A Comprehensible GloMoSim Tutorial

compilation by Jorge Nuevo, INRS - Université du Québec
nuevo@inrs-telecom.uquebec.ca

March 4, 2004

Abstract

The following document intends to present an easy tutorial to use and simulate wireless networks in GloMoSim, as well as the basic structure of the simulator. We have tried to gather in a consistent manner, information from several sources: white papers, On-Line Site information, README files from the software and mails from the glomosim mail-list. The software version discussed in this document is 2.03 for Unix platforms, although many concepts are in general applied to other platforms (windows 2000).

Contents

1 GloMoSim

Global Mobile Information System Simulator (GloMoSim) is a scalable simulation environment for large wireless and wireline communication networks [1]. GloMoSim uses a parallel discrete-event simulation capability provided by Parsec. [1]

GloMoSim simulates networks with up to thousand nodes linked by a heterogeneous communications capability that includes multicast, asymmetric communications using direct satellite broadcasts, multi-hop wireless communications using ad-hoc networking, and traditional Internet protocols. The following table lists the GloMoSim models currently available at each of the major layers:

Layer	*Models*
Physical (Radio Propagation)	Free space, Two-Ray
Data Link (MAC)	CSMA, MACA, TSMA, 802.11
Network (Routing)	Bellman-Ford, FSR, OSPF, DSR, WRP, LAR, AODV
Transport	TCP, UDP
Application	Telnet, FTP

The *node aggregation technique* is introduced into GloMoSim to give significant benefits to the simulation performance. Initializing each node as a separate entity inherently limits the the scalability because the memory requirements increase dramatically for a model with large number of nodes. With node aggregation, a single entity can simulate several network nodes in the system. Node aggregation technique implies that the number of nodes in the system can be increased while maintaining the same number of entities in the simulation. In GloMoSim, each entity represents a geographical area of the simulation. Hence the network nodes which a particular entity represents are determined by the physical position of the nodes [2].

1.1 Use of GloMoSim Simulator

After successfully installing GloMoSim, a simulation can be started by executing the following command in the *BIN* subdirectory.

$$./\text{glomosim} \quad < inputfile >$$

The <*input file*> contains the configuration parameters for the simulation (an example of such file is CONFIG.IN). A file called GLOMO.STAT is produced at the end of the simulation and contains all the statistics generated.

1.2 The Visualization Tool

GloMoSim has a Visualization Tool that is platform independent because it is coded in Java. To initialize the Visualization Tool, we must execute from the *java_gui* directory the following: *java GlomoMain*. This tool allows to debug and verify models and scenarios; stop, resume and step execution; show packet transmissions, show mobility groups in different colors and show statistics.

The radio layer is displayed in the Visualization Tool as follows: When a node transmits a packet, a yellow link is drawn from this node to all nodes within it's power range. As each node receives the packet, the link is erased and a green line is drawn for successful reception and a red line is drawn for unsuccessful reception. No distinction is made between different packet types (ie: control packets vs. regular pacekts, etc) [3].

[1] A C-based simulation language, developed by the Parallel Computing Laboratory at UCLA, for sequential and parallel execution of discrete-event simulation models. It can also be used as a parallel programming language

2 Setting up an Scenario

2.1 Basic Configuration File (Config.in)

The main configuration parameters for setting up an scenario are defined in the CONFIG.IN file. These parameters are the following [4]:

Parameter	*Description*
SIMULATION-TIME	Maximum simulation time. The number portion can be followed by optional letters to modify the simulation time. For example, 100NS (100 nano-seconds), 100MS (100 milli-seconds), 100S or 100 (100 seconds), 100M (100 minutes), 100H (100 hours) and 100D (100 days).
SEED	Is a random number used to initialize part of the seed of various randomly generated numbers in the simulation.
TERRAIN-DIMENSIONS	Terrain Area simulated in meters.
NUMBER-OF-NODES	Number of nodes being simulated

The **default values** of these parameters in the CONFIG.IN file are:

```
SIMULATION-TIME     15M
SEED                        1
TERRAIN-DIMENSIONS  (2000, 2000)
NUMBER-OF-NODES     30
```

Parameter	*Description*
NODE-PLACEMENT	Represents the node placement strategy.
MOBILITY	Represents the mobility model.

The NODE-PLACEMENT parameter can be assigned the following values: *RANDOM* (nodes are placed randomly within the physical terrain), *UNIFORM* (based on the number of nodes in the simulation, the physical terrain is divided into a number of cells. Within each cell, a node is placed randomly), *GRID* (Node placement starts at 0,0 and are placed in grid format with each node GRID-UNIT away from its neighbors, the number of nodes has to be square of an integer) and *FILE* (Position of nodes is read from NODE-PLACEMENT-FILE). The **default values** of these parameters in the CONFIG.IN file and an example of the NODES.INPUT file are:

```
# NODE-PLACEMENT      FILE
# NODE-PLACEMENT-FILE ./nodes.input
# NODE-PLACEMENT      GRID
# GRID-UNIT           30
# NODE-PLACEMENT      RANDOM

  NODE-PLACEMENT      UNIFORM
```

```
# Example of the NODES.IN file
# Format: nodeAddr 0 (x, y, z)
# The second parameter is for consistency with the mobility trace format

0  0 (20.2,  0.9, 0.11)
```

```
1  0 (20.3, 30.8, 0.01)
2  0 (20.4, 60.7, 0.12)
```

If the MOBILITY parameter is set to NONE then there is no movement of nodes in the model. The movement configuration for nodes is further explained at section [2.4].

Parameter	*Description*
PROPAGATION-LIMIT	Signals below this parameter (in dBm) are not delivered. This value must be smaller than RADIO-RX-SENSITIVITY + RADIO-ANTENNA-GAIN of any node in the model. Otherwise, simulation results may be incorrect. Lower value should make the simulation more precise, but it also make the execution time longer.
PROPAGATION-PATHLOSS	Specifies the pathloss model.
NOISE-FIGURE	
TEMPERATURE	temperature of the environment (in K).
RADIO-TYPE	Radio model to transmit and receive packets.
RADIO-FREQUENCY	Frequency in Hertz.
RADIO-BANDWIDTH	Bandwidth in bits per second.

The PROPAGATION-PATHLOSS parameter specifies the pathloss model. Models available in GloMoSim are FREE-SPACE -Friss free space model- which has (path loss exponent, sigma) = (2.0, 0.0). TWO-RAY -Two ray model- which uses free space path loss (2.0, 0.0) for near sight and plane earth path loss (4.0, 0.0) for far sight. The antenna height is hard-coded in the model (1.5m).

The values of the RADIO-TYPE parameter are: RADIO-ACCNOISE refers to the standard radio model, while RADIO-NONOISE refers to the abstract radio model (RADIO-NONOISE is compatible with the current version (2.1b5) of ns-2 radio model). The **default values** of these parameters in the CONFIG.IN file are:

```
   PROPAGATION-LIMIT      -111.0

#  PROPAGATION-PATHLOSS   FREE-SPACE
   PROPAGATION-PATHLOSS   TWO-RAY
#  PROPAGATION-PATHLOSS   PATHLOSS-MATRIX

   NOISE-FIGURE           10.0
   TEMPARATURE            290.0

   RADIO-TYPE             RADIO-ACCNOISE
#  RADIO-TYPE             RADIO-NONOISE

   RADIO-FREQUENCY        2.4e9
   RADIO-BANDWIDTH        2000000
```

Parameter	*Description*
RADIO-RX-TYPE	Specifies the packet reception model.
RADIO-TX-POWER	Radio transmission power (in dBm).
RADIO-ANTENNA-GAIN	Antenna Gain (in dB).
RADIO-RX-SENSITIVITY	Sensitivity of the radio (in dBm).
RADIO-RX-THRESHOLD	Minimum power for received packet (in dBm).

In the RADIO-RX-TYPE parameter, when the SNR-BOUNDED parameter is used, if the Signal to Noise Ratio (SNR) is more than RADIO-RX-SNR-THRESHOLD (in dB), it receives the signal without error. Otherwise the packet is dropped. RADIO-RX-SNR-THRESHOLD needs to be specified. The BER-BASED parameter looks up Bit Error Rate (BER) in the SNR-BER table specified by BER-TABLE-FILE. The **default values** of these parameters in the CONFIG.IN file and an example of the BER_BPSK.IN file are:

```
  RADIO-RX-TYPE           SNR-BOUNDED
  RADIO-RX-SNR-THRESHOLD  10.0
# RADIO-RX-SNR-THRESHOLD  8.49583

# RADIO-RX-TYPE           BER-BASED
# BER-TABLE-FILE          ./ber_bpsk.in

  RADIO-TX-POWER          15.0
  RADIO-ANTENNA-GAIN       0.0
  RADIO-RX-SENSITIVITY   -91.0
  RADIO-RX-THRESHOLD     -81.0
```

```
# Example of BER_BPSK.IN file:

0.000000        5.000000e-01
0.020000        4.207403e-01
0.040000        3.886487e-01
0.060000        3.645172e-01
```

Parameter	*Description*
MAC-PROTOCOL	Definition of Medium Access Protocol.
PROMISCUOUS-MODE	It is set to YES if nodes want to overhear packets destined to the neighboring node. Currently this option needs to be set to YES only for DSR. Setting it to NO may save a trivial amount of time for other protocols.
NETWORK-PROTOCOL	Definition of the Network Protocol.
ROUTING-PROTOCOL	Definition of the Routing Protocol.
APP-CONFIG-FILE	Specifies the file that sets up applications such as FTP, CBR and Telnet.

