
  - TCP (sim-chain-tcp.sh)

```
#! /bin/sh
export PATH=$PATH:/ns2/bin:/ns2/tcl8.4.11/unix:/ns2/tk8.4.11/unix
mkdir result
mkdir result/chain_tcp

for B in 1 2
  do
  mkdir result/chain_tcp/DSR$B
  for G in 0 1
    do
    for N in  2 4 8
      do
      for H in 4 5 6 7 8 10 12 14 16 20
   do

   if \
       ns main-tcp.tcl -alpha 0.01 -beta $B -gamma $G -stop 120 \
       -cp tcp-chain.tcl -sc chain22.dst -nn 25 -ntcp $N -hops $H ; \
       then
       mkdir result/chain_tcp/DSR$B/g$G.tcp$N.hop$H
       mv *.tr result/chain_tcp/DSR$B/g$G.tcp$N.hop$H/
       mv *.log result/chain_tcp/DSR$B/g$G.tcp$N.hop$H/
       else
       mkdir result/chain_tcp/DSR$B/g$G.tcp$N.hop$H.fail
       mv *.tr result/chain_tcp/DSR$B/g$G.tcp$N.hop$H.fail/
       mv *.log result/chain_tcp/DSR$B/g$G.tcp$N.hop$H.fail/
   fi
      done
```

```
    done
  done
done
```

- UDP (sim-chain-udp.sh)

```
#! /bin/sh
export PATH=$PATH:/ns2/bin:/ns2/tcl8.4.11/unix:/ns2/tk8.4.11/unix
mkdir result
mkdir result/chain_udp/

for B in 1 2
  do
  mkdir result/chain_udp/DSR$B
  for G in 0 1
    do
    for N in 2 4 8
      do
      for H in 4 5 6 7 8 10 12 14 16 20
   do

   if \
       ns main-udp.tcl -beta $B -gamma $G -stop 120 \
       -cp udp-chain.tcl -sc chain22.dst -nn 25 -ntcp $N -hops $H ; \
       then
       mkdir result/chain_udp/DSR$B/g$G.udp$N.hop$H
       mv *.tr result/chain_udp/DSR$B/g$G.udp$N.hop$H/
       mv *.log result/chain_udp/DSR$B/g$G.udp$N.hop$H/
       else
       mkdir result/chain_udp/DSR$B/g$G.udp$N.hop$H.fail
       mv *.tr result/chain_udp/DSR$B/g$G.udp$N.hop$H.fail/
       mv *.log result/chain_udp/DSR$B/g$G.udp$N.hop$H.fail/
   fi
      done
    done
  done
done
```

- Analysis
  - TCP (analysis-chain-tcp.pl)

```
#! /usr/bin/perl
foreach $B ( 1 , 2 ) {
   print "beta: $B ";
   open MAC_DROPS, "> result/chain_tcp/DSR$B/mac_drops.log";
   open RT_ERROR, "> result/chain_tcp/DSR$B/route_errors.log";
   open RT_CHANGE, "> result/chain_tcp/DSR$B/route_changes.log";
   foreach $N ( 2 , 4 , 8  ) {
      print "N: $N ";
      print RT_CHANGE "$N";
```

```
print RT_ERROR "$N";
   foreach $G ( "g0", "g1" ) {
   print "\n gamma $G\n";
   foreach $H ( 4 , 5 , 6 , 7 , 8 , 10 , 12 , 14 , 16 , 20) {

      print " $H";
      open LOSS, "> result/chain_tcp/DSR$B/$G.tcp$N.hop$H/loss.log";
      open Overhead, "> result/chain_tcp/DSR$B/$G.tcp$N.hop$H/overhead.log";
      # variables for MAC errors
      open RT_CHG, "> route_change.$B.$G.tcp$N.hop$H";

      print MAC_DROPS "$N $G $H";
      # variables for MAC errors
      $col = 0;
      $dup = 0;
      $ret = 0;
      $bsy = 0;
      $cbk = 0;   # variable for Route Error
      $rerr = 0;
      # counting data packets (send, drop and recv...)
      $nsend = 0;
      $nrecv = 0;
      #counting the number of packets RTR  and MAC
      $nRTR_OH = 0;
      $nMAC_OH = 0;
      # route changes. . .
      $RChange = 0;
      $RChangeFlows = 0;
      open TRACE, "result/chain_tcp/DSR$B/$G.tcp$N.hop$H/out.tr";
      while (<TRACE>) {
         @ll = split(' ');
         # if packet drop ...
         if ( $ll[0] eq "d" && $ll[18] eq "MAC"){
            # collision
            elsif ( $ll[20] eq "COL" )
               {$col++;}
            # duplicate
            elsif ( $ll[20] eq "DUP" )
               {$dup++;}
            # retry exceeded count
            elsif ( $ll[20] eq "RET" )
               {$ret++;}
            # busy
            elsif ( $ll[20] eq "BSY" )
               {$bsy++;}
            # MAC callback
            elsif ( $ll[20] eq "CBK" )
               {$cbk++;}
         }
         # if route error then count
         if ($ll[0] eq "RERR" )
```

```
                    {$rerr +=1;}

                 if ( $ll[0] eq "s" && $ll[18] eq "AGT")
                    {$nsend +=1;}
                 if ( $ll[0] eq "r" && $ll[18] eq "AGT" )
                    {$nrecv +=1;}

                 # may be tcp, ack, dsr. Count only -It (type of packet) = dsr
                 if ( $ll[18] eq "RTR" && $ll[34] eq "DSR" )
                    { $nRTR_OH++;}
                 # ROUTING OVERHEAD IN MAC
                 if ( $ll[18] eq "MAC" && $ll[34] eq "DSR" )
                    { $nMAC_OH++;}

                 # calculation of route changes FOR ALL FLOWS!!
                 if ($ll[0] eq "RChange"){

                    if ($N == 2){
                       if ($ll[4] eq "0" && $ll[6] eq $H)
                          {$RChangeFlows++;}

                    }elsif ($N == 4){
                       if  ($ll[4] eq "0" && $ll[6] eq $H)
                          {$RChangeFlows++;}

                    }elsif ($N == 8){
                       if  ($ll[4] eq "0" && $ll[6] eq $H )
                          {$RChangeFlows++;}
                    }
                 }
              }
              close TRACE;
              print LOSS "$nsend $nrecv";
              close LOSS;
              close RT_CHG; # log for the route changes of one flow over the time...
              print RT_ERROR " $rerr";
              $oh1 = $nRTR_OH/120;
              $oh2 = $nMAC_OH/120;
              print Overhead "RTR: $nRTR_OH MAC: $nMAC_OH => $oh1 $oh2\n";
              close Overhead;

              print MAC_DROPS " end: $end col: $col dup: $dup ret: $ret err: $err; sta: $sta
bsy: $bsy dst: $dst nrte: $nrte loop: $loop ttl: $ttl ifq: $ifq tout: $tout cbk: $cbk sal:
$sal arp: $arp fil: $fil out: $out\n";
              print RT_CHANGE " $RChangeFlows";
           }
        print "\n";
        print RT_ERROR "\n";
        print RT_CHANGE "\n";
        }
     }
```

```
   close RT_ERROR;
   close MAC_DROPS;
   close RT_CHANGE;
}
```

- UDP (analysis-chain-udp.pl)

```
#! /usr/bin/perl
foreach $B ( 1 , 2 ) {
   print "beta: $B ";
   open MAC_DROPS, "> result/chain_udp/DSR$B/mac_drops.log";
   open RT_ERROR, "> result/chain_udp/DSR$B/route_errors.log";
   open RT_CHANGE, "> result/chain_udp/DSR$B/route_changes.log";
   foreach $N ( 2 , 4 , 8  ) {
      print "N: $N ";
      print RT_CHANGE "$N";
      print RT_ERROR "$N";
         foreach $G ( "g0", "g1" ) {
         print "\n gamma $G\n";
         foreach $H ( 4 , 5 , 6 , 7 , 8 , 10 , 12 , 14 , 16 , 20) {
            print " $H";
            open LOSS, "> result/chain_udp/DSR$B/$G.udp$N.hop$H/loss.log";
            open Overhead, "> result/chain_udp/DSR$B/$G.udp$N.hop$H/overhead.log";
            # variables for MAC errors
            open RT_CHG, "> route_change.$B.$G.udp$N.hop$H";

            print MAC_DROPS "$N $G $H";
            # variables for MAC errors
            $col = 0;
            $dup = 0;
            $ret = 0;
            $bsy = 0;
            $cbk = 0;      # variable for Route Error
            $rerr = 0;
            # counting data packets (send, drop and recv...)
            $nsend = 0;
            $nrecv = 0;
            #counting the number of packets RTR  and MAC
            $nRTR_OH = 0;
            $nMAC_OH = 0;
            # route changes. . .
            $RChange = 0;
            $RChangeFlows = 0;
            open TRACE, "result/chain_udp/DSR$B/$G.udp$N.hop$H/out.tr";
            while (<TRACE>) {
               @ll = split(' ');
               # if packet drop ...
               if ( $ll[0] eq "d" && $ll[18] eq "MAC"){
                  # end of simulation
                  if ( $ll[20] eq "END" )
```

```
            {$end++;}
         # collision
         elsif ( $ll[20] eq "COL" )
            {$col++;}
         # duplicate
         elsif ( $ll[20] eq "DUP" )
            {$dup++;}
         # retry exceeded count
         elsif ( $ll[20] eq "RET" )
            {$ret++;}
         # busy
         elsif ( $ll[20] eq "BSY" )
            {$bsy++;}
         # MAC callback
         elsif ( $ll[20] eq "CBK" )
            {$cbk++;}
      }
      # if route error then count
      if ($ll[0] eq "RERR" )
         {$rerr +=1;}
      if ( $ll[0] eq "s" && $ll[18] eq "AGT")
         {$nsend +=1;}
      if ( $ll[0] eq "r" && $ll[18] eq "AGT" )
         {$nrecv +=1;}
      # may be tcp, ack, dsr. Count only -It (type of packet) = dsr
      if ( $ll[18] eq "RTR" && $ll[34] eq "DSR" )
         { $nRTR_OH++;}
      # ROUTING OVERHEAD IN MAC
      if ( $ll[18] eq "MAC" && $ll[34] eq "DSR" )
         { $nMAC_OH++;}
      # calculation of route changes FOR ALL FLOWS!!
      if ($ll[0] eq "RChange"){
         if ($N == 2){
            if ($ll[4] eq "0" && $ll[6] eq $H)
               {$RChangeFlows++;}
         }elsif ($N == 4){
            if  ($ll[4] eq "0" && $ll[6] eq $H)
               {$RChangeFlows++;}
         }elsif ($N == 8){
            if  ($ll[4] eq "0" && $ll[6] eq $H )
               {$RChangeFlows++;}
         }
      }
   }
   close TRACE;
   print LOSS "$nsend $nrecv";
   close LOSS;
   close RT_CHG; # log for the route changes of one flow over the time...
   print RT_ERROR " $rerr";
   $oh1 = $nRTR_OH/120;
   $oh2 = $nMAC_OH/120;
```

```
               print Overhead "RTR: $nRTR_OH MAC: $nMAC_OH => $oh1 $oh2\n";
               close Overhead;

               print MAC_DROPS " end: $end col: $col dup: $dup ret: $ret err: $err; sta: $sta
bsy: $bsy dst: $dst nrte: $nrte loop: $loop ttl: $ttl ifq: $ifq tout: $tout cbk: $cbk sal:
$sal arp: $arp fil: $fil out: $out\n";
               print RT_CHANGE " $RChangeFlows";
            }
         print "\n";
         print RT_ERROR "\n";
         print RT_CHANGE "\n";
         }
      }
      close RT_ERROR;
      close MAC_DROPS;
      close RT_CHANGE;
   }
```