The **default values** of these and related parameters in the CONFIG.IN file are:

```
  MAC-PROTOCOL        802.11
# MAC-PROTOCOL        CSMA
# MAC-PROTOCOL        MACA

# MAC-PROTOCOL        TSMA
# TSMA-MAX-NODE-DEGREE      8

# MAC-PROPAGATION-DELAY 1000NS
# PROMISCUOUS-MODE     NO

  NETWORK-PROTOCOL    IP
  NETWORK-OUTPUT-QUEUE-SIZE-PER-PRIORITY 100
```

```
# RED-MIN-QUEUE-THRESHOLD 150
# RED-MAX-QUEUE-THRESHOLD 200
# RED-MAX-MARKING-PROBABILITY 0.1
# RED-QUEUE-WEIGHT .0001
# RED-TYPICAL-PACKET-TRANSMISSION-TIME 64000NS

  ROUTING-PROTOCOL    BELLMANFORD
# ROUTING-PROTOCOL    AODV
# ROUTING-PROTOCOL    DSR
# ROUTING-PROTOCOL    LAR1
# ROUTING-PROTOCOL    WRP
# ROUTING-PROTOCOL    FISHEYE
# ROUTING-PROTOCOL    ZRP

# ZONE-RADIUS         2

# ROUTING-PROTOCOL    STATIC
# STATIC-ROUTE-FILE   ROUTES.IN
  APP-CONFIG-FILE   ./app.conf
```

Finally, the following parameters determine if we are interested in the statistics of a single or multiple layer. By specifying the following parameters as YES, the simulation will provide us with statistics for that particular layer.

```
APPLICATION-STATISTICS          YES
TCP-STATISTICS                  NO
UDP-STATISTICS                  NO
ROUTING-STATISTICS              NO
NETWORK-LAYER-STATISTICS        NO
MAC-LAYER-STATISTICS            NO
RADIO-LAYER-STATISTICS          NO
CHANNEL-LAYER-STATISTICS        NO
MOBILITY-STATISTICS             NO
```

```
# GUI-OPTION: YES allows GloMoSim to communicate with the Java Gui Visual Tool

GUI-OPTION   YES
GUI-RADIO    YES
GUI-ROUTING  YES
```

2.2 The Application Configuration File (app.conf)

Applications such as FTP and Telnet are configured in this file. The traffic generators currently available are FTP, FTP/GENERIC, TELNET, CBR, and HTTP. FTP uses tcplib to simulate the file transfer protocol. In order to use **FTP**, the following format is needed:

```
FTP <src> <dest> <items to send> <start time>
```

where $< src >$ is the client node, $< dest >$ is the server node, $<items\ to\ send>$ is how many application layer items to send, and <start time> is when to start FTP during the simulation. If <items to send> is set to 0, FTP will use tcplib to randomly determine the amount of application layer items to send. The size of each item will always be randomly determined by tcplib. Note that the term *item* in the application layer is equivalent to the term *packet* at the network layer and *frame* at the MAC layer. Some examples are

- **FTP 0 1 10 0S**. Node 0 sends node 1 ten items at the start of the simulation, with the size of each item randomly determined by tcplib.
- **FTP 0 1 0 100S**. Node 0 sends node 1 the number of items randomly picked by tcplib after 100 seconds into the simulation. The size of each item is also randomly determined by tcplib.

FTP/GENERIC does not use tcplib to simulate file transfer. Instead, the client simply sends the data items to the server without the server sending any control information back to the client. In order to use FTP/GENERIC, the following format is needed:

```
FTP/GENERIC <src> <dest> <items to send> <item size> <start time> <end time>
```

where <src> is the client node, <dest> is the server node, <items to send> is how many application layer items to send, <item size> is size of each application layer item, <start time> is when to start FTP/GENERIC during the simulation, and <end time> is when to terminate FTP/GENERIC. If <items to send> is set to 0, FTP/GENERIC will run until the specified <end time> or until the end of the simuation, which ever comes first. If <end time> is set to 0, FTP/GENERIC will run until all <items to send> is transmitted or until the end of simulation, which ever comes first. If <items to send> and <end time> are both greater than 0, FTP/GENERIC will run until either <items to send> is done, <end time> is reached, or the simulation ends, which ever comes first. Some examples are

- **FTP/GENERIC 0 1 10 1460 0S 600S**. Node 0 sends node 1 ten items of 1460B each at the start of the simulation up to 600 seconds into the simulation. If the ten items are sent before 600 seconds elapsed, no other items are sent.
- **FTP/GENERIC 0 1 10 1460 0S 0S**. Node 0 sends node 1 ten items of 1460B each at the start of the simulation until the end of the simulation. If the ten items are sent the simulation ends, no other items are sent.
- **FTP/GENERIC 0 1 0 1460 0S 0S**. Node 0 continuously sends node 1 items of 1460B each at the start of the simulation until the end of the simulation.

TELNET uses tcplib to simulate the telnet protocol. In order to use **TELNET**, the following format is needed:

```
TELNET <src> <dest> <session duration> <start time>
```

where <src> is the client node, <dest> is the server node, <session duration> is how long the telnet session will last, <start time> is when to start TELNET during the simulation. If <session duration> is set to 0, TELNET will use tcplib to randomly determine how long the telnet session will last. The interval between telnet items are determined by tcplib. Some examples are

- **TELNET 0 1 100S 0S**. Node 0 sends node 1 telnet traffic for a duration of 100 seconds at the start of the simulation.
- **TELNET 0 1 0S 0S**. Node 0 sends node 1 telnet traffic for a duration randomly determined by tcplib at the start of the simulation.

CBR simulates a constant bit rate generator. In order to use **CBR**, the following format is needed:

```
CBR <src> <dest> <items to send> <item size> <interval> <start time> <end time>
```

where <src> is the client node, <dest> is the server node, <items to send> is how many application layer items to send, <item size> is the size of each application layer item, <interval> is the inter-departure time between the application layer items, <start time> is when to start CBR during the simulation, <end time> is when to terminate CBR during the simulation. If <items to send> is set to 0, CBR will run until the specified <end time> or until the end of the simuation, which ever comes first. If <end time> is set to 0, CBR will run until all <items to send> is transmitted or until the end of simulation, which ever comes first. If <items to send> and <end time> are both greater than 0, CBR will run until either <items to send> is done, <end time> is reached, or the simulation ends, which ever comes first. Some examples are

- **CBR 0 1 10 1460 1S 0S 600S**. Node 0 sends node 1 ten items of 1460B each at the start of the simulation up to 600 seconds into the simulation. The inter-departure time for each item is 1 second. If the ten items are sent before 600 seconds elapsed, no other items are sent.
- **CBR 0 1 0 1460 1S 0S 600S**. Node 0 continuously sends node 1 items of 1460B each at the start of the simulation up to 600 seconds into the simulation. The inter-departure time for each item is 1 second.
- **CBR 0 1 0 1460 1S 0S 0S**. Node 0 continuously sends node 1 items of 1460B each at the start of the simulation up to the end of the simulation. The inter-departure time for each item is 1 second.

HTTP simulates single-TCP connection web servers and clients. The following format describes its use for servers:

```
HTTPD <address>
```

where <address> is the node address of a node which will be serving Web pages. For HTTP clients, the following format is used:

```
HTTP <address> <num_of_server> <server_1> ... <server_n> <start> <thresh>
```

where <address> is the node address of the node on which this client resides, <num_of_server> is the number of server addresses which will follow <server_1>, <server_n> are the node addresses of the servers which this client will choose between when requesting pages. There must be HTTPD <address> lines existing separately for each of these addresses; <start> is the start time for when the client will begin requesting pages <thresh> is a ceiling (specified in units of time) on the amount of *think time* that will be allowed for a client. The network-trace based amount of time modulo this threshhold is used to determine think time. An example is

```
HTTPD 2
HTTPD 5
HTTPD 8
HTTPD 11
HTTP 1 3 2 5 11 10S 120S
```

There are HTTP servers on nodes 2, 5, 8, and 11. There is an HTTP client on node 1. This client chooses between servers [2, 5, 11] only when requesting web pages. It begins browsing after 10S of simulation time have passed, and will *think* (remain idle) for at most 2 minutes of simulation time, at a time.

2.3 Setting up the Transmission Range

Because of the way radio transmissions are affected by the environment in such a complex way, it is quite difficult to predict the comportment of a system and to define a radio transmission range of a node. The radio range is the average maximum distance in usual operating conditions between two nodes. There is no standard and common operating procedure to measure a range (except in free space, which is useless), so

we can't really compare different products from the ranges as indicated in the mobile devices data-sheets. If we want to compare mobile nodes in term of range performance, we must look closely at the *transmitted power* and *sensitivity values*. These are some measurable characteristics of the hardware which indicate the performance of the products in that respect. The **transmitted power** is the strength of the emissions measured in Watts (or milliWatts). Government regulations limit this power, but also having a high transmit power will also be likely to drain the batteries faster. Nevertheless, having a high transmit power will help to emit signals stronger than the interferers in the band. The **sensitivity** is the measure of the weakest signal that may be reliably heard on the channel by the receiver (it is able to read the bits from the antenna with a low error probability). This indicates the performance of the receiver, and the lower the value the better the hardware. Usual values are around -80 dBm (the lowest, the better, for example -90 dBm is better). A possible methodology to determine the transmission radio range in GloMoSim would be the following:

1. Set the *propagation pathloss model* (PROPAGATION-PATHLOSS parameter).
2. Fix de received power of the destination antenna (RADIO-RX-THRESHOLD parameter).
3. Fix the distance and calculate the transmitted power according to the selected propagation pathloss model.
4. Set this value to the RADIO-TX-POWER parameter.

GloMoSim has the following propagation models: free space and the Ground Reflection (or Two-Ray) models. The **free space propagation model**,is used to predict received signal strength when the transmitter and receiver have a clear, unobstructed line-of-sight between them. This model predicts that transmission power is attenuated in proportion to the square of the distance. According to this model, the *Friis Free Space Equation* for non-isotropic antennas is the following:

$$P_r = P_t(\frac{\lambda}{4\pi d})^n G_t G_r \tag{1}$$

where P_r is the received power, P_t is the transmitted power (in Watts or milli-Watts), λ is the carrier wavelength (in meters), d is the distance between transmitter and receiver (in meters), n is the path loss coefficient, G_t is the antenna gain at the transmitter and G_r is the antenna gain at the receiver (adimensional). For the *Ideal Isotropic Antenna*, the free space loss equation is:

$$\frac{P_t}{P_r} = \frac{(4\pi d)^2}{\lambda^2} = \frac{(4\pi f d)^2}{c^2} \tag{2}$$

where c is the speed of light (3 x 10^8 m/s) and f is the frequency (in hertz or $\frac{1}{s}$). The **Ground Reflection (Two-Ray) Model** considers both the direct path and a ground reflected propagation path between transmitter and receiver. This model predicts that received power falls off with distance raised to the fourth power, or at a rate of 40 dB/decade. This is a much more rapid path loss than is experienced in free space [5]:

$$P_r = P_t \frac{h_t^2 h_r^2}{d^4} G_t G_r \tag{3}$$

where h_t and h_r is the height of the transmitter and receiver antennas, and in GloMoSim this value is hard-coded to 1.5 meters. This model is represented at Fig. [1]. The **Antenna gain** (represented in GloMoSim by the RADIO-ANTENNA-GAIN parameter) is a measure of the directionality of an antenna. Antenna gain is defined as the power output, in a particular direction, compared to that produced in any direction by a perfect omnidirectional antenna (isotropic antenna). For example, if an antenna has a gain

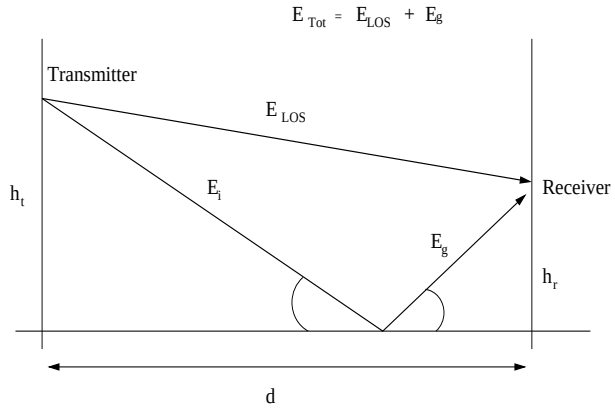

Figure 1: Ground Reflection Model

of 3 dB, that antenna improves upon the isotropic antenna in that direction by 3 dB, or a factor of 2. The increased power radiated in a given direction is at the expense of other directions [6].

In both models, we have to measure the loss or attenuation of signal strength. The *decibel* is a measure of the ratio between two signal levels. the decibel gain is given by the following equation:

$$G_{dB} = 10 \quad log_{10} \frac{P_{out}}{P_{in}} \tag{4}$$

There is some inconsistency in the literature over the use of the terms *gain* and *loss*. If the value of G_{dB} is positive, this represents an actual gain in power. For example, a gain of 3 dB means that the power has doubled. If the value of G_{dB} is negative, this represents an actual loss in power. For example, a gain of -3 dB means that the power has halved, and this is a loss of power. Normally, this is expressed by saying there is a loss of 3 dB. However, some of the literature would say that this is a loss of -3 dB. It makes more sense to say that a negative gain corresponds to a positive loss [6]. Decibel values refer to relative magnitudes or changes in magnitude, not to an absolute level. It is convenient to be able to refer to an absolute level of power in decibels so that gains and losses with reference to an initial signal level may be calculated easily. The *dBW (decibel Watt)* is used to extensively in microwave applications. The value of 1 W is selected as a reference and defined to be 0 dbW. Another common unit is the *dBm (decibel-milliWatt)*, which uses 1 mW as the reference. Thus 0 dbM = 1 mW. This is represented in the following formulae:

$$Power_{dbW} = 10 \quad log \frac{Power_W}{1W} \qquad Power_W = 10^{\frac{Power_{dBW}}{10}} \tag{5}$$

$$Power_{dbm} = 10 \quad log \frac{Power_{mW}}{1mW} \qquad Power_{mW} = 10^{\frac{Power_{dBm}}{10}} \tag{6}$$

Some examples of theoretical calculation of radio-range distance, according to the free space and two-ray models are the following. Let's consider the following parameters:

```
I) Example 1.
             PROPAGATION-PATHLOSS =  FREE-SPACE
             PROPAGATION-LIMIT    =  -111 (dBm)
             RADIO-FREQUENCY      =   2.4 e 9 (hertz)
             RADIO-TX-POWER     =    15 (dBm)
             RADIO-RX-THRESHOLD   =   -81 (dBm)
             RADIO-ANTENNA-GAIN   =   0.0 (dBm)
```

Thus we use the ideal isotropic antenna free space loss (RADIO-ANTENNA-GAIN=0.0 indicates it is an isotropic antenna). We transform P_t=15 dBm and P_r=-81 dBm to mW, and obtain the distance:

$$Power_{mW} = 10^{\frac{Power_{dBW}}{10}} = 10^{1.5} = 31.622776mW = P_t$$
$$Power_{mW} = 10^{\frac{Power_{dBW}}{10}} = 10^{-8.1} = 7x10^{-9}mW = P_r$$
$$d = \frac{\sqrt{\frac{P_t}{P_r}c^2}}{4\pi f} = d = \frac{\sqrt{\frac{31.622776}{7x10^{-9}}(3x10^8)^2}}{4\pi(2.4x10^9)} = 668.57m$$

```
II) Example 2.
             PROPAGATION-PATHLOSS =   TWO-RAY
             PROPAGATION-LIMIT    =  -111 (dBm)
             RADIO-FREQUENCY      =   2.4 e 9 (hertz)
             RADIO-TX-POWER       =    15 (dBm)
             RADIO-RX-THRESHOLD   =   -81 (dBm)
             RADIO-ANTENNA-GAIN   =   0.0 (dBm)
```

The propagation pathloss model indicates us to use the Two-Ray model formula. $G_t = G_r = 0$ dBm = 1 (adimensional, relative to an isotropic antenna), thus:

$$P_r = P_t\frac{h_t^2h_r^2}{d^4}G_tG_r$$
$$d = \sqrt[4]{\frac{P_th_t^2h_r^2}{P_r}} = \sqrt[4]{\frac{(31.6227mW)(1.5)^2(1.5)^2}{7x10^{-9}}} = 388m$$

```
III) Example 3.
             PROPAGATION-PATHLOSS =   TWO-RAY
             PROPAGATION-LIMIT    =  -111 (dBm)
             RADIO-FREQUENCY      =   2.4 e 9 (hertz)
             RADIO-TX-POWER       =    15 (dBm)
             RADIO-RX-THRESHOLD   =   -81 (dBm)
             RADIO-ANTENNA-GAIN   =   10.0 (dBm)
```

It is exactly the previous problem but with a RADIO-ANTENNA-GAIN parameter of 10.0 dBm. We transform it to an adimensional value comparing to an isotropic antenna.

$$G_{mW} = 10^{\frac{G_{dBm}}{10}} = 10mW = 10$$
$$d = \sqrt[4]{\frac{P_tG_tG_rh_t^2h_r^2}{P_r}} = \sqrt[4]{\frac{(31.6227mW)(10)(10)(1.5)^2(1.5)^2}{7x10^{-9}}} = 1229.749m$$

In the /BIN directory, program *radio_range* obtains the radio range of the configuration file. For example, for the default CONFIG.IN file we have:

```
$>./radio_range config.in

radio range: 376.782m
Execution time :   0.0281 sec
Number of messages processed : 0
Number of context switches occurred : 6
Number of Local NULL messages sent : 0
Number of Remote NULL messages sent : 0
Total Number of NULL messages sent : 0
```

Executing this program with different radio configuration parameters, we obtain the following results. Although we can change the radio range by varying the transmitter power (RADIO-TX-POWER) or the receiver power (RADIO-RX-THRESHOLD), it is somehow advisable to change the transmitter power, because the receiver power depends of the radio environment while we can control the transmitter power.

Radio Parameters					
PROPAGATION-LIMIT (dBm)	-111	-111	-111	-111	
PROPAGATION-PATHLOSS	Free-space	Two-Ray	Two-Ray	Two-Ray	
RADIO-FREQUENCY (hz)	2.4e9	2.4e9	2.4e9	2.4e9	
RADIO-TX-POWER (dBm)	15	15	15	15	
RADIO-ANTENNA-GAIN (dBm)	0	0	10	1	
RADIO-RX-SENSITIVITY (dBm)	-91	-91	-91	-91	
RADIO-RX-THRESHOLD (dBm)	-81	-81	-81	-81	
Radio Range (m)	627.625	376.782	1191.422	422.757	
Radio Parameters					
PROPAGATION-LIMIT (dBm)	-111	-111	-111	-111	-111
PROPAGATION-PATHLOSS	Two-Ray	Two-Ray	Two-Ray	Two-Ray	Two-Ray
RADIO-FREQUENCY (hz)	2.4e9	2.4e9	2.4e9	2.4e9	2.4e9
RADIO-TX-POWER (dBm)	15	1	-10	15	15
RADIO-ANTENNA-GAIN (dBm)	0	0	0	0	0
RADIO-RX-SENSITIVITY (dBm)	-61	-91	-91	-91	-91
RADIO-RX-THRESHOLD (dBm)	-81	-81	-81	-61	-101
Radio Range (m)	376.782	125.227	35.293	62.762	1191.492

2.4 Configuring Mobility

The only available mobility model in GloMoSim v2.03 is the *Random Waypoint Mobility Model (RWPM)* [7]. In this model a node randomly selects a destination from the physical terrain, and moves in the direction of that destination in a speed uniformly chosen between MOBILITY-WP-MIN-SPEED and MOBILITY-WP-MAX-SPEED parameters (defined in meter/sec). After it reaches its destination, the node stays there for a MOBILITY-WP-PAUSE time period.