Grid 7x7

- Connections set up
  - TCP (tcp-grid7x7.tcl)

```
# parallel 1//1

if { $opt(ntcp) == 1 } {
    create_tcp_connection 21 27 tcp 3
    create_tcp_connection 21 27 tcp 3
}

# cross 1x1

if { $opt(ntcp) == 2 } {
    create_tcp_connection  3 45 tcp 3
    create_tcp_connection 21 27 tcp 3
}


# cross 2x2

if { $opt(ntcp) == 4 } {
    create_tcp_connection 1 43 tcp 3
    create_tcp_connection 5 47 tcp 3

    create_tcp_connection 7 13 tcp 3
    create_tcp_connection 35 41 tcp 3
}


# cross 3x3

if { $opt(ntcp)==6 } {
    create_tcp_connection  1 43 tcp 3
    create_tcp_connection  3 45 tcp 3
    create_tcp_connection  5 47 tcp 3

    create_tcp_connection  7 13 tcp 3
    create_tcp_connection 21 27 tcp 3
    create_tcp_connection 35 41 tcp 3
}

# cross 4x4

if { $opt(ntcp) == 8 } {
    create_tcp_connection 0 42 tcp 5
    create_tcp_connection 2 44 tcp 5
    create_tcp_connection 4 46 tcp 5
    create_tcp_connection 6 48 tcp 5
```

```
    create_tcp_connection 0 6 tcp 5
    create_tcp_connection 14 20 tcp 5
    create_tcp_connection 28 34 tcp 5
    create_tcp_connection 42 48 tcp 5
}

# cross 7x7

if { $opt(ntcp) == 14 } {
    create_tcp_connection 0 42 tcp 5
    create_tcp_connection 1 43 tcp 5
    create_tcp_connection 2 44 tcp 5
    create_tcp_connection 3 45 tcp 5
    create_tcp_connection 4 46 tcp 5
    create_tcp_connection 5 47 tcp 5
    create_tcp_connection 6 48 tcp 5

    create_tcp_connection 0 6 tcp 5
    create_tcp_connection 7 13 tcp 5
    create_tcp_connection 14 20 tcp 5
    create_tcp_connection 21 27 tcp 5
    create_tcp_connection 28 34 tcp 5
    create_tcp_connection 35 41 tcp 5
    create_tcp_connection 42 48 tcp 5
}

# random 10 we did simulate this but did not show in results

if { $opt(ntcp) == 10 } {
    create_tcp_connection 22 10 tcp 5
    create_tcp_connection 23 12 tcp 5
    create_tcp_connection 14 21 tcp 5
    create_tcp_connection 19 32 tcp 5
    create_tcp_connection 15 1 tcp 5

    create_tcp_connection 21 11 tcp 5
    create_tcp_connection 5 33 tcp 5
    create_tcp_connection 3 30 tcp 5
    create_tcp_connection 15 1 tcp 5
    create_tcp_connection 27 20 tcp 5
}
```

- UDP (udp-grid7x7.tcl)

```
# parallel 1//1

if { $opt(ntcp) == 1 } {
    create_udp_connection 21 27 udp 3
    create_udp_connection 21 27 udp 3
}
```

```
# cross 1x1

if { $opt(ntcp) == 2 } {
    create_udp_connection  3 45 udp 3
    create_udp_connection 21 27 udp 3
}


# cross 2x2

if { $opt(ntcp) == 4 } {
    create_udp_connection 1 43 udp 3
    create_udp_connection 5 47 udp 3

    create_udp_connection 7 13 udp 3
    create_udp_connection 35 41 udp 3
}


# cross 3x3

if { $opt(ntcp)==6 } {
    create_udp_connection  1 43 udp 3
    create_udp_connection  3 45 udp 3
    create_udp_connection  5 47 udp 3

    create_udp_connection  7 13 udp 3
    create_udp_connection 21 27 udp 3
    create_udp_connection 35 41 udp 3
}

# cross 4x4

if { $opt(ntcp) == 8 } {
    create_udp_connection 0 42 udp 5
    create_udp_connection 2 44 udp 5
    create_udp_connection 4 46 udp 5
    create_udp_connection 6 48 udp 5

    create_udp_connection 0 6 udp 5
    create_udp_connection 14 20 udp 5
    create_udp_connection 28 34 udp 5
    create_udp_connection 42 48 udp 5
}

# cross 7x7

if { $opt(ntcp) == 14 } {
    create_udp_connection 0 42 udp 5
    create_udp_connection 1 43 udp 5
    create_udp_connection 2 44 udp 5
```

```
    create_udp_connection 3 45 udp 5
    create_udp_connection 4 46 udp 5
    create_udp_connection 5 47 udp 5
    create_udp_connection 6 48 udp 5

    create_udp_connection 0 6 udp 5
    create_udp_connection 7 13 udp 5
    create_udp_connection 14 20 udp 5
    create_udp_connection 21 27 udp 5
    create_udp_connection 28 34 udp 5
    create_udp_connection 35 41 udp 5
    create_udp_connection 42 48 udp 5
}

# random 10

if { $opt(ntcp) == 10 } {
    create_udp_connection 22 10 udp 5
    create_udp_connection 23 12 udp 5
    create_udp_connection 14 21 udp 5
    create_udp_connection 19 32 udp 5
    create_udp_connection 15 1 udp 5

    create_udp_connection 21 11 udp 5
    create_udp_connection 5 33 udp 5
    create_udp_connection 3 30 udp 5
    create_udp_connection 15 1 udp 5
    create_udp_connection 27 20 udp 5
}
```

- Execution
  - TCP (sim-grid-tcp.sh)

```
#! /bin/sh

export PATH=$PATH:/ns2/bin:/ns2/tcl8.4.11/unix:/ns2/tk8.4.11/unix

mkdir result
mkdir result/grid7x7
mkdir result/grid7x7/DSR1
mkdir result/grid7x7/DSR2

for B in 1 2
do
   for G in 0 1
   do
      for N in 1 2 4 6 8 14
      do
      if \
         ns main-tcp.tcl -beta $B -gamma $G -nn 50 -ntcp $N -rate 2e6 \
```

```
         -cp tcp-grid7x7.tcl -sc grid7x7.dst -stop 120; \
         then
         mkdir result/grid7x7/DSR$B/g$G.tcp$N
         mv *.tr result/grid7x7/DSR$B/g$G.tcp$N/
         mv *.loss result/grid7x7/DSR$B/g$G.tcp$N/
         mv *.log result/grid7x7/DSR$B/g$G.tcp$N/
         else
         mkdir result/grid7x7/DSR$B/g$G.tcp$N.fail
         mv *.tr result/grid7x7/DSR$B/g$G.tcp$N.fail/
         mv *.log result/grid7x7/DSR$B/g$G.tcp$N.fail/
         mv *.loss result/grid7x7/DSR$B/g$G.tcp$N.fail/
      fi

      done
   done
done
```

- UDP (sim-grid-udp.sh)

```
#! /bin/sh

export PATH=$PATH:/ns2/bin:/ns2/tcl8.4.11/unix:/ns2/tk8.4.11/unix

mkdir result
mkdir result/grid7x7
mkdir result/grid7x7/DSR1
mkdir result/grid7x7/DSR2

for B in 1 2
do
   for G in 0 1
   do
      for N in 1 2 4 6 8 14
      do
      if \
         ns main-udp.tcl -beta $B -gamma $G -nn 50 -ntcp $N -rate 2e6 \
         -cp udp-grid7x7.tcl -sc grid7x7.dst -stop 120; \
         then
         mkdir result/grid7x7/DSR$B/g$G.udp$N
         mv *.tr result/grid7x7/DSR$B/g$G.udp$N/
         mv *.loss result/grid7x7/DSR$B/g$G.udp$N/
         mv *.log result/grid7x7/DSR$B/g$G.udp$N/
         else
         mkdir result/grid7x7/DSR$B/g$G.udp$N.fail
         mv *.tr result/grid7x7/DSR$B/g$G.udp$N.fail/
         mv *.log result/grid7x7/DSR$B/g$G.udp$N.fail/
         mv *.loss result/grid7x7/DSR$B/g$G.udp$N.fail/
      fi

      done
```

```
   done
done
```

- Analysis
  - TCP (analysis-grid-tcp.pl)

```
#! /usr/bin/perl

foreach $B ( 1, 2 ) {
   print "beta: $B ";
   open MAC_DROPS, "> result/grid7x7tcp/DSR$B/mac_drops.log";
   open RT_ERROR, "> result/grid7x7tcp/DSR$B/route_errors.log";
   open RT_CHANGE, "> result/grid7x7tcp/DSR$B/route_changes.log";
   foreach $N ( 1, 2, 4, 6, 8 ,14 ,10  ) {
      print "N: $N ";
      print RT_CHANGE "$N";
      print RT_ERROR "$N";
            foreach $G ( "g0", "g1" ) {
         print " gamma $G\n";
         open LOSS, "> result/grid7x7tcp/DSR$B/$G.tcp$N/loss.log";
         open Overhead, "> result/grid7x7tcp/DSR$B/$G.tcp$N/overhead.log";
         # variables for MAC errors

         print MAC_DROPS "$N $G";
         # variables for MAC errors
         $col = 0;
         $dup = 0;
         $ret = 0;
         $bsy = 0;
         $cbk = 0;
         # variable for Route Error
         $rerr = 0;
         # counting data packets (send, drop and recv...)
         $nsend = 0;
         $nrecv = 0;
         #counting the number of packets RTR  and MAC
         $nRTR_OH = 0;
         $nMAC_OH = 0;
         # route changes. . .
         $RChange = 0;
         $RChangeFlows = 0;
            open TRACE, "result/grid7x7tcp/DSR$B/$G.tcp$N/out.tr";
         while (<TRACE>) {
            @ll = split(' ');
            # if packet drop ...
            if ( $ll[0] eq "d" && $ll[18] eq "MAC"){
               # collision
               elsif ( $ll[20] eq "COL" )
                  {$col++;}
               # duplicate
```

```
   elsif ( $ll[20] eq "DUP" )
      {$dup++;}
   # retry exceeded count
   elsif ( $ll[20] eq "RET" )
   # busy
   elsif ( $ll[20] eq "BSY" )
      {$bsy++;}
   # MAC callback
   elsif ( $ll[20] eq "CBK" )
      {$cbk++;}
}
# if route error then count
if ($ll[0] eq "RERR" )
   {$rerr +=1;}

if ( $ll[0] eq "s" && $ll[18] eq "AGT")
   {$nsend +=1;}
if ( $ll[0] eq "r" && $ll[18] eq "AGT" )
   {$nrecv +=1;}

# may be tcp, ack, dsr. Count only -It (type of packet) = dsr
if ( $ll[18] eq "RTR" && $ll[34] eq "DSR" )
   { $nRTR_OH++;}
# ROUTING OVERHEAD IN MAC
if ( $ll[18] eq "MAC" && $ll[34] eq "DSR" )
   { $nMAC_OH++;}

# calculation of route changes FOR ALL FLOWS!!
if ($ll[0] eq "RChange"){
   if ($N == 1) {
      if  ($ll[4] eq "21" && $ll[6] eq "27")
         {$RChangeFlows++;}

   }elsif ($N == 2){
      if  ( ($ll[4] eq "3" && $ll[6] eq "45")
      || ($ll[4] eq "21" && $ll[6] eq "27") )
         {$RChangeFlows++;}

   }elsif ($N == 4){
      if  ( ($ll[4] eq "1" && $ll[6] eq "43")
      || ($ll[4] eq "5" && $ll[6] eq "47")
      || ($ll[4] eq "7" && $ll[6] eq "13")
      || ($ll[4] eq "35" && $ll[6] eq "41") )
         {$RChangeFlows++;}

   }elsif ($N == 6){
      if  ( ($ll[4] eq "1" && $ll[6] eq "43")
      || ($ll[4] eq "3" && $ll[6] eq "45")
      || ($ll[4] eq "5" && $ll[6] eq "47")
      || ($ll[4] eq "7" && $ll[6] eq "13")
      || ($ll[4] eq "21" && $ll[6] eq "27")
```

```
         || ($ll[4] eq "35" && $ll[6] eq "41") )
            {$RChangeFlows++;}

      }elsif ($N == 8){
         if  ( ($ll[4] eq "0" && $ll[6] eq "42")
         || ($ll[4] eq "2" && $ll[6] eq "44")
         || ($ll[4] eq "4" && $ll[6] eq "46")
         || ($ll[4] eq "6" && $ll[6] eq "48")
         || ($ll[4] eq "0" && $ll[6] eq "6")
         || ($ll[4] eq "14" && $ll[6] eq "20")
         || ($ll[4] eq "28" && $ll[6] eq "34")
         || ($ll[4] eq "42" && $ll[6] eq "48") )
            {$RChangeFlows++;}

      }elsif ($N == 14){
         if  ( ($ll[4] eq "0" && $ll[6] eq "42")
         || ($ll[4] eq "1" && $ll[6] eq "43")
         || ($ll[4] eq "2" && $ll[6] eq "44")
         || ($ll[4] eq "3" && $ll[6] eq "45")
         || ($ll[4] eq "4" && $ll[6] eq "46")
         || ($ll[4] eq "5" && $ll[6] eq "47")
         || ($ll[4] eq "6" && $ll[6] eq "48")

         || ($ll[4] eq "0" && $ll[6] eq "6")
         || ($ll[4] eq "7" && $ll[6] eq "13")
         || ($ll[4] eq "14" && $ll[6] eq "20")
         || ($ll[4] eq "21" && $ll[6] eq "27")
         || ($ll[4] eq "28" && $ll[6] eq "34")
         || ($ll[4] eq "35" && $ll[6] eq "41")
         || ($ll[4] eq "42" && $ll[6] eq "48") )
            {$RChangeFlows++;}

      }elsif ($N == 10){
         if  ( ($ll[4] eq "22" && $ll[6] eq "10")
         || ($ll[4] eq "23" && $ll[6] eq "12")
         || ($ll[4] eq "14" && $ll[6] eq "21")
         || ($ll[4] eq "19" && $ll[6] eq "32")
         || ($ll[4] eq "15" && $ll[6] eq "1")

         || ($ll[4] eq "21" && $ll[6] eq "11")
         || ($ll[4] eq "5" && $ll[6] eq "33")
         || ($ll[4] eq "3" && $ll[6] eq "30")
         || ($ll[4] eq "15" && $ll[6] eq "1")
         || ($ll[4] eq "27" && $ll[6] eq "20") )
            {$RChangeFlows++;}
      }
   }

}
close TRACE;
print LOSS "$nsend $nrecv";
```

```
            close LOSS;

            print RT_ERROR " $rerr";
            $oh1 = $nRTR_OH/120;
            $oh2 = $nMAC_OH/120;
            print Overhead "RTR: $nRTR_OH MAC: $nMAC_OH => $oh1 $oh2\n";
            close Overhead;

            print MAC_DROPS " end: $end col: $col dup: $dup ret: $ret err: $err; sta: $sta
bsy: $bsy dst: $dst nrte: $nrte loop: $loop ttl: $ttl ifq: $ifq tout: $tout cbk: $cbk sal:
$sal arp: $arp fil: $fil out: $out\n";
            print RT_CHANGE " $RChangeFlows";

         }
         print "\n";
         print RT_ERROR "\n";
         print RT_CHANGE "\n";
      }
      close RT_ERROR;
      close MAC_DROPS;
      close RT_CHANGE;
   }
```