If we want to use mobility patterns other than RWPM, then we must specify the parameter MOBILITY TRACE in order to indicate GloMoSim that individual movements for nodes will be taken from a file specified by MOBILITY-TRACE-FILE. The MOBILITY-INTERVAL parameter is used to indicate nodes to update their position every MOBILITY-INTERVAL time period, while MOBILITY-D-UPDATE is used when a node updates its position based on the distance (in meters). The philosophy in GloMoSim is somehow different to NS-2 [8]. In NS-2, we indicate the destination point and a constant velocity, with this the node arrives at a certain time. In GloMoSim we specify the destination point and the time the node will arrive there, the tool determines the constant velocity to do this. The mobility parameters defined in the CONFIG.IN file are:

```
MOBILITY  NONE

# MOBILITY RANDOM-WAYPOINT
# MOBILITY-WP-PAUSE       30S
# MOBILITY-WP-MIN-SPEED   0
# MOBILITY-WP-MAX-SPEED   10

# MOBILITY TRACE
```

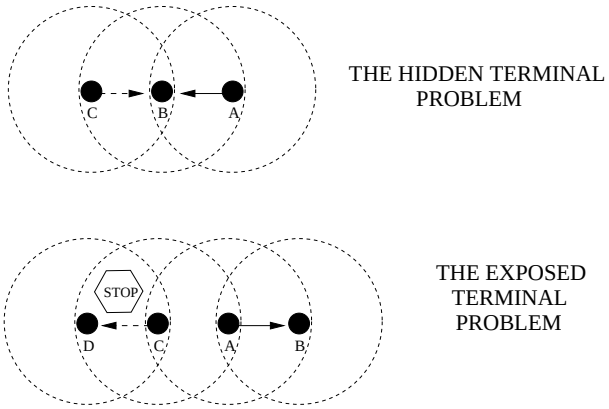

Figure 2: The Hidden and Exposed Terminal Problems

```
# MOBILITY-TRACE-FILE ./mobility.in

# MOBILITY PATHLOSS-MATRIX

# The following parameter is necessary for
# all  mobility models

  MOBILITY-POSITION-GRANULARITY 0.5
```

```
# Example of MOBILITY.IN file
# mobility trace format: node-address simclock destination(x y z)
# All lines for a node must be sorted in time increasing order.

10 100S (200.0, 150.0, 0.2)
10 200S (200.0, 150.0, 0.2)
10 500S (250.0, 250.0, 0.2)
```

3 Some Simulation Examples

3.1 MAC protocols Simulation

Glomosim has several MAC protocols to be used in a simulation, such as CSMA, MACA and 802.11. In *CSMA (Carrier Sense Multiple Access)* protocol, a station wishing to transmit, first listen to the medium in order to determine if another transmission is in progress (carrier sense). If the transmission medium is in use, the station waits, otherwise it may transmit. Unfortunately, CSMA is limited by two interference mechanisms: the hidden and the exposed terminal problems. The *hidden terminal* problem occurs because the radio network, as opposed to other networks, such as a LAN, does not guarantee high degree of connectivity. Thus, two nodes, which maintain connectivity to a third node, do not necessarily, can hear each other. In Fig. [2] node A is in communication with node B where A is currently transmitting. Node C wishes to communicate with node B as well. Following the CSMA protocol, node C listens to the medium, but since C does not detect node's A transmission, it declares the medium free. Consequently, C accesses the medium, causing collisions at B.

The second problem also shown at Fig. [2], the *exposed terminal* occurs when for example, node A is transmitting to node B, while node C wants to transmit D. Following the CSMA protocol, node C listens to the medium, hears that node A transmits and defers from accessing the medium. However, there is no

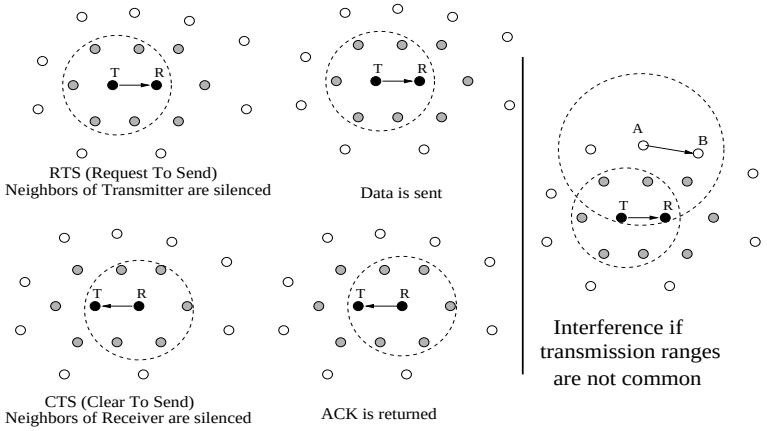

Figure 3: The RTS/CTS dialogue

reason why node C cannot transmit concurrently with the transmission of node A, as the transmission of node C would not interfere with the reception at node B due to the distance between the two. The point here is, again, the fact that the collisions occur at the receiver, while the CSMA protocol checks the status of the medium at the transmitter.

In general, the hidden terminal problem reduces the capacity of a network due to increasing the number of collisions, while the exposed terminal problem reduces the network capacity due to the unnecessarily deferring nodes from transmitting. Several attempts have been made in the literature to reduce the effect of these two problems. The necessity of a dialogue between the transmitting and the receiving nodes that preempts the actual transmission and that is referred to as the *RTS/CTS dialogue*, has generally accepted. In the RTS/CTS dialogue example (Fig. [3]), a node ready to transmit a packet, send a short control packet, the *Request To Send (RTS)*, with all nodes that hear the RTS defer from accessing the channel for the duration of the RTS/CTS dialogue. The destination, upon reception of the RTS responds with another short control packet, the *Clear To Send (CTS)* [9].

All nodes that hear the CTS packet defer from accessing the channel for the duration of the data packet transmission. The reception of the CTS packet at the transmitting node acknowledges that the RTS/CTS dialogue has been successful and the node starts the transmission of the actual data packet. In the event that two nodes send simultaneous RTS frames to the same node, the RTS transmissions collide and are lost. If this occurs, the nodes which sent the unsuccessful RTS packets set a random timer utilizing the binary exponential backoff algorithm for the next transmission attempt. Although the RTS/CTS dialogue does not eliminate the *hidden* and the *expose terminal* problems, it does provide some degree of improvement over the traditional *CSMA schemes*. Nevertheless, the RTS/CTS dialogue can fail when nodes have different transmission ranges.

The *MACA (Multiple Access Collision Avoidance)* and the *802.11* protocols use the described RTS/CTS dialogue for collision avoidance on the shared channel. Where IEEE 802.11 differs however, is in its requirement of an acknowledgment (ACK) transmission by the receiver after the successful reception of the data packet. The inclusion of the ACK allows immediate retransmission if necessary by verifying that the data packet was successfully received.

3.1.1 Example 1: The Hidden Terminal Problem

In the following example, three nodes are within communication range. Node 0 starts sending data packets to node 1 after node 2 starts sending data packets to node 1. The configuration parameters for this scenario are the following:

```
In nodes.input file:
                    0  0 (50,  1000, 0.0)
                    1  0 (650, 1000, 0.0)
                    2  0 (1250, 1000, 0.0)

so the simulation model is:
                Node0----600m----Node1----600m----Node2

In config.in file:
                    MOBILITY   =   NONE
                    SIMULATION-TIME  =   5S
                    PROPAGATION-PATHLOSS = FREE-SPACE
                    RADIO-TX-POWER  =  15dBm
                    RADIO-RX-SENSITIVITY =  -81 dBm
                    RADIO-RX-THRESHOLD  =  -81dBm
                    ANTENNA_GAIN  =  0dB
                    MAC_PROTOCOL  =  CSMA
```

If we let that only node 2 sends data packets to node 1, while node 0 remains silence, we obtain the following output in the glomo.stat file.

```
In app.conf:    CBR 2  1   0    512   4MS  0  0

Node: 1, Layer:         MacCSMA, Number of UNICAST packets received clearly: 1250
Node: 1, Layer:    AppCbrServer, (0) Total number of bytes received: 640000
Node: 1, Layer:    AppCbrServer, (0) Total number of packets received: 1250
Node: 2, Layer:         MacCSMA, Number of UNICAST packets output to the channel: 1250
Node: 2, Layer:    AppCbrClient, (0) Total number of bytes sent: 640000
Node: 2, Layer:    AppCbrClient, (0) Total number of packets sent: 1250
```

The number of packets to be sent is (5S - 0)/ 4EXP(-3) S = 1250. That is, all packets that are sent by node 2 are clearly received by node 1. The same occurs if we let node 0 to transmit alone, but if we let both nodes 0 and 2 to transmit at the same time to node 1, we have:

```
In app.conf:    CBR 0  1   0    512   4MS  6MS  0
                CBR 2  1   0    512   4MS  0  0

Node: 0, Layer:         MacCSMA, Number of UNICAST packets output to the channel: 1248
Node: 0, Layer:    AppCbrClient, (0) Total number of bytes sent: 639488
Node: 0, Layer:    AppCbrClient, (0) Total number of packets sent: 1249
Node: 1, Layer:         MacCSMA, Number of UNICAST packets received clearly: 1605
Node: 1, Layer:    AppCbrServer, (0) Client address: 0
Node: 1, Layer:    AppCbrServer, (0) Total number of bytes received: 410624
Node: 1, Layer:    AppCbrServer, (0) Total number of packets received: 802
Node: 1, Layer:    AppCbrServer, (0) Client address: 2
Node: 1, Layer:    AppCbrServer, (0) Total number of bytes received: 411136
Node: 1, Layer:    AppCbrServer, (0) Total number of packets received: 803
Node: 2, Layer:         MacCSMA, Number of UNICAST packets output to the channel: 1250
Node: 2, Layer:    AppCbrClient, (0) Total number of bytes sent: 640000
Node: 2, Layer:    AppCbrClient, (0) Total number of packets sent: 1250
```

which shows that nodes 0 sends 1248 packets and node 2 sends 1250 packets, but node 1 is receiving only 802 packets from node 0 and 803 packets from node 2 due to collisions. If we have the same scenario but we change to MACA protocol instead, we have the following results:

```
In app.conf:    CBR 0  1   0    512   4MS  6MS  0
                CBR 2  1   0    512   4MS  0  0

Node: 0, Layer:         MacMACA, Number of UNICAST packets output to the channel: 162
Node: 0, Layer:         MacMACA, Number of RTS Packets sent: 4779
Node: 0, Layer:         MacMACA, Number of CTS Packets sent: 1
Node: 0, Layer:         MacMACA, Number of RTS Packets got: 1
Node: 0, Layer:         MacMACA, Number of CTS Packets got: 162
Node: 0, Layer:         MacMACA, Number of Noisy Packets got: 135
Node: 0, Layer:    AppCbrClient, (0) Server address: 1
Node: 0, Layer:    AppCbrClient, (0) Total number of bytes sent: 639488
Node: 0, Layer:    AppCbrClient, (0) Total number of packets sent: 1249
Node: 1, Layer:         MacMACA, Number of UNICAST packets received clearly: 281
Node: 1, Layer:         MacMACA, Number of RTS Packets sent: 40
Node: 1, Layer:         MacMACA, Number of CTS Packets sent: 2055
Node: 1, Layer:         MacMACA, Number of RTS Packets got: 2055
Node: 1, Layer:         MacMACA, Number of CTS Packets got: 2
Node: 1, Layer:    AppCbrServer, (0) Client address: 0
Node: 1, Layer:    AppCbrServer, (0) Total number of bytes received: 75264
Node: 1, Layer:    AppCbrServer, (0) Total number of packets received: 147
Node: 1, Layer:    AppCbrServer, (0) Client address: 2
Node: 1, Layer:    AppCbrServer, (0) Total number of bytes received: 68608
Node: 1, Layer:    AppCbrServer, (0) Total number of packets received: 134
Node: 2, Layer:         MacMACA, Number of UNICAST packets output to the channel: 144
Node: 2, Layer:         MacMACA, Number of RTS Packets sent: 4800
Node: 2, Layer:         MacMACA, Number of CTS Packets sent: 1
Node: 2, Layer:         MacMACA, Number of RTS Packets got: 1
Node: 2, Layer:         MacMACA, Number of CTS Packets got: 144
Node: 2, Layer:         MacMACA, Number of Noisy Packets got: 149
Node: 2, Layer:    AppCbrClient, (0) Total number of bytes sent: 640000
Node: 2, Layer:    AppCbrClient, (0) Total number of packets sent: 1250
```

And if we change to IEEE 802.11 MAC protocol, we obtain:

```
In app.conf:    CBR 0  1   0    512   4MS  6MS  0
                CBR 2  1   0    512   4MS  0  0

Node: 0, Layer:          802.11, UCAST (non-frag) pkts sent to chanl: 701
Node: 0, Layer:          802.11, retx pkts due to CTS timeout: 44
Node: 0, Layer:    AppCbrClient, (0) Total number of bytes sent: 639488
Node: 0, Layer:    AppCbrClient, (0) Total number of packets sent: 1249
Node: 1, Layer:          802.11, UCAST pkts rcvd clearly: 1425
Node: 1, Layer:    AppCbrServer, (0) Client address: 0
Node: 1, Layer:    AppCbrServer, (0) Total number of bytes received: 358912
Node: 1, Layer:    AppCbrServer, (0) Total number of packets received: 701
Node: 1, Layer:    AppCbrServer, (0) Client address: 2
Node: 1, Layer:    AppCbrServer, (0) Total number of bytes received: 370688
Node: 1, Layer:    AppCbrServer, (0) Total number of packets received: 724
Node: 2, Layer:          802.11, UCAST (non-frag) pkts sent to chanl: 724
Node: 2, Layer:          802.11, retx pkts due to CTS timeout: 44
Node: 2, Layer:    AppCbrClient, (0) Total number of bytes sent: 640000
Node: 2, Layer:    AppCbrClient, (0) Total number of packets sent: 1250
```

Which shows that a better behavior is obtained when using CSMA instead of MACA because of the RTS/CTS messages. The use of RTS packets whenever a source has a data packet to send without first sensing the channel, results in an increase in packet collisions and hence decreased throughput. The collision avoidance mechanism incorporated into IEEE 802.11 for the transmission of fRTS packets aids in the reduction of the number of collisions. Consequently, more data packets reach their destinations. These results are also obtained and confirmed in [10].

3.1.2 Example 2: The Exposed Terminal Problem

In the following example, four nodes are within communication range. Node 1 starts sending data packets to node 0 after node 2 starts sending data packets to node 3. We establish the packet inter-departure time very small, so that when node 1 wants to transmit, it senses the medium busy. The configuration parameters for this scenario are the following:

```
In nodes.input file:
                    0  0 (50,  1000, 0.0)
                    1  0 (650, 1000, 0.0)
                    2  0 (1250, 1000, 0.0)
                    3  0 (1850, 1000, 0.0)

so the simulation model is:
            Node0----600m----Node1----600m----Node2----600m----Node3

In config.in file:
                    MOBILITY   =  NONE
                    SIMULATION-TIME  =   5S
                    PROPAGATION-PATHLOSS = FREE-SPACE
                    RADIO-TX-POWER  =  15dBm
                    RADIO-RX-SENSITIVITY =  -81 dBm
                    RADIO-RX-THRESHOLD  =  -81dBm
                    ANTENNA_GAIN  =  0dB
                    MAC_PROTOCOL  =  CSMA
```

If we let that only node 2 sends data packets to node 3, while node 1 remains silence, we obtain the following output in the glomo.stat file.

```
In app.conf:    CBR 2  3   0    512   4MS  0  0

In glomo.stat:
Node: 2, Layer:         MacCSMA, Number of UNICAST packets output to the channel: 1250
Node: 2, Layer:         MacCSMA, Number of BROADCAST packets output to the channel: 1
Node: 2, Layer:    AppCbrClient, (0) Total number of bytes sent: 640000
Node: 2, Layer:    AppCbrClient, (0) Total number of packets sent: 1250
Node: 3, Layer:         MacCSMA, Number of UNICAST packets received clearly: 1250
Node: 3, Layer:         MacCSMA, Number of BROADCAST packets received clearly: 1
Node: 3, Layer:    AppCbrServer, (0) Total number of bytes received: 640000
Node: 3, Layer:    AppCbrServer, (0) Total number of packets received: 1250
```

The number of packets to be sent is (5S - 0)/ 4EXP(-3) S = 1250. That is, all packets that are sent by node 2 are clearly received by node 3. The same occurs if we let node 1 to transmit alone, but if we let both nodes 1 and 2 to transmit at the same time, we have:

```
In app.conf:
```

```
               CBR 1  0   0    512   4MS  4MS  0
               CBR 2  3   0    512   4MS  0  0

In glomo.stat:
Node: 0, Layer:         MacCSMA, Number of UNICAST packets received clearly: 0
Node: 0, Layer:         MacCSMA, Number of BROADCAST packets received clearly: 0
Node: 1, Layer:         MacCSMA, Number of UNICAST packets output to the channel: 0
Node: 1, Layer:         MacCSMA, Number of BROADCAST packets output to the channel: 6
Node: 1, Layer:         MacCSMA, Number of UNICAST packets received clearly: 0
Node: 1, Layer:         MacCSMA, Number of BROADCAST packets received clearly: 1
Node: 1, Layer:    AppCbrClient, (0) Total number of bytes sent: 639488
Node: 1, Layer:    AppCbrClient, (0) Total number of packets sent: 1249
Node: 2, Layer:         MacCSMA, Number of UNICAST packets output to the channel: 1250
Node: 2, Layer:         MacCSMA, Number of BROADCAST packets output to the channel: 1
Node: 2, Layer:    AppCbrClient, (0) Total number of packets sent: 1250
Node: 2, Layer:    AppCbrClient, (0) Throughput (bits per second): 1024000
Node: 3, Layer:         MacCSMA, Number of UNICAST packets received clearly: 1244
Node: 3, Layer:         MacCSMA, Number of BROADCAST packets received clearly: 1
Node: 3, Layer:    AppCbrServer, (0) Total number of bytes received: 636928
Node: 3, Layer:    AppCbrServer, (0) Total number of packets received: 1244
```

which represents node 2 transmitting to node 3 and all packets are received clearly by 3, node 1 starts transmitting 4 MS after start of simulation but does not sense any carrier (although the application layer has 1249 packets to transmit) and the MAC layer does not transmit almost any packet to the channel. If we change the MAC protocol in the config.in file to MACA and let that only node 2 sends packets to node 3, we obtain the following glomo.stat file:

```
In app.conf:
               CBR 2  3   0    512   4MS  0  0

Node: 2, Layer:         MacMACA, Number of UNICAST packets output to the channel: 1220
Node: 2, Layer:         MacMACA, Number of RTS Packets sent: 1220
Node: 2, Layer:         MacMACA, Number of CTS Packets sent: 1
Node: 2, Layer:         MacMACA, Number of RTS Packets got: 1
Node: 2, Layer:         MacMACA, Number of CTS Packets got: 1220
Node: 2, Layer:    AppCbrClient, (0) Total number of bytes sent: 624640
Node: 2, Layer:    AppCbrClient, (0) Total number of packets sent: 1220
Node: 3, Layer:         MacMACA, Number of UNICAST packets received clearly: 1219
Node: 3, Layer:         MacMACA, Number of RTS Packets sent: 1
Node: 3, Layer:         MacMACA, Number of CTS Packets sent: 1220
Node: 3, Layer:         MacMACA, Number of RTS Packets got: 1220
Node: 3, Layer:         MacMACA, Number of CTS Packets got: 1
```

Which shows that all packets sent by node 2 are clearly received by node 3. If we let node 1 transmit to node 0 and node 2 transmit to node 3 simultaneously, then we obtain the following glomo.stat file:

```
In app.conf:
               CBR 1  0   0    512   4MS  4MS  0
               CBR 2  3   0    512   4MS  0  0

Node: 0, Layer:         MacMACA, Number of UNICAST packets received clearly: 41
Node: 0, Layer:         MacMACA, Number of BROADCAST packets received clearly: 1
Node: 0, Layer:         MacMACA, Number of RTS Packets sent: 64
```

```
Node: 0, Layer:         MacMACA, Number of CTS Packets sent: 3162
Node: 0, Layer:         MacMACA, Number of RTS Packets got: 3162
Node: 0, Layer:         MacMACA, Number of CTS Packets got: 1
Node: 0, Layer:    AppCbrServer, (0) Total number of bytes received: 20992
Node: 0, Layer:    AppCbrServer, (0) Total number of packets received: 41
Node: 1, Layer:         MacMACA, Number of UNICAST packets output to the channel: 721
Node: 1, Layer:         MacMACA, Number of BROADCAST packets output to the channel: 1
Node: 1, Layer:         MacMACA, Number of UNICAST packets received clearly: 1
Node: 1, Layer:         MacMACA, Number of BROADCAST packets received clearly: 1
Node: 1, Layer:         MacMACA, Number of RTS Packets sent: 3842
Node: 1, Layer:         MacMACA, Number of CTS Packets sent: 21
Node: 1, Layer:         MacMACA, Number of RTS Packets got: 21
Node: 1, Layer:         MacMACA, Number of CTS Packets got: 721
Node: 1, Layer:         MacMACA, Number of Noisy Packets got: 65
Node: 1, Layer:    AppCbrClient, (0) Total number of bytes sent: 639488
Node: 1, Layer:    AppCbrClient, (0) Total number of packets sent: 1249
Node: 2, Layer:         MacMACA, Number of UNICAST packets output to the channel: 770
Node: 2, Layer:         MacMACA, Number of BROADCAST packets output to the channel: 1
Node: 2, Layer:         MacMACA, Number of UNICAST packets received clearly: 1
Node: 2, Layer:         MacMACA, Number of BROADCAST packets received clearly: 0
Node: 2, Layer:         MacMACA, Number of RTS Packets sent: 3837
Node: 2, Layer:         MacMACA, Number of CTS Packets sent: 1
Node: 2, Layer:         MacMACA, Number of RTS Packets got: 1
Node: 2, Layer:         MacMACA, Number of CTS Packets got: 770
Node: 2, Layer:         MacMACA, Number of Noisy Packets got: 42
Node: 2, Layer:    AppCbrClient, (0) Total number of bytes sent: 640000
Node: 2, Layer:    AppCbrClient, (0) Total number of packets sent: 1250
Node: 3, Layer:         MacMACA, Number of UNICAST packets output to the channel: 1
Node: 3, Layer:         MacMACA, Number of BROADCAST packets output to the channel: 0
Node: 3, Layer:         MacMACA, Number of UNICAST packets received clearly: 52
Node: 3, Layer:         MacMACA, Number of BROADCAST packets received clearly: 1
Node: 3, Layer:         MacMACA, Number of RTS Packets sent: 1
Node: 3, Layer:         MacMACA, Number of CTS Packets sent: 3179
Node: 3, Layer:         MacMACA, Number of RTS Packets got: 3179
Node: 3, Layer:         MacMACA, Number of CTS Packets got: 1
Node: 3, Layer:         MacMACA, Number of Noisy Packets got: 0
Node: 3, Layer:    AppCbrServer, (0) Total number of bytes received: 26624
Node: 3, Layer:    AppCbrServer, (0) Total number of packets received: 52
```

which shows that in an exposed-terminal scenario, both CSMA and MACA protocols present a poor performance behavior.