- UDP (analysis-grid-udp.pl)

```
   #! /usr/bin/perl
   foreach $B ( 1, 2 ) {
      print "beta: $B ";
      open MAC_DROPS, "> result/grid7x7udp/DSR$B/mac_drops.log";
      open RT_ERROR, "> result/grid7x7udp/DSR$B/route_errors.log";
      open RT_CHANGE, "> result/grid7x7udp/DSR$B/route_changes.log";
      foreach $N ( 1, 2, 4, 6, 8 ,14 ,10  ) {
         print "N: $N ";
         print RT_CHANGE "$N";
         print RT_ERROR "$N";
               foreach $G ( "g0", "g1" ) {
            print " gamma $G\n";

            open LOSS, "> result/grid7x7udp/DSR$B/$G.udp$N/loss.log";
            open Overhead, "> result/grid7x7udp/DSR$B/$G.udp$N/overhead.log";
            # variables for MAC errors

            print MAC_DROPS "$N $G";
            # variables for MAC errors
            $col = 0;
            $dup = 0;
            $ret = 0;
            $bsy = 0;
            $cbk = 0;
            # variable for Route Error
```

```
$rerr = 0;
# counting data packets (send, drop and recv...)
$nsend = 0;
$nrecv = 0;
#counting the number of packets RTR  and MAC
$nRTR_OH = 0;
$nMAC_OH = 0;
# route changes. . .
$RChange = 0;
$RChangeFlows = 0;
   open TRACE, "result/grid7x7udp/DSR$B/$G.udp$N/out.tr";
open CHANGES_ROUTE, "> changesRoute/route_change.$B.$G.udp$N";
while (<TRACE>) {
   @ll = split(' ');
   # if packet drop ...
   if ( $ll[0] eq "d" && $ll[18] eq "MAC"){
      # collision
      elsif ( $ll[20] eq "COL" )
         {$col++;}
      # duplicate
      elsif ( $ll[20] eq "DUP" )
         {$dup++;}
      # retry exceeded count
      elsif ( $ll[20] eq "RET" )
         {$ret++;}
      # busy
      elsif ( $ll[20] eq "BSY" )
         {$bsy++;}
      # MAC callback
      elsif ( $ll[20] eq "CBK" )
         {$cbk++;}
   }
   # if route error then count
   if ($ll[0] eq "RERR" )
      {$rerr +=1;}

   if ( $ll[0] eq "s" && $ll[18] eq "AGT")
      {$nsend +=1;}
   if ( $ll[0] eq "r" && $ll[18] eq "AGT" )
      {$nrecv +=1;}
   # may be udp, ack, dsr. Count only -It (type of packet) = dsr
   if ( $ll[18] eq "RTR" && $ll[34] eq "DSR" )
      { $nRTR_OH++;}
   # ROUTING OVERHEAD IN MAC
   if ( $ll[18] eq "MAC" && $ll[34] eq "DSR" )
      { $nMAC_OH++;}
   # calculation of route changes FOR ALL FLOWS!!
   if ($ll[0] eq "RChange"){
      if ($N == 1) {
         if  ($ll[4] eq "21" && $ll[6] eq "27")
            {$RChangeFlows++;}
```

```
   # if the source is 21 and dest 27
   if ($ll[4] eq "21" && $ll[6] eq "27") {
      # time, routeHops, godHops
      print CHANGES_ROUTE "$ll[1] $ll[3] $ll[8]\n";
   }


}elsif ($N == 2){
   if  ( ($ll[4] eq "3" && $ll[6] eq "45")
   || ($ll[4] eq "21" && $ll[6] eq "27") )
      {$RChangeFlows++;}

   # if the source is 21 and dest 27
   if ($ll[4] eq "21" && $ll[6] eq "27") {
      # time, routeHops, godHops
      print CHANGES_ROUTE "$ll[1] $ll[3] $ll[8]\n";
   }

}elsif ($N == 4){
   if  ( ($ll[4] eq "1" && $ll[6] eq "43")
   || ($ll[4] eq "5" && $ll[6] eq "47")
   || ($ll[4] eq "7" && $ll[6] eq "13")
   || ($ll[4] eq "35" && $ll[6] eq "41") )
      {$RChangeFlows++;}

}elsif ($N == 6){
   if  ( ($ll[4] eq "1" && $ll[6] eq "43")
   || ($ll[4] eq "3" && $ll[6] eq "45")
   || ($ll[4] eq "5" && $ll[6] eq "47")
   || ($ll[4] eq "7" && $ll[6] eq "13")
   || ($ll[4] eq "21" && $ll[6] eq "27")
   || ($ll[4] eq "35" && $ll[6] eq "41") )
      {$RChangeFlows++;}

   # if the source is 21 and dest 27
   if ($ll[4] eq "21" && $ll[6] eq "27") {
      # time, routeHops, godHops
      print CHANGES_ROUTE "$ll[1] $ll[3] $ll[8]\n";
   }


}elsif ($N == 8){
   if  ( ($ll[4] eq "0" && $ll[6] eq "42")
   || ($ll[4] eq "2" && $ll[6] eq "44")
   || ($ll[4] eq "4" && $ll[6] eq "46")
   || ($ll[4] eq "6" && $ll[6] eq "48")
   || ($ll[4] eq "0" && $ll[6] eq "6")
   || ($ll[4] eq "14" && $ll[6] eq "20")
   || ($ll[4] eq "28" && $ll[6] eq "34")
   || ($ll[4] eq "42" && $ll[6] eq "48") )
```

```
                {$RChangeFlows++;}

          }elsif ($N == 14){
             if  ( ($ll[4] eq "0" && $ll[6] eq "42")
             || ($ll[4] eq "1" && $ll[6] eq "43")
             || ($ll[4] eq "2" && $ll[6] eq "44")
             || ($ll[4] eq "3" && $ll[6] eq "45")
             || ($ll[4] eq "4" && $ll[6] eq "46")
             || ($ll[4] eq "5" && $ll[6] eq "47")
             || ($ll[4] eq "6" && $ll[6] eq "48")

             || ($ll[4] eq "0" && $ll[6] eq "6")
             || ($ll[4] eq "7" && $ll[6] eq "13")
             || ($ll[4] eq "14" && $ll[6] eq "20")
             || ($ll[4] eq "21" && $ll[6] eq "27")
             || ($ll[4] eq "28" && $ll[6] eq "34")
             || ($ll[4] eq "35" && $ll[6] eq "41")
             || ($ll[4] eq "42" && $ll[6] eq "48") )
                {$RChangeFlows++;}

          }elsif ($N == 10){
             if  ( ($ll[4] eq "22" && $ll[6] eq "10")
             || ($ll[4] eq "23" && $ll[6] eq "12")
             || ($ll[4] eq "14" && $ll[6] eq "21")
             || ($ll[4] eq "19" && $ll[6] eq "32")
             || ($ll[4] eq "15" && $ll[6] eq "1")

             || ($ll[4] eq "21" && $ll[6] eq "11")
             || ($ll[4] eq "5" && $ll[6] eq "33")
             || ($ll[4] eq "3" && $ll[6] eq "30")
             || ($ll[4] eq "15" && $ll[6] eq "1")
             || ($ll[4] eq "27" && $ll[6] eq "20") )
                {$RChangeFlows++;}
          }
       }
    }
    close TRACE;
    close CHANGES_ROUTE;
    print LOSS "$nsend $nrecv";
    close LOSS;

    print RT_ERROR " $rerr";
    $oh1 = $nRTR_OH/120;
    $oh2 = $nMAC_OH/120;
    print Overhead "RTR: $nRTR_OH MAC: $nMAC_OH => $oh1 $oh2\n";
    close Overhead;

    print MAC_DROPS "col: $col dup: $dup ret: $ret ;bsy: $bsy cbk: $cbk\n";
    print RT_CHANGE " $RChangeFlows";

}
```

```
   print "\n";
   print RT_ERROR "\n";
   print RT_CHANGE "\n";
}
close RT_ERROR;
close MAC_DROPS;
close RT_CHANGE;
    }
```

- Calculation of changes of the length of the route
  - TCP (route-changes-grid-tcp.pl)

```
#! /usr/bin/perl

# print "Loss rate calculation takes a long time. Now, running!\n";
print "calculating route changes for node 21 to 27,=> n1,n2 and n6\n";


foreach $N ( 1 , 2 , 6 ) {
   foreach $B ( 1, 2 ) {
      print "beta: $B";

         foreach $G ( "g0", "g1" ) {
         open RT_CHG, "> route_change.$B.$G.tcp$N";
         print " gamma $G\n";

            open TRACE, "result/grid7x7tcp/DSR$B/$G.tcp$N/out.tr";
         while (<TRACE>) {
            @ll = split(' ');
            if ( $ll[0] eq "RChange" ){
               # if the source is 21 and dest 27
               if ($ll[4] eq "21" && $ll[6] eq "27") {
                  # time, routeHops, godHops
                  print RT_CHG "$ll[1] $ll[3] $ll[8]\n";
               }
            }
         }
         close TRACE;
         close RT_CHG;
      }
   }
}
```

  - UDP (route-changes-grid-udp.pl)

```
#! /usr/bin/perl

# print "Loss rate calculation takes a long time. Now, running!\n";
print "calculating route changes for node 21 to 27,=> n1,n2 and n6\n";


foreach $N ( 1 , 2 , 6 ) {
   foreach $B ( 1, 2 ) {
      print "beta: $B";

         foreach $G ( "g0", "g1" ) {
         open RT_CHG, "> route_change.$B.$G.udp$N";
         print " gamma $G\n";

            open TRACE, "result/grid7x7udp/DSR$B/$G.udp$N/out.tr";
```

```
        while (<TRACE>) {
           @ll = split(' ');
           if ( $ll[0] eq "RChange" ){
              # if the source is 21 and dest 27
              if ($ll[4] eq "21" && $ll[6] eq "27") {
                 # time, routeHops, godHops
                 print RT_CHG "$ll[1] $ll[3] $ll[8]\n";
              }
           }
        }
        close TRACE;
        close RT_CHG;
     }
  }
}
```