3.2 Results Analysis

The data generated in previous examples was small, allowing to do a direct analysis from the *glomo.stat* file. Nevertheless, in most cases this is not possible due to the size of the network and the duration of the simulation time. In these cases we use other tools to do an analysis of the *glomo.stat* file. For example, the following shell program

```
# ./analy-datpak1.sh <file>
```

```
cat $1 | grep AppCbrServer | grep Total | grep packets |  grep received
```

obtains the number of data packets (at the application layer -or AppCbrServer) received by their destinations. An example of it when executed with the *glomo.stat* is the following:

```
./analy-datpak1.sh cmpmac-802aodv-200.stat
Node:      3, Layer:    AppCbrServer, (0) Total number of packets received: 404
Node:      5, Layer:    AppCbrServer, (0) Total number of packets received: 1115
Node:      7, Layer:    AppCbrServer, (0) Total number of packets received: 622
Node:      9, Layer:    AppCbrServer, (0) Total number of packets received: 260
Node:     62, Layer:    AppCbrServer, (0) Total number of packets received: 360
Node:     65, Layer:    AppCbrServer, (0) Total number of packets received: 338
Node:     67, Layer:    AppCbrServer, (0) Total number of packets received: 779
Node:     69, Layer:    AppCbrServer, (1) Total number of packets received: 1011
Node:     70, Layer:    AppCbrServer, (0) Total number of packets received: 922
Node:     73, Layer:    AppCbrServer, (0) Total number of packets received: 1079
Node:     76, Layer:    AppCbrServer, (0) Total number of packets received: 935
Node:     79, Layer:    AppCbrServer, (0) Total number of packets received: 939
Node:     81, Layer:    AppCbrServer, (0) Total number of packets received: 647
Node:     83, Layer:    AppCbrServer, (0) Total number of packets received: 632
Node:     85, Layer:    AppCbrServer, (0) Total number of packets received: 1184
Node:     88, Layer:    AppCbrServer, (0) Total number of packets received: 875
Node:     90, Layer:    AppCbrServer, (0) Total number of packets received: 988
Node:     94, Layer:    AppCbrServer, (0) Total number of packets received: 363
Node:     95, Layer:    AppCbrServer, (0) Total number of packets received: 1014
Node:     96, Layer:    AppCbrServer, (0) Total number of packets received: 874
```

Other parameters such as Packet Data Ratio, Percentage of Loss Rate and Packets Sent by the sources can be obtained with the following shell and awk programs:

```
Shell program:
      cat $1 | grep App | grep Total | grep packets  |  awk -f glomopak1.awk

AWK program:
     BEGIN{
      sumcountsent = 0;
      sumcountrcv  = 0;
     }

     {
      if ($10 == "sent:") sumcountsent+= $11;
        else if ($10 == "received:") sumcountrcv += $11;
     }

     END{
      printf("Loss Packet Percentage = %f \% \n", 100-((sumcountrcv*100)/sumcountsent));
      printf("Packet Delivery Ratio = %f \n\n",sumcountrcv/sumcountsent);
      printf("Packets Received = %d \n", sumcountrcv);
      printf("Packets Sent     = %d \n\n", sumcountsent);
     }
```

An example of the execution of these programs on the previous example is the following:

```
./analy-datpak2.sh cmpmac-802aodv-200.stat
Loss Packet Percentage = 36.079167 %
```

```
Packet Delivery Ratio = 0.639208

Packets Received = 15341
Packets Sent     = 24000
```

To obtain the Average Delay (in the case of CBR traffic) and the Control Packets generated by the AODV routing protocol, the following programs can be used:

```
# program analy-delay1.sh
cat $1 | grep AppCbrServer | grep end-to-end | grep delay  | awk -f delay1.awk

where delay1 AWK program is:
BEGIN{
    sumdelay = 0;
    countdelay = 0;
}

{
    sumdelay += $10;
    countdelay++;
}

END{
   printf("Average Delay = %f \n", sumdelay/countdelay);
}

-----------------------------------
# program analy-routAODVctrl-1.sh
  cat $1 | grep RoutingAodv | grep CTRL  \
     | awk '{sum += $11} END {print sum}'
```

We can even obtain some interesting graphs from simulations. Lets suppose we want to obtain a graph of the variance of packets delivered to final destination, with networks of different sizes (30, 50, 70, 100 and 200 nodes) and with nodes move at different speeds (1, 5, 10 and 20 m/s). We can run several simulations (each with a specified node number and speed of the mobility model). To shorten the simulation time, the tests can be executed at the same time [2], thus obtaining the following table.

Speed	*30 nodes*	*50 nodes*	*70 nodes*	*100 nodes*	*200 nodes*
1	12885	34940	56539	57514	67310
5	16927	33433	43038	36315	68720
10	14393	22421	34559	34884	44615
20	12108	20455	29128	27706	28297

If the data of this table is used in a file (e.g *mobil.txt*), then we may use graphic tools such as GnuPlot to obtain an appropriate graph. In the case of GnuPlot, we can execute the following commands to obtain the graph represented at Fig. [4] and contained in file *nodedens12.eps*.

```
set terminal postscript eps
set size 1/1., 1/1.
```

[2] In order to run several simulations simultaneously, several images of GloMoSim may be installed in different directories. At the end of each simulation, the results of the *glomo.stat* file have to be collected individually. The environment variables of GloMoSim have to be specified according to each case.

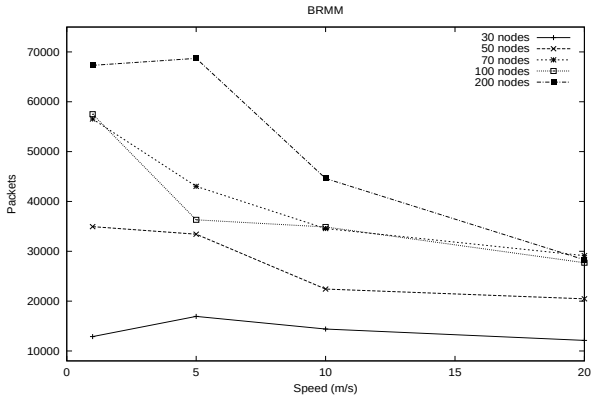

Figure 4: Number of Data Packets received by their destination

```
set title "BRMM"
set output "nodedens12.eps"
set xlabel "Speed (m/s)"
set ylabel "Packets"
plot [0:20][8000:75000] "mobil22.txt" using 1:2 title '30 nodes' with linespoints, \
                  "mobil.txt" using 1:3 title '50 nodes' with linespoints, \
                  "mobil.txt" using 1:4 title '70 nodes' with linespoints, \
                  "mobil.txt" using 1:5 title '100 nodes' with linespoints, \
                  "mobil.txt" using 1:6 title '200 nodes' with linespoints
```

4 Notes on Routing Protocols in GloMoSim

4.1 AODV

GloMoSim implements AODV based on an older IETF draft (draft-ietf-manet-aodv-03.txt). The protocol assumes the MAC protocol sends a signal to the routing protocol when it detects link breaks. MAC protocols such as IEEE 802.11 and MACAW have this functionality. In IEEE 802.11 for example, when no CTS is received after RTS, and no ACK is received after retransmissions of unicasted packet, it sends the signal to the routing protocol.

If users want to use MAC protocols other than IEEE 802.11, they must implement schemes to detect link breaks. A way to do this is, for example, using HELLO packets, as specified in AODV documents. Unsolicited RREPs are broadcasted and forwarded only if the node is part of the broken route and not the source of that route. If more than one route uses the broken link, multiple RREP messages are sent.

4.2 FSR

FSR is based on the premise that changes in a network region's topology have less effect on a router's packet forwarding decisions as the distance (in hops) between the router and the network increases. FSR is an adaptation of GSR where instead of propagating information through the network by periodic exchanges, a node exchanges individual link state table entries at different rates depending on the distance to the link's source. Each update message does not contain information about all nodes. It exchanges information about closer nodes more frequently than it does about farther nodes thus reducing the update message size.

Fig. [5] defines the scope of fisheye for the center node. The scope is defined as the set of nodes that can be reached within a given number of hops; where three scopes are shown in the example. In GloMoSim,

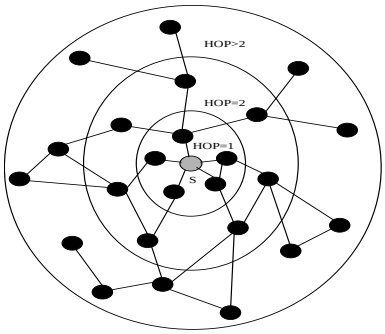

Figure 5: Accuracy of information in FSR

when FSR will be used as the routing protocol, the following entries have to be defined in the CONFIG.IN file:

```
                    ROUTING-PROTOCOL    FISHEYE
                    FISHEYE-FILE        FISHEYEConfig
--------
Where the default FISHEYEConfig file is:   2  15S   5S  15S
```

The FISHEYEConfig file contains the following routing parameters:

- *Size of the scope.* This parameter specifies the scope radius of a node in number of hops.
- *Time-out for the neighboring nodes.* If a node does not hear from a neighbor specified by this value, the neighbor node will be deleted from the neighbor list.
- *Intra scope update interval.* The update interval of sending the updates of the nodes within the scope radius.
- *Inter scope update interval.* The update interval of sending the updates of the nodes outside the scope radius.