Random way point

- Connections set up
    - TCP (tcp-randway.tcl)

```
if { $opt(ntcp) == 8 } {
    create_tcp_connection 90 91 tcp 5
    create_tcp_connection 138 92 tcp 5
    create_tcp_connection 95 96 tcp 5
    create_tcp_connection 96 97 tcp 5
    create_tcp_connection 16 98 tcp 5
    create_tcp_connection 105 106 tcp 5
    create_tcp_connection 112 54 tcp 5
    create_tcp_connection 113 114 tcp 5
}


if { $opt(ntcp) == 16 } {

    create_tcp_connection 0 12 tcp 5
    create_tcp_connection 26 38 tcp 5
    create_tcp_connection 52 64 tcp 5
    create_tcp_connection 78 90 tcp 5
    create_tcp_connection 104 116 tcp 5
    create_tcp_connection 130 142 tcp 5
    create_tcp_connection 111 68 tcp 5
    create_tcp_connection 141 18 tcp 5
    create_tcp_connection 17 19 tcp 5
    create_tcp_connection 0 15 tcp 5
    create_tcp_connection 2 18 tcp 5
    create_tcp_connection 4 90 tcp 5
    create_tcp_connection 6 62 tcp 5
    create_tcp_connection 8 104 tcp 5
    create_tcp_connection 10 42 tcp 5
    create_tcp_connection 12 97 tcp 5
}


if { $opt(ntcp) == 32 } {
    create_tcp_connection 141 18 tcp 5
    create_tcp_connection 17 19 tcp 5
    create_tcp_connection 20 21 tcp 5
    create_tcp_connection 120 22 tcp 5
    create_tcp_connection 24 25 tcp 5
    create_tcp_connection 28 30 tcp 5
    create_tcp_connection 33 34 tcp 5
    create_tcp_connection 31 35 tcp 5
    create_tcp_connection 135 36 tcp 5
    create_tcp_connection 148 37 tcp 5
    create_tcp_connection 23 38 tcp 5
    create_tcp_connection 147 39 tcp 5
```

```
    create_tcp_connection 43 45 tcp 5
    create_tcp_connection 44 49 tcp 5
    create_tcp_connection 50 46 tcp 5
    create_tcp_connection 47 48 tcp 5
    create_tcp_connection 60 61 tcp 5
    create_tcp_connection 71 72 tcp 5
    create_tcp_connection 67 73 tcp 5
    create_tcp_connection 66 74 tcp 5
    create_tcp_connection 77 78 tcp 5
    create_tcp_connection 80 82 tcp 5
    create_tcp_connection 52 84 tcp 5
    create_tcp_connection 83 85 tcp 5
    create_tcp_connection 88 89 tcp 5
    create_tcp_connection 90 91 tcp 5
    create_tcp_connection 138 92 tcp 5
    create_tcp_connection 95 96 tcp 5
    create_tcp_connection 96 97 tcp 5
    create_tcp_connection 16 98 tcp 5
    create_tcp_connection 105 106 tcp 5
    create_tcp_connection 112 54 tcp 5
}


if { $opt(ntcp) == 50 } {
    create_tcp_connection 1 2 tcp 5
    create_tcp_connection 3 5 tcp 5
    create_tcp_connection 4 6 tcp 5
    create_tcp_connection 8 9 tcp 5
    create_tcp_connection 5 10 tcp 5
    create_tcp_connection 9 11 tcp 5
    create_tcp_connection 144 12 tcp 5
    create_tcp_connection 16 17 tcp 5
    create_tcp_connection 146 142 tcp 5
    create_tcp_connection 141 18 tcp 5
    create_tcp_connection 17 19 tcp 5
    create_tcp_connection 20 21 tcp 5
    create_tcp_connection 120 22 tcp 5
    create_tcp_connection 24 25 tcp 5
    create_tcp_connection 28 30 tcp 5
    create_tcp_connection 33 34 tcp 5
    create_tcp_connection 31 35 tcp 5
    create_tcp_connection 135 36 tcp 5
    create_tcp_connection 148 37 tcp 5
    create_tcp_connection 23 38 tcp 5
    create_tcp_connection 147 39 tcp 5
    create_tcp_connection 43 45 tcp 5
    create_tcp_connection 44 49 tcp 5
    create_tcp_connection 50 46 tcp 5
    create_tcp_connection 47 48 tcp 5
    create_tcp_connection 60 61 tcp 5
    create_tcp_connection 71 72 tcp 5
```

```
    create_tcp_connection 67 73 tcp 5
    create_tcp_connection 66 74 tcp 5
    create_tcp_connection 77 78 tcp 5
    create_tcp_connection 80 82 tcp 5
    create_tcp_connection 52 84 tcp 5
    create_tcp_connection 83 85 tcp 5
    create_tcp_connection 88 89 tcp 5
    create_tcp_connection 90 91 tcp 5
    create_tcp_connection 138 92 tcp 5
    create_tcp_connection 95 96 tcp 5
    create_tcp_connection 96 97 tcp 5
    create_tcp_connection 16 98 tcp 5
    create_tcp_connection 105 106 tcp 5
    create_tcp_connection 112 54 tcp 5
    create_tcp_connection 113 114 tcp 5
    create_tcp_connection 55 115 tcp 5
    create_tcp_connection 116 117 tcp 5
    create_tcp_connection 122 124 tcp 5
    create_tcp_connection 56 125 tcp 5
    create_tcp_connection 124 126 tcp 5
    create_tcp_connection 125 57 tcp 5
    create_tcp_connection 131 132 tcp 5
    create_tcp_connection 133 134 tcp 5
}
```

- UDP (udp-randway.tcl)

```
if { $opt(ntcp) == 8 } {
    create_udp_connection 90 91 udp 5
    create_udp_connection 138 92 udp 5
    create_udp_connection 95 96 udp 5
    create_udp_connection 96 97 udp 5
    create_udp_connection 16 98 udp 5
    create_udp_connection 105 106 udp 5
    create_udp_connection 112 54 udp 5
    create_udp_connection 113 114 udp 5
}


if { $opt(ntcp) == 16 } {

    create_udp_connection 0 12 udp 5
    create_udp_connection 26 38 udp 5
    create_udp_connection 52 64 udp 5
    create_udp_connection 78 90 udp 5
    create_udp_connection 104 116 udp 5
    create_udp_connection 130 142 udp 5
    create_udp_connection 111 68 udp 5
    create_udp_connection 141 18 udp 5
    create_udp_connection 17 19 udp 5
```

```
    create_udp_connection 0 15 udp 5
    create_udp_connection 2 18 udp 5
    create_udp_connection 4 90 udp 5
    create_udp_connection 6 62 udp 5
    create_udp_connection 8 104 udp 5
    create_udp_connection 10 42 udp 5
    create_udp_connection 12 97 udp 5
}


if { $opt(ntcp) == 32 } {
    create_udp_connection 141 18 udp 5
    create_udp_connection 17 19 udp 5
    create_udp_connection 20 21 udp 5
    create_udp_connection 120 22 udp 5
    create_udp_connection 24 25 udp 5
    create_udp_connection 28 30 udp 5
    create_udp_connection 33 34 udp 5
    create_udp_connection 31 35 udp 5
    create_udp_connection 135 36 udp 5
    create_udp_connection 148 37 udp 5
    create_udp_connection 23 38 udp 5
    create_udp_connection 147 39 udp 5
    create_udp_connection 43 45 udp 5
    create_udp_connection 44 49 udp 5
    create_udp_connection 50 46 udp 5
    create_udp_connection 47 48 udp 5
    create_udp_connection 60 61 udp 5
    create_udp_connection 71 72 udp 5
    create_udp_connection 67 73 udp 5
    create_udp_connection 66 74 udp 5
    create_udp_connection 77 78 udp 5
    create_udp_connection 80 82 udp 5
    create_udp_connection 52 84 udp 5
    create_udp_connection 83 85 udp 5
    create_udp_connection 88 89 udp 5
    create_udp_connection 90 91 udp 5
    create_udp_connection 138 92 udp 5
    create_udp_connection 95 96 udp 5
    create_udp_connection 96 97 udp 5
    create_udp_connection 16 98 udp 5
    create_udp_connection 105 106 udp 5
    create_udp_connection 112 54 udp 5
}


if { $opt(ntcp) == 50 } {
    create_udp_connection 1 2 udp 5
    create_udp_connection 3 5 udp 5
    create_udp_connection 4 6 udp 5
    create_udp_connection 8 9 udp 5
```

```
    create_udp_connection 5 10 udp 5
    create_udp_connection 9 11 udp 5
    create_udp_connection 144 12 udp 5
    create_udp_connection 16 17 udp 5
    create_udp_connection 146 142 udp 5
    create_udp_connection 141 18 udp 5
    create_udp_connection 17 19 udp 5
    create_udp_connection 20 21 udp 5
    create_udp_connection 120 22 udp 5
    create_udp_connection 24 25 udp 5
    create_udp_connection 28 30 udp 5
    create_udp_connection 33 34 udp 5
    create_udp_connection 31 35 udp 5
    create_udp_connection 135 36 udp 5
    create_udp_connection 148 37 udp 5
    create_udp_connection 23 38 udp 5
    create_udp_connection 147 39 udp 5
    create_udp_connection 43 45 udp 5
    create_udp_connection 44 49 udp 5
    create_udp_connection 50 46 udp 5
    create_udp_connection 47 48 udp 5
    create_udp_connection 60 61 udp 5
    create_udp_connection 71 72 udp 5
    create_udp_connection 67 73 udp 5
    create_udp_connection 66 74 udp 5
    create_udp_connection 77 78 udp 5
    create_udp_connection 80 82 udp 5
    create_udp_connection 52 84 udp 5
    create_udp_connection 83 85 udp 5
    create_udp_connection 88 89 udp 5
    create_udp_connection 90 91 udp 5
    create_udp_connection 138 92 udp 5
    create_udp_connection 95 96 udp 5
    create_udp_connection 96 97 udp 5
    create_udp_connection 16 98 udp 5
    create_udp_connection 105 106 udp 5
    create_udp_connection 112 54 udp 5
    create_udp_connection 113 114 udp 5
    create_udp_connection 55 115 udp 5
    create_udp_connection 116 117 udp 5
    create_udp_connection 122 124 udp 5
    create_udp_connection 56 125 udp 5
    create_udp_connection 124 126 udp 5
    create_udp_connection 125 57 udp 5
    create_udp_connection 131 132 udp 5
    create_udp_connection 133 134 udp 5
}
```

- Execution
  - TCP (sim-rand-tcp.sh)

```
#! /bin/sh
export PATH=$PATH:/ns2/bin:/ns2/tcl8.4.11/unix:/ns2/tk8.4.11/unix

mkdir result
mkdir result/randway_tcp

for B in 1 2
do
   mkdir result/randway_tcp/DSR$B
   for G in 0 1
   do
      for N in 8 16 32 50
      do
      if \
         ns main-tcp.tcl -alpha 0.01 -beta $B -gamma $G -nn 150 -ntcp $N -rate 2e6 \
         -cp tcp-randway.tcl -sc randway5.dst -stop 120 > output.$B.$N ;\
         then
         mkdir result/randway_tcp/DSR$B/g$G.tcp$N
         mv *.tr result/randway_tcp/DSR$B/g$G.tcp$N/
         mv *.log result/randway_tcp/DSR$B/g$G.tcp$N/

         else
         mkdir result/randway_tcp/DSR$B/g$G.tcp$N.fail
         mv *.tr result/randway_tcp/DSR$B/g$G.tcp$N.fail/
         mv *.log result/randway_tcp/DSR$B/g$G.tcp$N.fail/
      fi
      done
   done
done
```

  - UDP (sim-rand-udp.sh)

```
#! /bin/sh
export PATH=$PATH:/ns2/bin:/ns2/tcl8.4.11/unix:/ns2/tk8.4.11/unix

mkdir result
mkdir result/randway_udp

for B in 1 2
do
   mkdir result/randway_udp/DSR$B
   for G in 0 1
   do
      for N in 8 16 32 50
      do
      if \
         ns main-udp.tcl -alpha 0.01 -beta $B -gamma $G -nn 150 -ntcp $N -rate 2e6 \
```

```
        -cp udp-randway.tcl -sc randway5.dst -stop 120 > output.$B.$N ;\
        then
        mkdir result/randway_udp/DSR$B/g$G.udp$N
        mv *.tr result/randway_udp/DSR$B/g$G.udp$N/
        mv *.log result/randway_udp/DSR$B/g$G.udp$N/

        else
        mkdir result/randway_udp/DSR$B/g$G.udp$N.fail
        mv *.tr result/randway_udp/DSR$B/g$G.udp$N.fail/
        mv *.log result/randway_udp/DSR$B/g$G.udp$N.fail/
      fi
      done
   done
done
```

- Analysis
  - TCP (analysis-rand-tcp.pl)

```
#! /usr/bin/perl

foreach $B ( 1, 2 ) {
   print "beta: $B ";
   open MAC_DROPS, "> result/randway_tcp/DSR$B/mac_drops.log";
   open RT_ERROR, "> result/randway_tcp/DSR$B/route_errors.log";
   open RT_CHANGE, "> result/randway_tcp/DSR$B/route_changes.log";

   foreach $N ( 8 ,16,32,50  ) {
      print "N: $N ";

      print RT_CHANGE "$N";
      print RT_ERROR "$N";

         foreach $G ( "g0", "g1" ) {
         print " gamma $G\n";

         open LOSS, "> result/randway_tcp/DSR$B/$G.tcp$N/loss.log";
         open Overhead, "> result/randway_tcp/DSR$B/$G.tcp$N/overhead.log";
         # variables for MAC errors

         print MAC_DROPS "$N $G";
         # variables for MAC errors
         $col = 0;
         $dup = 0;
         $ret = 0;
         $bsy = 0;
         $cbk = 0;
         # variable for Route Error
         $rerr = 0;
         # counting data packets (send, drop and recv...)
         $nsend = 0;
```

```
$nrecv = 0;
#counting the number of packets RTR  and MAC
$nRTR_OH = 0;
$nMAC_OH = 0;
# route changes. . .
$RChangeFlows = 0;
   open TRACE, "result/randway_tcp/DSR$B/$G.tcp$N/out.tr";
while (<TRACE>) {
   @ll = split(' ');
   # if packet drop ...
   if ( $ll[0] eq "d" && $ll[18] eq "MAC"){
      # collision
      elsif ( $ll[20] eq "COL" )
         {$col++;}
      # duplicate
      elsif ( $ll[20] eq "DUP" )
         {$dup++;}
      # retry exceeded count
      elsif ( $ll[20] eq "RET" )
         {$ret++;}
      # busy
      elsif ( $ll[20] eq "BSY" )
         {$bsy++;}
      # MAC callback
      elsif ( $ll[20] eq "CBK" )
         {$cbk++;}
   }
   # if route error then count
   if ($ll[0] eq "RERR" )
      {$rerr +=1;}

   if ( $ll[0] eq "s" && $ll[18] eq "AGT")
      {$nsend +=1;}
   if ( $ll[0] eq "r" && $ll[18] eq "AGT" )
      {$nrecv +=1;}

   # may be tcp, ack, dsr. Count only -It (type of packet) = dsr
   if ( $ll[18] eq "RTR" && $ll[34] eq "DSR" )
      { $nRTR_OH++;}
   # ROUTING OVERHEAD IN MAC
   if ( $ll[18] eq "MAC" && $ll[34] eq "DSR" )
      { $nMAC_OH++;}

   # calculation of route changes FOR ALL FLOWS!!
   if ($ll[0] eq "RChange"){

      if ($N == 8) {
         if ( ($ll[4] eq "90" && $ll[6] eq "91")
         || ($ll[4] eq "138" && $ll[6] eq "92")
         || ($ll[4] eq "95" && $ll[6] eq "96")
         || ($ll[4] eq "96" && $ll[6] eq "97")
```

```
   || ($ll[4] eq "16" && $ll[6] eq "98")
   || ($ll[4] eq "105" && $ll[6] eq "106")
   || ($ll[4] eq "112" && $ll[6] eq "54")
   || ($ll[4] eq "113" && $ll[6] eq "114") )
      {$RChangeFlows++;}
}elsif ($N == 16){
   if ( ($ll[4] eq "0" && $ll[6] eq "12")
   || ($ll[4] eq "26" && $ll[6] eq "38")
   || ($ll[4] eq "52" && $ll[6] eq "64")
   || ($ll[4] eq "78" && $ll[6] eq "90")
   || ($ll[4] eq "104" && $ll[6] eq "116")
   || ($ll[4] eq "130" && $ll[6] eq "142")
   || ($ll[4] eq "111" && $ll[6] eq "68")
   || ($ll[4] eq "141" && $ll[6] eq "18")

   || ($ll[4] eq "17" && $ll[6] eq "19")
   || ($ll[4] eq "0" && $ll[6] eq "15")
   || ($ll[4] eq "2" && $ll[6] eq "18")
   || ($ll[4] eq "4" && $ll[6] eq "90")
   || ($ll[4] eq "6" && $ll[6] eq "62")
   || ($ll[4] eq "8" && $ll[6] eq "104")
   || ($ll[4] eq "10" && $ll[6] eq "42")
   || ($ll[4] eq "12" && $ll[6] eq "97") )
      {$RChangeFlows++;}

}elsif ($N == 32){
   if ( ($ll[4] eq "141" && $ll[6] eq "18")
   || ($ll[4] eq "17" && $ll[6] eq "19")
   || ($ll[4] eq "20" && $ll[6] eq "21")
   || ($ll[4] eq "120" && $ll[6] eq "22")
   || ($ll[4] eq "24" && $ll[6] eq "25")
   || ($ll[4] eq "28" && $ll[6] eq "30")
   || ($ll[4] eq "33" && $ll[6] eq "34")
   || ($ll[4] eq "31" && $ll[6] eq "35")

   || ($ll[4] eq "135" && $ll[6] eq "36")
   || ($ll[4] eq "148" && $ll[6] eq "37")
   || ($ll[4] eq "23" && $ll[6] eq "38")
   || ($ll[4] eq "147" && $ll[6] eq "39")
   || ($ll[4] eq "43" && $ll[6] eq "45")
   || ($ll[4] eq "44" && $ll[6] eq "49")
   || ($ll[4] eq "50" && $ll[6] eq "46")
   || ($ll[4] eq "47" && $ll[6] eq "48")

   || ($ll[4] eq "60" && $ll[6] eq "61")
   || ($ll[4] eq "71" && $ll[6] eq "72")
   || ($ll[4] eq "67" && $ll[6] eq "73")
   || ($ll[4] eq "66" && $ll[6] eq "74")
   || ($ll[4] eq "77" && $ll[6] eq "78")
   || ($ll[4] eq "80" && $ll[6] eq "82")
   || ($ll[4] eq "52" && $ll[6] eq "84")
```

```
   || ($ll[4] eq "83" && $ll[6] eq "85")

   || ($ll[4] eq "88" && $ll[6] eq "89")
   || ($ll[4] eq "90" && $ll[6] eq "91")
   || ($ll[4] eq "138" && $ll[6] eq "92")
   || ($ll[4] eq "95" && $ll[6] eq "96")
   || ($ll[4] eq "96" && $ll[6] eq "97")
   || ($ll[4] eq "16" && $ll[6] eq "98")
   || ($ll[4] eq "105" && $ll[6] eq "106")
   || ($ll[4] eq "112" && $ll[6] eq "54") )
      {$RChangeFlows++;}

}elsif ($N == 50){
   if ( ($ll[4] eq "1" && $ll[6] eq "2")
   || ($ll[4] eq "3" && $ll[6] eq "5")
   || ($ll[4] eq "4" && $ll[6] eq "6")
   || ($ll[4] eq "8" && $ll[6] eq "9")
   || ($ll[4] eq "5" && $ll[6] eq "10")

   || ($ll[4] eq "9" && $ll[6] eq "11")
   || ($ll[4] eq "144" && $ll[6] eq "12")
   || ($ll[4] eq "16" && $ll[6] eq "17")
   || ($ll[4] eq "146" && $ll[6] eq "142")
   || ($ll[4] eq "141" && $ll[6] eq "18")

   || ($ll[4] eq "17" && $ll[6] eq "19")
   || ($ll[4] eq "20" && $ll[6] eq "21")
   || ($ll[4] eq "120" && $ll[6] eq "22")
   || ($ll[4] eq "24" && $ll[6] eq "25")
   || ($ll[4] eq "28" && $ll[6] eq "30")

   || ($ll[4] eq "33" && $ll[6] eq "34")
   || ($ll[4] eq "31" && $ll[6] eq "35")
   || ($ll[4] eq "135" && $ll[6] eq "36")
   || ($ll[4] eq "148" && $ll[6] eq "37")
   || ($ll[4] eq "23" && $ll[6] eq "38")

   || ($ll[4] eq "147" && $ll[6] eq "39")
   || ($ll[4] eq "43" && $ll[6] eq "45")
   || ($ll[4] eq "44" && $ll[6] eq "49")
   || ($ll[4] eq "50" && $ll[6] eq "46")
   || ($ll[4] eq "47" && $ll[6] eq "48")

   || ($ll[4] eq "60" && $ll[6] eq "61")
   || ($ll[4] eq "71" && $ll[6] eq "72")
   || ($ll[4] eq "67" && $ll[6] eq "73")
   || ($ll[4] eq "66" && $ll[6] eq "74")
   || ($ll[4] eq "77" && $ll[6] eq "78")

   || ($ll[4] eq "80" && $ll[6] eq "82")
   || ($ll[4] eq "52" && $ll[6] eq "84")
```

```
                    || ($ll[4] eq "83" && $ll[6] eq "85")
                    || ($ll[4] eq "88" && $ll[6] eq "89")
                    || ($ll[4] eq "90" && $ll[6] eq "91")

                    || ($ll[4] eq "138" && $ll[6] eq "92")
                    || ($ll[4] eq "95" && $ll[6] eq "96")
                    || ($ll[4] eq "96" && $ll[6] eq "97")
                    || ($ll[4] eq "16" && $ll[6] eq "98")
                    || ($ll[4] eq "105" && $ll[6] eq "106")

                    || ($ll[4] eq "112" && $ll[6] eq "54")
                    || ($ll[4] eq "113" && $ll[6] eq "114")
                    || ($ll[4] eq "55" && $ll[6] eq "115")
                    || ($ll[4] eq "116" && $ll[6] eq "117")
                    || ($ll[4] eq "122" && $ll[6] eq "124")

                    || ($ll[4] eq "56" && $ll[6] eq "125")
                    || ($ll[4] eq "124" && $ll[6] eq "126")
                    || ($ll[4] eq "125" && $ll[6] eq "57")
                    || ($ll[4] eq "131" && $ll[6] eq "132")
                    || ($ll[4] eq "133" && $ll[6] eq "134") )
                       {$RChangeFlows++;}
                  }
               }

            }
            close TRACE;
            print LOSS "$nsend $nrecv";
            close LOSS;

            print RT_ERROR " $rerr";
            $oh1 = $nRTR_OH/120;
            $oh2 = $nMAC_OH/120;
            print Overhead "RTR: $nRTR_OH MAC: $nMAC_OH => $oh1 $oh2\n";
            close Overhead;

            print MAC_DROPS " end: $end col: $col dup: $dup ret: $ret err: $err; sta: $sta
bsy: $bsy dst: $dst nrte: $nrte loop: $loop ttl: $ttl ifq: $ifq tout: $tout cbk: $cbk sal:
$sal arp: $arp fil: $fil out: $out\n";
            print RT_CHANGE " $RChangeFlows";

         }
         print "\n";
         print RT_ERROR "\n";
         print RT_CHANGE "\n";
      }
      close RT_ERROR;
      close MAC_DROPS;
      close RT_CHANGE;
   }
```