An example of scenario using the FSR routing protocol can be found at *./glomosim-2.03/ glomosim/ scenarios/ routing-fisheye* directory.

4.3 WRP

Wireless Routing Protocol (WRP) is a pro-active protocol that maintains routing information through the exchange of triggered and periodic updates. A node that successfully receives an update message, transmits an acknowledgement back to the sender, indicating the link is still viable. In the event that a node has not transmitted anything within a specified period of time, it must transmit a *Hello* message (instead of exchanging the entire route table) to ensure connectivity. Otherwise, the lack of messages from a node indicates the failure of that link. When a node receives a Hello message from a new node, it sends that neighbor a copy of its routing table information. Each node maintains a *distance table*, a *routing table*, a *link-cost table*, a *message retransmission list* and an *ack-status table.* The implementation of WRP in GloMoSim is based upon the pseudocode in [11].

However, the relatively large number of tables that this protocol requires imposes tight memory constraints on the mobile devices, especially when the network begins to grow. This in turn places greater demands on the battery power of the mobile device, and in the case of the GloMoSim, it happens sometimes that simulations of more than 100 nodes result in a segmentation core error. In such cases, parameter NUMBER-OF-NODES must be set to 99.

5 The GloMoSim Developper Guide

5.1 Coding in GloMoSim

In many cases we would like to modify the source files or create new files in order to add or obtain new functionality. New protocols can be written and interfaced with GloMoSim at any layer of the OSI model. In general, the coding style of the source file should be followed when modifying the code. Some general advices are the following:

- Use long, descriptive and easily read procedure and variable names. Try not to use abbreviations or encodings. Use real words in variable names like *Header* not *Hdr*.
- Macro constants should be upper case letters with underlines.
- Identifiers for functions performing an action should contain a verb, for example, *InitializeValues, CreateQueue* etc.
- Identifiers for boolean functions or variables should have *Is* or *Has* in their name. For example, *IsOccupied*.
- Do not use more than 80 column width as wrapped lines are very difficult to read.
- Block comments are on separate lines from code. They are indented to the same level as the code they are in. The format for block comments is:

```
//
// This is a block comment. First line.
// This is a block comment. Second line.
// This is a block comment. Third line.
// This is a block comment. Fourth line.
//
              or

/*
 * This is a block comment. First line.
 * This is a block comment. Second line.
 * This is a block comment. Third line.
 * This is a block comment. Fourth line.
 */
```

Source Files must have the following layout: There should be a block comment giving the filename, its purpose and perhaps a revision history. For header files, there should always be a *#ifndef HEADERFILE_H, #define HEADERFILE_H, #endif* block around the header to allow for multiple includes. Source file must be organized into a logical order (grouping related functions together). Functions, types and macros that are only used in a single file should be declared *static* and put in the source file and not in a header file. Functions, types and macros that are used by other source files must be declared in a header. For function commenting, precede every prototype by a block comment containing the function name, a description of its purpose, what it returns and the assumptions it makes. Optionally, you may also list each argument and its effect if not already obvious from the description. For Example:

```
//
// FUNCTION:     IsEmptyQueue()
// PURPOSE:      Checks if the queue is empty.
```

```
// RETURN VALUE: Returns True if the queue is empty
// ASSUMPTIONS:  None.
//
```

For identation, each nested piece of code should be indented by four (4) spaces. TAB characters are not allowed and will be removed by the *expand* command. For emacs, put *(setq-default indent-tabs-mode nil)* in the *.emacs* file to eliminate TAB characters. Other recommendations are:

- Complex expressions must match the parentheses level. Always use braces on *if/else, while* and *for* statements.

```
  if (.....) {
    Statement
}
else {
    Statement
}
```

- Use enumerated constants to give names to a set of flags. for example:

```
 BAD:                          GOOD:
#define  INTEGER    0              typedef enum {
#define  CHAR       1                  Integer,
#define  FLOAT      2                  Char,
#define  DOUBLE     3                  Float,
                                       Double
                                   } DataType;
```

- Functions like *bcopy()* and *bzero()* are non-ANSI standard functions and should not be used. Instead use the functions *memcpy()* and *memset()* which are ANSI standard functions.

5.2 Structure of GloMoSim

After downloading and upzipping GloMoSim, it contains the following directories:

/application	contains code for the application layer
/bin	for executable and input/output files
/doc	contains the documentation
/include	contains common include files
/java_gui	contains the visual tool
/mac	contains the code for the mac layer
/main	contains the basic framework design
/network	contains the code for the network layer
/radio	contains the code for the radio layer
/scenarios	contains some example scenarios
/tcplib	contains libraries for TCP
/transport	contains the code for the transport layer

Directory *main* contains the basic framework to execute GloMoSim, which includes **driver.pc** file that defines the driver entity. Every Parsec program must include an entity called *driver* which serves a purpose

similar to the main function of a C program. This file reads the configuration file, initiate the simulation and writes the final statistics results file *glomo.stat* from some temporary file (.STAT.x). The sequence of events at run time is as follows:

1. The main function in driver.pc is run. This is the C main function, where GloMoSim starts.
2. The main function calls parsec_main() to start the Parsec simulation engine, initialize the simulation runtime variables and create the driver entity. The *parsec_main* function is used when the user wants to write his own *main* and is found at *PCC_DIRECTORY/include/pc_api.h* (since the function is part of the Parsec runtime system, it is not possible to access the source for it).
3. When the simulation ends, parsec_main() returns, and the rest of the main function is executed.

The reason that GloMoSim uses a main function is because it is needed to collect the statistics into a single file and close the file neatly. Therefore a function that runs after all the glomo entities have been finalized is required. An example of how to use parsec_main() function to print a message is represented by the following *hola.pc* program:

```
#include <stdio.h>

int main(int argc, char **argv){
   parsec_main(argc,argv);

  printf("Simulation Ended\n\n");
}

entity driver (int argc, char **argv) {
    int i;

    printf("HELLO WORLD \n\n");
}
```

The entity driver is called by the parsec_main() function, prints the message "HELLO WORLD", the simulation ends and the message "Simulation Ended is printed". The commands to compile the program are shown as follows. The *clock* option specify clocktype to use (unsigned by default), while the *user_main* option rename main() to parsec_main() for a user-defined main().

```
> pcc -g -O3 -clock longlong -lm -c hola.pc
> pcc -g -O3 -clock longlong -user_main hola.o -lm -o hola
>./hola
HELLO WORLD

Execution time :   0.0007 sec
Number of events (including timeouts) processed : 0
Number of messages processed : 0
Number of context switches occurred : 6
Number of Local NULL messages sent : 0
Number of Remote NULL messages sent : 0
Total Number of NULL messages sent : 0
Simulation Ended

>
```

In GloMoSim, the driver entity (in ./main/driver.pc) reads the input file descriptor, establishes partitions, allocates memory for node information, calls appropriate functions depending on the read input values such as simulation time and node placement, and finally starts simulation by sending a *StartSim* message to the *partitionEntityName* instance of the GLOMOPartition entity type (defined in the glomo.pc file). The **./main/glomo.pc** file performs the following steps:

1. Receive information from driver entity.
2. Find out nodes which belong to current partition and initialize all the layers for these nodes by calling the following entities: GLOMO_PropInit(), GLOMO_RadioInit(), GLOMO_MacInit(), GLOMO_NetworkInit(), GLOMO_TransportInit(), GLOMO_AppInit() and GLOMO_MobilityInit().
3. Go into an infinite loop trying to receive messages. When a message is received it retrieves information about the receiving node and calls the appropriate layer. It displays current simulation time during program execution.
4. When the simulation ends, it goes to the finalize code. This code will call the Finalize function for all the layers of all the nodes in this partition. The developer can thus collect any needed statistics. The finalize functions are: GLOMO_RadioFinalize(), GLOMO_MacFinalize(), GLOMO_NetworkFinalize(), GLOMO_TransportFinalize(), GLOMO_AppFinalize() and GLOMO_MobilityFinalize().

The mentioned functions to initialize, call and finalize all the layers are defined at **./include/structmsg.h** file and coded in the appropriate directory. For example GLOMO_RadioInit() is coded at **./radio/radio.pc**, while GLOMO_MacInit() is coded at **./mac/mac.pc**, GLOMO_NetworkInit() is coded at **./network/network.pc**, GLOMO_TransportInit() is coded at **./transport/transport.pc**, etc.

Eventually, all layers are executed for a message sent by any node.

5.3 GloMoSim Design

GloMoSim is designed in a layered approach with standard APIs used between the different simulation layers. The protocol stack includes models for the channel, radio, MAC, network, transport and higher layers. The GloMoSim kernel APIs are in the form of function calls, while for the other layers, the API is in the form of message exchanges required to interact with the layers. Several APIs are presented in the rest of this section.