- UDP (analysis-rand-udp.pl)

```
#! /usr/bin/perl
foreach $B ( 1, 2 ) {
   print "beta: $B ";
   open MAC_DROPS, "> result/randway_udp/DSR$B/mac_drops.log";
   open RT_ERROR, "> result/randway_udp/DSR$B/route_errors.log";
   open RT_CHANGE, "> result/randway_udp/DSR$B/route_changes.log";

   foreach $N ( 8 ,16,32,50  ) {
      print "N: $N ";

      print RT_CHANGE "$N";
      print RT_ERROR "$N";

         foreach $G ( "g0", "g1" ) {
         print " gamma $G\n";

         open LOSS, "> result/randway_udp/DSR$B/$G.udp$N/loss.log";
         open Overhead, "> result/randway_udp/DSR$B/$G.udp$N/overhead.log";
         # variables for MAC errors

         print MAC_DROPS "$N $G";
         # variables for MAC errors
         $col = 0;
         $dup = 0;
         $ret = 0;
         $bsy = 0;
         $cbk = 0;
         # variable for Route Error
         $rerr = 0;
         # counting data packets (send, drop and recv...)
         $nsend = 0;
         $nrecv = 0;
         #counting the number of packets RTR  and MAC
         $nRTR_OH = 0;
         $nMAC_OH = 0;
         # route changes. . .
         $RChangeFlows = 0;
            open TRACE, "result/randway_udp/DSR$B/$G.udp$N/out.tr";
         while (<TRACE>) {
            @ll = split(' ');
            # if packet drop ...
            if ( $ll[0] eq "d" && $ll[18] eq "MAC"){
               # collision
               elsif ( $ll[20] eq "COL" )
                  {$col++;}
               # duplicate
               elsif ( $ll[20] eq "DUP" )
                  {$dup++;}
               # retry exceeded count
```

```
   elsif ( $ll[20] eq "RET" )
      {$ret++;}
   # busy
   elsif ( $ll[20] eq "BSY" )
      {$bsy++;}
   # MAC callback
   elsif ( $ll[20] eq "CBK" )
      {$cbk++;}
}
# if route error then count
if ($ll[0] eq "RERR" )
   {$rerr +=1;}

if ( $ll[0] eq "s" && $ll[18] eq "AGT")
   {$nsend +=1;}
if ( $ll[0] eq "r" && $ll[18] eq "AGT" )
   {$nrecv +=1;}

# may be cbr, ack, dsr. Count only -It (type of packet) = dsr
if ( $ll[18] eq "RTR" && $ll[34] eq "DSR" )
   { $nRTR_OH++;}
# ROUTING OVERHEAD IN MAC
if ( $ll[18] eq "MAC" && $ll[34] eq "DSR" )
   { $nMAC_OH++;}

# calculation of route changes FOR ALL FLOWS!!
if ($ll[0] eq "RChange"){

   if ($N == 8) {
      if ( ($ll[4] eq "90" && $ll[6] eq "91")
      || ($ll[4] eq "138" && $ll[6] eq "92")
      || ($ll[4] eq "95" && $ll[6] eq "96")
      || ($ll[4] eq "96" && $ll[6] eq "97")
      || ($ll[4] eq "16" && $ll[6] eq "98")
      || ($ll[4] eq "105" && $ll[6] eq "106")
      || ($ll[4] eq "112" && $ll[6] eq "54")
      || ($ll[4] eq "113" && $ll[6] eq "114") )
         {$RChangeFlows++;}
   }elsif ($N == 16){
      if ( ($ll[4] eq "0" && $ll[6] eq "12")
      || ($ll[4] eq "26" && $ll[6] eq "38")
      || ($ll[4] eq "52" && $ll[6] eq "64")
      || ($ll[4] eq "78" && $ll[6] eq "90")
      || ($ll[4] eq "104" && $ll[6] eq "116")
      || ($ll[4] eq "130" && $ll[6] eq "142")
      || ($ll[4] eq "111" && $ll[6] eq "68")
      || ($ll[4] eq "141" && $ll[6] eq "18")

      || ($ll[4] eq "17" && $ll[6] eq "19")
      || ($ll[4] eq "0" && $ll[6] eq "15")
      || ($ll[4] eq "2" && $ll[6] eq "18")
```

```
   || ($ll[4] eq "4" && $ll[6] eq "90")
   || ($ll[4] eq "6" && $ll[6] eq "62")
   || ($ll[4] eq "8" && $ll[6] eq "104")
   || ($ll[4] eq "10" && $ll[6] eq "42")
   || ($ll[4] eq "12" && $ll[6] eq "97") )
      {$RChangeFlows++;}

}elsif ($N == 32){
   if ( ($ll[4] eq "141" && $ll[6] eq "18")
   || ($ll[4] eq "17" && $ll[6] eq "19")
   || ($ll[4] eq "20" && $ll[6] eq "21")
   || ($ll[4] eq "120" && $ll[6] eq "22")
   || ($ll[4] eq "24" && $ll[6] eq "25")
   || ($ll[4] eq "28" && $ll[6] eq "30")
   || ($ll[4] eq "33" && $ll[6] eq "34")
   || ($ll[4] eq "31" && $ll[6] eq "35")

   || ($ll[4] eq "135" && $ll[6] eq "36")
   || ($ll[4] eq "148" && $ll[6] eq "37")
   || ($ll[4] eq "23" && $ll[6] eq "38")
   || ($ll[4] eq "147" && $ll[6] eq "39")
   || ($ll[4] eq "43" && $ll[6] eq "45")
   || ($ll[4] eq "44" && $ll[6] eq "49")
   || ($ll[4] eq "50" && $ll[6] eq "46")
   || ($ll[4] eq "47" && $ll[6] eq "48")

   || ($ll[4] eq "60" && $ll[6] eq "61")
   || ($ll[4] eq "71" && $ll[6] eq "72")
   || ($ll[4] eq "67" && $ll[6] eq "73")
   || ($ll[4] eq "66" && $ll[6] eq "74")
   || ($ll[4] eq "77" && $ll[6] eq "78")
   || ($ll[4] eq "80" && $ll[6] eq "82")
   || ($ll[4] eq "52" && $ll[6] eq "84")
   || ($ll[4] eq "83" && $ll[6] eq "85")

   || ($ll[4] eq "88" && $ll[6] eq "89")
   || ($ll[4] eq "90" && $ll[6] eq "91")
   || ($ll[4] eq "138" && $ll[6] eq "92")
   || ($ll[4] eq "95" && $ll[6] eq "96")
   || ($ll[4] eq "96" && $ll[6] eq "97")
   || ($ll[4] eq "16" && $ll[6] eq "98")
   || ($ll[4] eq "105" && $ll[6] eq "106")
   || ($ll[4] eq "112" && $ll[6] eq "54") )
      {$RChangeFlows++;}

}elsif ($N == 50){
   if ( ($ll[4] eq "1" && $ll[6] eq "2")
   || ($ll[4] eq "3" && $ll[6] eq "5")
   || ($ll[4] eq "4" && $ll[6] eq "6")
   || ($ll[4] eq "8" && $ll[6] eq "9")
   || ($ll[4] eq "5" && $ll[6] eq "10")
```

```
|| ($ll[4] eq "9" && $ll[6] eq "11")
|| ($ll[4] eq "144" && $ll[6] eq "12")
|| ($ll[4] eq "16" && $ll[6] eq "17")
|| ($ll[4] eq "146" && $ll[6] eq "142")
|| ($ll[4] eq "141" && $ll[6] eq "18")

|| ($ll[4] eq "17" && $ll[6] eq "19")
|| ($ll[4] eq "20" && $ll[6] eq "21")
|| ($ll[4] eq "120" && $ll[6] eq "22")
|| ($ll[4] eq "24" && $ll[6] eq "25")
|| ($ll[4] eq "28" && $ll[6] eq "30")

|| ($ll[4] eq "33" && $ll[6] eq "34")
|| ($ll[4] eq "31" && $ll[6] eq "35")
|| ($ll[4] eq "135" && $ll[6] eq "36")
|| ($ll[4] eq "148" && $ll[6] eq "37")
|| ($ll[4] eq "23" && $ll[6] eq "38")

|| ($ll[4] eq "147" && $ll[6] eq "39")
|| ($ll[4] eq "43" && $ll[6] eq "45")
|| ($ll[4] eq "44" && $ll[6] eq "49")
|| ($ll[4] eq "50" && $ll[6] eq "46")
|| ($ll[4] eq "47" && $ll[6] eq "48")

|| ($ll[4] eq "60" && $ll[6] eq "61")
|| ($ll[4] eq "71" && $ll[6] eq "72")
|| ($ll[4] eq "67" && $ll[6] eq "73")
|| ($ll[4] eq "66" && $ll[6] eq "74")
|| ($ll[4] eq "77" && $ll[6] eq "78")

|| ($ll[4] eq "80" && $ll[6] eq "82")
|| ($ll[4] eq "52" && $ll[6] eq "84")
|| ($ll[4] eq "83" && $ll[6] eq "85")
|| ($ll[4] eq "88" && $ll[6] eq "89")
|| ($ll[4] eq "90" && $ll[6] eq "91")

|| ($ll[4] eq "138" && $ll[6] eq "92")
|| ($ll[4] eq "95" && $ll[6] eq "96")
|| ($ll[4] eq "96" && $ll[6] eq "97")
|| ($ll[4] eq "16" && $ll[6] eq "98")
|| ($ll[4] eq "105" && $ll[6] eq "106")

|| ($ll[4] eq "112" && $ll[6] eq "54")
|| ($ll[4] eq "113" && $ll[6] eq "114")
|| ($ll[4] eq "55" && $ll[6] eq "115")
|| ($ll[4] eq "116" && $ll[6] eq "117")
|| ($ll[4] eq "122" && $ll[6] eq "124")

|| ($ll[4] eq "56" && $ll[6] eq "125")
|| ($ll[4] eq "124" && $ll[6] eq "126")
```

```
                    || ($ll[4] eq "125" && $ll[6] eq "57")
                    || ($ll[4] eq "131" && $ll[6] eq "132")
                    || ($ll[4] eq "133" && $ll[6] eq "134") )
                       {$RChangeFlows++;}
                 }
              }

           }
           close TRACE;
           print LOSS "$nsend $nrecv";
           close LOSS;

           print RT_ERROR " $rerr";
           $oh1 = $nRTR_OH/120;
           $oh2 = $nMAC_OH/120;
           print Overhead "RTR: $nRTR_OH MAC: $nMAC_OH => $oh1 $oh2\n";
           close Overhead;

           print MAC_DROPS " end: $end col: $col dup: $dup ret: $ret err: $err; sta: $sta
bsy: $bsy dst: $dst nrte: $nrte loop: $loop ttl: $ttl ifq: $ifq tout: $tout cbk: $cbk sal:
$sal arp: $arp fil: $fil out: $out\n";
           print RT_CHANGE " $RChangeFlows";

        }
        print "\n";
        print RT_ERROR "\n";
        print RT_CHANGE "\n";
     }
     close RT_ERROR;
     close MAC_DROPS;
     close RT_CHANGE;
  }
```

Manhattan

- Connections set up
    - TCP (tcp-man-n200.tcl)

```
if { $opt(ntcp) == 20 } {
   create_tcp_connection 0 82 tcp 3
   create_tcp_connection 52 64 tcp 3
   create_tcp_connection 104 116 tcp 3
   create_tcp_connection 156 168 tcp 3
   create_tcp_connection 56 198 tcp 3

   create_tcp_connection 192 156 tcp 3
   create_tcp_connection 4 160 tcp 3
   create_tcp_connection 8 124 tcp 3
   create_tcp_connection 12 111 tcp 3
   create_tcp_connection 15 168 tcp 3

   create_tcp_connection 94 12 tcp 3
   create_tcp_connection 52 64 tcp 3
   create_tcp_connection 104 116 tcp 3
   create_tcp_connection 159 87 tcp 3
   create_tcp_connection 133 18 tcp 3

   create_tcp_connection 120 156 tcp 3
   create_tcp_connection 41 168 tcp 3
   create_tcp_connection 81 159 tcp 3
   create_tcp_connection 28 179 tcp 3
   create_tcp_connection 69 155 tcp 3
}
```

    - UDP (tcp-man-n200.tcl)

```
if { $opt(ntcp) == 20 } {
   create_udp_connection 0 82 udp 3
   create_udp_connection 52 64 udp 3
   create_udp_connection 104 116 udp 3
   create_udp_connection 156 168 udp 3
   create_udp_connection 56 198 udp 3

   create_udp_connection 192 156 udp 3
   create_udp_connection 4 160 udp 3
   create_udp_connection 8 124 udp 3
   create_udp_connection 12 111 udp 3
   create_udp_connection 15 168 udp 3

   create_udp_connection 94 12 udp 3
   create_udp_connection 52 64 udp 3
   create_udp_connection 104 116 udp 3
   create_udp_connection 159 87 udp 3
```

```
    create_udp_connection 133 18 udp 3

    create_udp_connection 120 156 udp 3
    create_udp_connection 41 168 udp 3
    create_udp_connection 81 159 udp 3
    create_udp_connection 28 179 udp 3
    create_udp_connection 69 155 udp 3
}
```

- Execution
  - TCP (sim-man-tcp.sh)

```
#! /bin/sh
export PATH=$PATH:/ns2/bin:/ns2/tcl8.4.11/unix:/ns2/tk8.4.11/unix

mkdir result
mkdir result/manhattan_tcp

for G in 0 1
do
   for B in 1 2
   do

      mkdir result/manhattan_tcp/DSR$B
      for M in 0 1 2 5 10 20
      do
         if \
            ns main-tcp.tcl -alpha 0.01 -gamma $G -nn 200 -ntcp 20 -beta $B \
            -cp tcp-man-n200.tcl -sc man$M-s1.dst \
            -stop 200 -seed 0 -rate 2e6 -dist 60; \
            then
            mkdir result/manhattan_tcp/DSR$B/g$G.tcp.man$M
            mv *.tr result/manhattan_tcp/DSR$B/g$G.tcp.man$M/
            mv *.log result/manhattan_tcp/DSR$B/g$G.tcp.man$M/
            else
            mkdir result/manhattan_tcp/DSR$B/g$G.tcp.man$M.fail
            mv *.tr result/manhattan_tcp/DSR$B/g$G.tcp.man$M.fail/
            mv *.log result/manhattan_tcp/DSR$B/g$G.tcp.man$M.fail/
         fi
         date
      done
   done
done
```

  - UDP (sim-man-udp.sh)

```
#! /bin/sh
export PATH=$PATH:/ns2/bin:/ns2/tcl8.4.11/unix:/ns2/tk8.4.11/unix
mkdir result
mkdir result/manhattan_udp
for G in 0 1
do
   for B in 1 2
   do
      mkdir result/manhattan_udp/DSR$B
      for M in 0 1 2 5 10 20
      do
         if \
            ns main-udp.tcl -alpha 0.01 -gamma $G -nn 200 -nudp 20 -beta $B \
```

```
            -cp udp-man-n200.tcl -sc man$M-s1.dst \
            -stop 200 -seed 0 -rate 2e6 -dist 60; \
            then
            mkdir result/manhattan_udp/DSR$B/g$G.udp.man$M
            mv *.tr result/manhattan_udp/DSR$B/g$G.udp.man$M/
            mv *.log result/manhattan_udp/DSR$B/g$G.udp.man$M/
            else
            mkdir result/manhattan_udp/DSR$B/g$G.udp.man$M.fail
            mv *.tr result/manhattan_udp/DSR$B/g$G.udp.man$M.fail/
            mv *.log result/manhattan_udp/DSR$B/g$G.udp.man$M.fail/
         fi
      done
   done
done
```

- Analysis
  - TCP (analysis-man-tcp.pl)

```
#! /usr/bin/perl

foreach $B ( 1, 2 ) {
   print "beta: $B ";
   open MAC_DROPS, "> result/manhattan_tcp/DSR$B/mac_drops.log";
   open RT_ERROR, "> result/manhattan_tcp/DSR$B/route_errors.log";
   open RT_CHANGE, "> result/manhattan_tcp/DSR$B/route_changes.log";

   foreach $N ( 0, 1, 2, 5, 10, 20 ) {
      print "N: $N ";

      print RT_CHANGE "$N";
      print RT_ERROR "$N";

         foreach $G ( "g0", "g1" ) {
         print " gamma $G\n";
         open LOSS, "> result/manhattan_tcp/DSR$B/$G.tcp.man$N/loss.log";
         open Overhead, "> result/manhattan_tcp/DSR$B/$G.tcp.man$N/overhead.log";
         # variables for MAC errors

         print MAC_DROPS "$N $G";
         # variables for MAC errors
         $col = 0;
         $dup = 0;
         $ret = 0;
         $bsy = 0;
         $cbk = 0;
         # variable for Route Error
         $rerr = 0;
         # counting data packets (send, drop and recv...)
         $nsend = 0;
```

```
$nrecv = 0;
#counting the number of packets RTR  and MAC
$nRTR_OH = 0;
$nMAC_OH = 0;
# route changes. . .
$RChangeFlows = 0;
   open TRACE, "result/manhattan_tcp/DSR$B/$G.tcp.man$N/out.tr";
while (<TRACE>) {
   @ll = split(' ');
   # if packet drop ...
   if ( $ll[0] eq "d" && $ll[18] eq "MAC"){
      # collision
      elsif ( $ll[20] eq "COL" )
         {$col++;}
      # duplicate
      elsif ( $ll[20] eq "DUP" )
         {$dup++;}
      # retry exceeded count
      elsif ( $ll[20] eq "RET" )
         {$ret++;}
      # MAC callback
      elsif ( $ll[20] eq "CBK" )
         {$cbk++;}
   }
   # if route error then count
   if ($ll[0] eq "RERR" )
      {$rerr +=1;}

   if ( $ll[0] eq "s" && $ll[18] eq "AGT")
      {$nsend +=1;}
   if ( $ll[0] eq "r" && $ll[18] eq "AGT" )
      {$nrecv +=1;}

   # may be tcp, ack, dsr. Count only -It (type of packet) = dsr
   if ( $ll[18] eq "RTR" && $ll[34] eq "DSR" )
      { $nRTR_OH++;}
   # ROUTING OVERHEAD IN MAC
   if ( $ll[18] eq "MAC" && $ll[34] eq "DSR" )
      { $nMAC_OH++;}

   # calculation of route changes FOR ALL FLOWS!!
   if ($ll[0] eq "RChange"){
      if ( ($ll[4] eq "0" && $ll[6] eq "82")
      || ($ll[4] eq "52" && $ll[6] eq "64")
      || ($ll[4] eq "104" && $ll[6] eq "116")
      || ($ll[4] eq "156" && $ll[6] eq "168")
      || ($ll[4] eq "56" && $ll[6] eq "198")

      || ($ll[4] eq "192" && $ll[6] eq "156")
      || ($ll[4] eq "4" && $ll[6] eq "160")
      || ($ll[4] eq "8" && $ll[6] eq "124")
```

```
                || ($ll[4] eq "12" && $ll[6] eq "111")
                || ($ll[4] eq "15" && $ll[6] eq "168")

                || ($ll[4] eq "94" && $ll[6] eq "12")
                || ($ll[4] eq "52" && $ll[6] eq "64")
                || ($ll[4] eq "104" && $ll[6] eq "116")
                || ($ll[4] eq "159" && $ll[6] eq "87")
                || ($ll[4] eq "133" && $ll[6] eq "18")

                || ($ll[4] eq "120" && $ll[6] eq "156")
                || ($ll[4] eq "41" && $ll[6] eq "168")
                || ($ll[4] eq "81" && $ll[6] eq "159")
                || ($ll[4] eq "28" && $ll[6] eq "179")
                || ($ll[4] eq "69" && $ll[6] eq "155")
                )
                   {$RChangeFlows++;}
             }

          }
          close TRACE;
          print LOSS "$nsend $nrecv";
          close LOSS;

          print RT_ERROR " $rerr";
          $oh1 = $nRTR_OH/150;
          $oh2 = $nMAC_OH/150;
          print Overhead "RTR: $nRTR_OH MAC: $nMAC_OH => $oh1 $oh2\n";
          close Overhead;

          print MAC_DROPS " end: $end col: $col dup: $dup ret: $ret err: $err; sta: $sta
bsy: $bsy dst: $dst nrte: $nrte loop: $loop ttl: $ttl ifq: $ifq tout: $tout cbk: $cbk sal:
$sal arp: $arp fil: $fil out: $out\n";
          print RT_CHANGE " $RChangeFlows";
       }
       print "\n";
       print RT_ERROR "\n";
       print RT_CHANGE "\n";
    }
    close RT_ERROR;
    close MAC_DROPS;
    close RT_CHANGE;
 }
```

- UDP (analysis-man-udp.pl)

```
 #! /usr/bin/perl

 foreach $B ( 1, 2 ) {
    print "beta: $B ";
    open MAC_DROPS, "> result/manhattan_udp/DSR$B/mac_drops.log";
```

```
open RT_ERROR, "> result/manhattan_udp/DSR$B/route_errors.log";
open RT_CHANGE, "> result/manhattan_udp/DSR$B/route_changes.log";

foreach $N ( 0, 1, 2, 5, 10, 20 ) {
   print "N: $N ";

   print RT_CHANGE "$N";
   print RT_ERROR "$N";

      foreach $G ( "g0", "g1" ) {
      print " gamma $G\n";

      open LOSS, "> result/manhattan_udp/DSR$B/$G.udp.man$N/loss.log";
      open Overhead, "> result/manhattan_udp/DSR$B/$G.udp.man$N/overhead.log";
      # variables for MAC errors

      print MAC_DROPS "$N $G";
      # variables for MAC errors
      $col = 0;
      $dup = 0;
      $ret = 0;
      $bsy = 0;
      $cbk = 0;
      # variable for Route Error
      $rerr = 0;
      # counting data packets (send, drop and recv...)
      $nsend = 0;
      $nrecv = 0;
      #counting the number of packets RTR  and MAC
      $nRTR_OH = 0;
      $nMAC_OH = 0;
      # route changes. . .
      $RChangeFlows = 0;
         open TRACE, "result/manhattan_udp/DSR$B/$G.udp.man$N/out.tr";
      while (<TRACE>) {
         @ll = split(' ');
         # if packet drop ...
         if ( $ll[0] eq "d" && $ll[18] eq "MAC"){
            # collision
            elsif ( $ll[20] eq "COL" )
               {$col++;}
            # duplicate
            elsif ( $ll[20] eq "DUP" )
               {$dup++;}
            # retry exceeded count
            elsif ( $ll[20] eq "RET" )
               {$ret++;}
            # MAC callback
            elsif ( $ll[20] eq "CBK" )
               {$cbk++;}
         }
```

```
   # if route error then count
   if ($ll[0] eq "RERR" )
      {$rerr +=1;}

   if ( $ll[0] eq "s" && $ll[18] eq "AGT")
      {$nsend +=1;}
   if ( $ll[0] eq "r" && $ll[18] eq "AGT" )
      {$nrecv +=1;}

   # may be udp, ack, dsr. Count only -It (type of packet) = dsr
   if ( $ll[18] eq "RTR" && $ll[34] eq "DSR" )
      { $nRTR_OH++;}
   # ROUTING OVERHEAD IN MAC
   if ( $ll[18] eq "MAC" && $ll[34] eq "DSR" )
      { $nMAC_OH++;}

   # calculation of route changes FOR ALL FLOWS!!
   if ($ll[0] eq "RChange"){
      if ( ($ll[4] eq "0" && $ll[6] eq "82")
      || ($ll[4] eq "52" && $ll[6] eq "64")
      || ($ll[4] eq "104" && $ll[6] eq "116")
      || ($ll[4] eq "156" && $ll[6] eq "168")
      || ($ll[4] eq "56" && $ll[6] eq "198")

      || ($ll[4] eq "192" && $ll[6] eq "156")
      || ($ll[4] eq "4" && $ll[6] eq "160")
      || ($ll[4] eq "8" && $ll[6] eq "124")
      || ($ll[4] eq "12" && $ll[6] eq "111")
      || ($ll[4] eq "15" && $ll[6] eq "168")

      || ($ll[4] eq "94" && $ll[6] eq "12")
      || ($ll[4] eq "52" && $ll[6] eq "64")
      || ($ll[4] eq "104" && $ll[6] eq "116")
      || ($ll[4] eq "159" && $ll[6] eq "87")
      || ($ll[4] eq "133" && $ll[6] eq "18")

      || ($ll[4] eq "120" && $ll[6] eq "156")
      || ($ll[4] eq "41" && $ll[6] eq "168")
      || ($ll[4] eq "81" && $ll[6] eq "159")
      || ($ll[4] eq "28" && $ll[6] eq "179")
      || ($ll[4] eq "69" && $ll[6] eq "155")
      )
         {$RChangeFlows++;}
   }

}
close TRACE;
print LOSS "$nsend $nrecv";
close LOSS;

print RT_ERROR " $rerr";
```

```
            $oh1 = $nRTR_OH/150;
            $oh2 = $nMAC_OH/150;
            print Overhead "RTR: $nRTR_OH MAC: $nMAC_OH => $oh1 $oh2\n";
            close Overhead;

            print MAC_DROPS " end: $end col: $col dup: $dup ret: $ret err: $err; sta: $sta
bsy: $bsy dst: $dst nrte: $nrte loop: $loop ttl: $ttl ifq: $ifq tout: $tout cbk: $cbk sal:
$sal arp: $arp fil: $fil out: $out\n";
            print RT_CHANGE " $RChangeFlows";
         }
         print "\n";
         print RT_ERROR "\n";
         print RT_CHANGE "\n";
      }
      close RT_ERROR;
      close MAC_DROPS;
      close RT_CHANGE;
   }
```

Appendix C

Folder "*common*"

- File "*packet.h*"

```
struct hdr_cmn {

 . . .

        #define XMIT_REASON_RTS 0x01
        #define XMIT_REASON_ACK 0x02

        // ktnahm add for DAMPEN policy
        #define XMIT_REASON_CONFIRM 0x03

        // Alonso - Rocha //cross-layer feature
        #define XMIT_REASON_HIGH_POWER 0x04

 . . .

};
```

Folder "*dsr*"

- File "*dsragent.h*"

```
private:

  // ktnahm for DAMPEN policy
  int beta_;     // bulk loss threshold activating maintenance operation
  int nlinkfail; // number of successive link failure

  // Alonso - Rocha
  int gamma_;  // activation of cross-layer feature

. . .
```

- File "*dsragent.cc*"

```
DSRAgent::DSRAgent(): . . .

. . .

  // ktnahm
```

```
  beta_ = 1;
  bind("beta_", &beta_);
  nlinkfail = 0;

  // Alonso - Rocha
  gamma_ = 0;
  bind("gamma_", &gamma_);

. . .
```

- File “*dsragent.cc*”

```
DSRAgent::sendOutPacketWithRoute(SRPacket& p, bool fresh, Time delay)
{

. . .

    trace("SF%ss %.9f _%s_ %d [%s -> %s] %d(%d) to %d %s",
   srh->num_addrs() ? "EST" : "",
   Scheduler::instance().clock(), net_id.dump(), cmnh->uid(),
   p.src.dump(), p.dest.dump(), flow_table[flowidx].flowId,
   srh->flow_header(), flow_table[flowidx].nextHop,
   srh->num_addrs() ? srh->dump() : "");

    // Alonso - Rocha: trace route change. Time, actual hops, source, destine and god hops
    if (srh->num_addrs()) {
   trace ("RChange %.9f h: %d %s -> %s -god %d",
      Scheduler::instance().clock(),p.route.length()-1,
      p.src.dump(), p.dest.dump(),
      God::instance()->hops(p.src.getNSAddr_t(), p.dest.getNSAddr_t())
      ); // monitoring number of route changes
   }
      // Alonso - Rocha END
. . .
}

DSRAgent::xmitFailed(Packet *pkt, const char* reason)
{
. . .

  if (srh->num_route_errors() >= MAX_ROUTE_ERRORS)
    {       trace("SDFU  %.5f _%s_ dumping maximally nested error %s  %d -> %d",
       Scheduler::instance().clock(), net_id.dump(),  tell_id.dump(),
       from_id.dump(),  to_id.dump());
      Packet::free(pkt);   // no drop needed
      pkt = 0;
      return;
    }
```

```
    // Alonso - Rocha. If mac informs that the node is still present, do not mark as route
error
    if (gamma_) {
     if ( cmh->xmit_reason_ == XMIT_REASON_HIGH_POWER
        && strcmp(reason, "DROP_RTR_MAC_CALLBACK") == 0 ) {
        Packet::free(pkt);
        pkt = 0;
        return;
     }
    }
    // monitoring number of route errors
    trace ("RERR %.5f %s -> %s -GH %d", Scheduler::instance().clock(), from_id.dump(),
to_id.dump(),God::instance()->hops(from_id.getNSAddr_t(), to_id.getNSAddr_t())
        );
    // Alonso - Rocha ends

    link_down *deadlink = &(srh->down_links()[srh->num_route_errors()]);
    deadlink->addr_type = srh->addrs()[srh->cur_addr()].addr_type;

  . . .

  }
```

Folder "*gen*"

- File "*ns_tcl.cc*"

```
  . . .

  Agent/DSRAgent set dport_ 255\n\
  Agent/DSRAgent set beta_ 2              ;# ktnahm: improved robustness for DSR\n\
  Agent/DSRAgent set gamma_ 1              ;# Alonso - Rocha: cross layer feature\n\
  \n\
  Agent/MIPBS set adSize_ 48\n\
  Agent/MIPBS set shift_ 0\n\

  . . .
```

Folder "*mac*"

- File "mac-802_11.*h*"

```
  . . .

  #include "mac-timers.h"
  #include "marshall.h"
  #include <math.h>

  // Added by Alonso - Rocha to support received power tracing
  #include <vector>
```

```
class EventTrace;

. . .

struct hdr_mac802_11 {

. . .

};

//Alonso - Rocha: structure for keeping the received power of the nodes
struct time_power {
   int pos;
   double power[20];
        u_int32_t idNode;
};

#define DSSS_MaxPropagationDelay        0.000002        // 2us   XXXX

class PHY_MIB {
public:

. . .

Private:

. . .

   // Added by Alonso-Rocha
   int findNode (uint32_t idNode, vect nodesPower);
   void insertNode (u_int32_t idNode, double power, vect &v);
   void initializePowers(time_power &tp);
   void previousPos(int &pos);
        void nextPos (int &pos);
   void average(time_power tp, float& averag, int &movingAway);

   // debug procedures
   void viewVector (vect v);
   void viewArray (time_power tp);
   // Alonso - Rocha ends

. . .

   // Added by Alonso - Rocha to support received power storage
       vect nodesPower;
   // int average_selected; variable that activates average mode

. . .
```

- File “*mac-802_11.cc*”

```
. . .

// Added by Alonso - Rocha to support received power tracing
#include <vector>
#include <stdlib.h>
. . .
void
Mac802_11::RetransmitRTS()
{
 . . .

   if(ssrc_ >= macmib_.getShortRetryLimit()) {
      discard(pktRTS_, DROP_MAC_RETRY_COUNT_EXCEEDED); pktRTS_ = 0;
      hdr_cmn *ch = HDR_CMN(pktTx_);
      if (ch->xmit_failure_) {

            struct hdr_mac802_11* dh = HDR_MAC802_11(pktTx_);
            //destinity dh_ra

            uint32_t idNode = ETHER_ADDR(dh->dh_ra);
            double received_Power = pktTx_->txinfo_.RxPr;

            int i = findNode(idNode, nodesPower);
            if (i != -1)
               received_Power = nodesPower[i].power[nodesPower[i].pos];
            // designed average mode
            //float av=0; int movAw=0;
            //average(nodesPower[i],av,movAw);


            if (received_Power > RxThreshold )
            {
               ch->size() -= phymib_.getHdrLen11();
               ch->xmit_reason_ = XMIT_REASON_HIGH_POWER;
               ch->xmit_failure_(pktTx_->copy(),
                     ch->xmit_failure_data_);
            }
            // Alonso - Rocha ends
            else {
               ch->size() -= phymib_.getHdrLen11();
               ch->xmit_reason_ = XMIT_REASON_RTS;
               ch->xmit_failure_(pktTx_->copy(),
               ch->xmit_failure_data_);
            }

. . .
```

```
        else {
          // Alonso - Rocha begins
          // source rf_ta; dest = rf_ra
          if (average_selected){
             u_int32_t idNode = ETHER_ADDR(rf->rf_ra);
             int pos =findNode (idNode, nodesPower);
             if (pos!=-1){
                //tell us if the node is moving away in the last 1, 2 or n movements
                float av=0; int movAw=0;
                average(nodesPower[pos],av,movAw);

                                  //these values need a deep study to improve results
                if ( (movAw >= 18) && (av < RxThreshold) ) {
                      //cout <<"LOW POWER (rts) idnode: "<<idNode<<" av: "<<av << " movaw:
"<<movAw<<" FarFactor: "<<av/RxThreshold<<" retryCount: "<<ssrc_<<endl;
                      discard(pktRTS_, DROP_MAC_RETRY_COUNT_EXCEEDED); pktRTS_ = 0;
                      hdr_cmn *ch = HDR_CMN(pktTx_);
                      if (ch->xmit_failure_) {
                         ch->size() -= phymib_.getHdrLen11();
                         ch->xmit_reason_ = XMIT_REASON_RTS;
                         ch->xmit_failure_(pktTx_->copy(),
                            ch->xmit_failure_data_);
                      }
                      discard(pktTx_, DROP_MAC_RETRY_COUNT_EXCEEDED);
                      pktTx_ = 0;
                      ssrc_ = 0;
                      rst_cw();

                      return;
                }
             }
          }
          // Alonso - Rocha ends
          inc_cw();
          mhBackoff_.start(cw_, is_idle());
       }
    }

    . . .
    void
    Mac802_11::recv(Packet *p, Handler *h)
    {

    . . .
       if(tx_active_ && hdr->error() == 0) {
          hdr->error() = 1;
       }
       // Alonso - Rocha
       //insert here the packet in our array of nodes and powers

       hdr_mac802_11 *mh = HDR_MAC802_11(p);
```

```
   //dh_ra = dest; dh_ta = source
   u_int32_t idNode = ETHER_ADDR(mh->dh_ta);
   double power = p->txinfo_.RxPr;
   insertNode (idNode,power,nodesPower);
   // Alonso – Rocha

. . .

}
```

```
void
Mac802_11::viewArray (time_power tp)
{
   int pos = tp.pos;
   cout <<"NodeId:"<<tp.idNode;
   for (int j=0;j<20;j++){
      cout <<" "<<tp.power[pos];
      previousPos(pos);
   }
   cout <<endl;
}

void
Mac802_11::previousPos(int &pos)
{
   if (pos == 0) pos = 19;
   else pos--;
}

int // returns the position in the vector nodesPower
Mac802_11::findNode (uint32_t idNode, vect nodesPower){
   int tam = nodesPower.size();
   //find in STL nops
   for (int i = 0;i<tam;i++){
      if (idNode == nodesPower[i].idNode)
         return i;
   }
   return -1;
}


void
Mac802_11::average(time_power tp, float& averag, int &movingAway)
{
   int p = tp.pos;

   double av = 0;
   double w = 100;
   double wTot = 0;
   //moving Away variable section
   int movAw=0;
   bool counting=true;

   for (int i=1;i<21;i++)
   {
      double act = tp.power[p];
      previousPos(p);
      double prev = tp.power[p];
      if (prev !=0)
      {
```

```
         double avPond = prev*w;
         //cout <<i<<" "<<avPond<<" ";
         //w = w +(100/i); //progressive
         wTot = wTot + w;
         w = w/2; //half each time
         av = av+avPond;

         //the move away section
         if ( (counting) && (i>1) ){
            //when the previous value is greather than the actual,
            //increase the moving away factor
            if (prev>act){
               movAw++;
            }else{
               counting = false;
            }
         }
      }
   }
   movingAway=movAw;
   averag = (av / wTot);
}

/*
void
Mac802_11::viewVector (vect v){ //changing to back fashion

   int tam = v.size();
   cout <<"number elements: "<<tam<<endl;
   for (int i= 0;i<tam;i++){
      viewArray(v[i]);
      cout <<endl;
   }
}
*/

void
Mac802_11::nextPos (int &pos)
{
   if (pos == 19) pos = 0;
   else pos++;// = pos + 1;
}

void
Mac802_11::initializePowers(time_power &tp){
   for (int i=0;i<20;i++){
      tp.power[i]=0;
   }
}
```

```
void
Mac802_11::insertNode (u_int32_t idNode, double power, vect &v)
{
   time_power tp;
   if (v.empty()){
      tp.pos = 0;
      initializePowers(tp);

      nextPos(tp.pos);
      tp.power[tp.pos] = power;
      tp.idNode = idNode;

      //nextPos(tp.pos);
      v.push_back(tp);
      //return 0;
   }else {

      //if not found, insert like empty vector
      //if found, recover pos in the array and insert it

      // returns the position in the vector nodesPower
      int i = findNode (idNode, v);

      if (i != -1){
         tp = v[i];
         nextPos(tp.pos);
         tp.power[tp.pos] = power;
         v[i] = tp;
      }else {
         tp.pos = 0;
         initializePowers(tp);
         nextPos(tp.pos);

         tp.power[tp.pos] = power;
         tp.idNode = idNode;

         v.push_back(tp);
      }
   }
}
```

Folder “*tcl/lib*”

- File “*ns_default.tcl*”

```
. . .

Agent/DSRAgent set dport_ 255
Agent/DSRAgent set beta_ 2              ;# ktnahm: improved robustness for DSR
Agent/DSRAgent set gamma_ 1             ;# Alonso - Rocha: cross layer feature

Agent/MIPBS set adSize_ 48

. . .
```

Folder “*trace*”

- File “*cmu-trace.cc*”

```
void
CMUTrace::format_mac_common(Packet *p, const char *why, int offset)
{

 . . .
 If (newtrace)

 . . .
  //Alonso - Rocha: before analysis
  //here it is possible to get information from the current packet
  //for example: double receivedPower = p->txinfo_.RxPr

   sprintf(pt_->buffer() + offset,
     "%c-t %.9f -Hs %d -Hd %d -Ni %d -Nx%.2f -Ny %.2f -Nz %.2f -Ne %f -Nl %3s -Nw %s ",
      op,                       // event type
      Scheduler::instance().clock(),  // time
      src_,                           // this node
      ch->next_hop_,                  // next hop
      src_,                           // this node
      x,                              // x coordinate
      y,                              // y coordinate
      z,                              // z coordinate
      energy,                         // energy, -1 = not existing
      tracename,                      // trace level
      why);                           // reason
  // here we could add some information like the received power signal
  // and write this information with the parameter %e for double values
  // that is -Npw (for example), receivedPower
  // Alonso - Rocha: before analysis. End section
```