5.3.1 Timers

GloMoSim has several in-built timers at different layers. Generally, a timer schedules an event that goes to the corresponding layer. File */glomosim/include/structmsg.h* defines events used in GloMoSim, although the user can add in this file his own-defined events. Some of the events defined are:

```
/* Message Types for Channel layer */
MSG_CHANNEL_FromChannel,
MSG_CHANNEL_FromRadio,

/* Message Types for Network layer */
MSG_NETWORK_FromApp,
MSG_NETWORK_FromMac,
MSG_NETWORK_FromTransportOrRoutingProtocol,
MSG_NETWORK_DelayedSendToMac,
MSG_NETWORK_RTBroadcastAlarm,
MSG_NETWORK_CheckTimeoutAlarm,
MSG_NETWORK_TriggerUpdateAlarm,
MSG_NETWORK_InitiateSend,
MSG_NETWORK_FlushTables,
```

```
    MSG_NETWORK_CheckAcked,
    MSG_NETWORK_CheckReplied,
```

We can consider the AODV routing protocol as an example. In the protocol, when a node sends a Route Request (RREQ), it has to verify that eventually a Route Reply (RREP) is received by using a timer. Therefore in file */glomosim/network/aodv.pc* (further functions discussed in this subsection are defined in this file and in */glomosim/network/aodv.h*), the last instruction of the *RoutingAodvInitiateRREQ* function is:

```
void RoutingAodvInitiateRREQ(GlomoNode *node, NODE_ADDR destAddr)
{
                .  .  .

 RoutingAodvSetTimer(node, MSG_NETWORK_CheckReplied, destAddr,
                        (clocktype)2 * ttl * NODE_TRAVERSAL_TIME);

} /* RoutingAodvInitiateRREQ */
```

Function *RoutingAodvSetTimer* set the timer by sending a message to the corresponding network layer. The last argument of this function is the delay from the current simulation time. NODE_TRAVERSAL_TIME is defined by default to 40ms in the */glomosim/network/aodv.h* file. Function *RoutingAodvSetTimer* is defined as follows:

```
void RoutingAodvSetTimer(
    GlomoNode *node, long eventType, NODE_ADDR destAddr, clocktype delay)
{
    Message *newMsg;
    NODE_ADDR *info;

    newMsg = GLOMO_MsgAlloc(node,
                            GLOMO_NETWORK_LAYER,
                            ROUTING_PROTOCOL_AODV,
                            eventType);

    GLOMO_MsgInfoAlloc(node, newMsg, sizeof(NODE_ADDR));
    info = (NODE_ADDR *) GLOMO_MsgReturnInfo(newMsg);
    *info = destAddr;
    GLOMO_MsgSend(node, newMsg, delay);

} /* RoutingAodvSetTimer */
```

When timer expires, function *RoutingAodvHandleProtocolEvent* takes the appropriate actions for this event:

```
void RoutingAodvHandleProtocolEvent(GlomoNode *node, Message *msg)
{
          .  .  .
    switch (msg->eventType) {
               .  .  .
                   /* Check if RREP is received after sending RREQ */
        case MSG_NETWORK_CheckReplied: {
            NODE_ADDR *destAddr = (NODE_ADDR *)GLOMO_MsgReturnInfo(msg);
```

```
            /* Route has not been obtained */
            if (!RoutingAodvCheckRouteExist(*destAddr, &aodv->routeTable))
            {
                if (RoutingAodvGetTimes(*destAddr, &aodv->sent) < RREQ_RETRIES)
                {
                    /* Retry with increased TTL */
                    RoutingAodvRetryRREQ(node, *destAddr);
                } /* if under the retry limit */
                            .  .  .

} /* RoutingAodvHandleProtocolEvent */
```

5.4 How to add a new Protocol

Different steps have to be considered when adding a model or protocol to a layer in order to maintain consistency and proper functionality in GloMoSim. Three main functions have to be elaborated to add a new protocol:

1. **Initialization Function**. It must allocate and initialize the model specific data.
2. **Finalization Function**. It generates the output statistics from the simulation run for this model.
3. **Simulation Event Hanling Function**. It performs simulation actions when scheduled with an event.

We will use a new protocol called NEWPROT to show the main steps to define a protocol in the network layer. This new protocol will call the AODV routing protocol in order to simplify the explanation, and show the mechanisms used by GloMoSim to initiate the new model.

We start by indicating how the network layer is called. As already mentioned, file *glomo.pc* receives information from the driver entity, initializes all layers for nodes and goes into an infinite loop trying to receive messages. When a message is received, it calls the appropriate node and layer with the *GLOMO_CallLayer()* function:

```
    GLOMO_CallLayer(GlomoNode  *node, Message * msg){
       .  .  .
         switch (GLOMO_MsgGetLayer(msg) ){
            case GLOMO_RADIO_LAYER:
                    GLOMO_RadioLayer(node, msg);
                    break;
            case GLOMO_MAC_LAYER:
                    GLOMO_MacLayer(node, msg);
                    break;
            case GLOMO_NETWORK_LAYER:
                    GLOMO_NetworkLayer(node, msg);
                    break;
            case GLOMO_TRANSPORT_LAYER:
                    GLOMO_TransportLayer(node, msg);
                    break;
            case GLOMO_APP_LAYER:
                    GLOMO_AppLayer(node, msg);
                    break;
         }
       .  .  .
}
```

Function *GLOMO_NetworkLayer()* models the behaviour of the network layer on receiving the message, and calls function *NetworkIPLayer()* -defined in file *./network/nwip.pc*. The first step consists in defining the initialization routine in this file. In our example, we add the necessary code in order to recognize the NEWPROT protocol:

```
  void NetworkIpInit(GlomoNode* node, const GlomoNodeInput* nodeInput)
{
        .  .  .
   retVal = GLOMO_ReadString(node->nodeAddr, nodeInput,
                             "ROUTING-PROTOCOL", protocolString);

   if (retVal == FALSE) {
      printf("CONFIG.IN Error: ROUTING-PROTOCOL not specified!\n");
      assert(FALSE); abort();
   }
        .  .  .
         else if (strcmp(protocolString, "AODV") == 0) {
      ipLayer->routingProtocolChoice = ROUTING_PROTOCOL_AODV;
      RoutingAodvInit(
         node, (GlomoRoutingAodv**)&ipLayer->routingProtocol, nodeInput);
   }

/* *******          THIS IS THE ADDED CODE             ******* */

   else if (strcmp(protocolString, "NEWPROT") == 0) {
      ipLayer->routingProtocolChoice = ROUTING_PROTOCOL_NEWPROT;
      RoutingAodvInit(
         node, (GlomoRoutingAodv**)&ipLayer->routingProtocol, nodeInput);
   }
        .  .  .
}
```

This function takes the value read from the input-configuration file, and calls the appropriate initialization routine. The Finalization routine is also defined in the *nwip.pc* file:

```
void NetworkIpFinalize(GlomoNode *node)
{
   GlomoNetworkIp* ipLayer = (GlomoNetworkIp*)node->networkData.networkVar;

   switch (ipLayer->routingProtocolChoice) {
   case ROUTING_PROTOCOL_LAR1:
      NetworkLar1Finalize(node);
      break;
   case ROUTING_PROTOCOL_AODV:
      RoutingAodvFinalize(node);
      break;
   case ROUTING_PROTOCOL_NEWPROT:
      RoutingAodvFinalize(node);
      break;
      .  .  .
}
```

The type ROUTING_PROTOCOL_NEWPROT used in these functions has to be declared in the file *./include/network.h*:

```
typedef enum {
    NETWORK_PROTOCOL_IP = 0,
    ROUTING_PROTOCOL_AODV,
    ROUTING_PROTOCOL_NEWPROT,   /* New Protocol added */
    ROUTING_PROTOCOL_DSR,
    ROUTING_PROTOCOL_LAR1,
    ROUTING_PROTOCOL_ODMRP,
    ROUTING_PROTOCOL_OSPF,
    ROUTING_PROTOCOL_ZRP,
    ROUTING_PROTOCOL_ALL,
    ROUTING_PROTOCOL_NONE
} NetworkRoutingProtocolType;
```

Then the simulation event handling function has to be modified, in this case *NetworkIpLayer()* that can be found at *nwip.pc* file:

```
void NetworkIpLayer(GlomoNode *node, Message *msg) {
   switch (msg->protocolType) {
      .  .  .
   case ROUTING_PROTOCOL_AODV: {
      RoutingAodvHandleProtocolEvent(node, msg);
      break;
   }

   case ROUTING_PROTOCOL_NEWPROT: {
      RoutingAodvHandleProtocolEvent(node, msg);
      break;
   }
      .  .  .
```

The Internet protocol has an 8-bit field called *Protocol* (IPv4) or *Next Header* (IPv6) to identify the next level protocol. The next step is to define the protocol number for our new protocol in file *./include/nwcommon.h*:

```
/* protocol number for IP */
#define IPPROTO_TCP  6
#define IPPROTO_UDP  17
#define IPPROTO_OSPF 87
#define IPPROTO_BELLMANFORD 520
#define IPPROTO_FISHEYE 530
#define IPPROTO_AODV 123
#define IPPROTO_NEWPROT 123
#define IPPROTO_DSR 135
#define IPPROTO_ODMRP 145
#define IPPROTO_LAR1 110
#define IPPROTO_ZRP 133
```

In this case, we assign the same protocol number of AODV because in fact, NEWPROT protocol is calling AODV. Finally we have to modify file *./application/application.pc*, to indicate the simulator that the NEWPROT is defined at the network layer and not at the application layer [3]:

[3]GloMoSim defines certain routing protocols such as AODV, DSR and LAR at the network directory, while others such as WRP and FSR at the application directory.

```
void GLOMO_AppInit(GlomoNode *node, const GlomoNodeInput *nodeInput)
{
      .  .  .
    retVal = GLOMO_ReadString(node->nodeAddr, nodeInput,
                              "ROUTING-PROTOCOL", buf);
    if (strcmp(buf, "BELLMANFORD") == 0) {
        node->appData.routingProtocol = APP_ROUTING_BELLMANFORD;
        RoutingBellmanfordInit(node);
    }
    else if (strcmp(buf, "OSPF") == 0) {
        // Protocol is at routing layer.
    }
    else if (strcmp(buf, "WRP") == 0) {
        node->appData.routingProtocol = APP_ROUTING_WRP;
        RoutingWrpInit(node, nodeInput);
    }
      .  .  .
    /* Fine because AODV and NEWPROT are defined at network layer*/
    else if (strcmp(buf, "AODV") == 0) {
    }
    else if (strcmp(buf, "NEWPROT") == 0) {
    }
      .  .  .
```

When these modifications have been completed, we rebuild GloMoSim by executing the command *make* in the *main* directory. When we execute any simulation that defines NEWPROT as routing protocol, we will verify in file *glomo.stat* that AODV protocol has been used.

6 Installation and Troubleshooting

6.1 Installation

After uncompressing the GloMoSim downloaded file, two subdirectories can be found: *glomosim* and *parsec*. The pre-compiled PARSEC runtime libraries for the following operating systems are included in the *parsec* directory:

./parsec/aix/	IBM AIX with xlc compiler
./parsec/windowsnt-4.0-vc6/	MS Windows NT or 2000 with Visual C++ ver. 6.0
./parsec/freebsd-3.3/	FreeBSD 3.3
./parsec/redhat-6.0/	Red Hat 6.0 or higher
./parsec/solaris-2.5.1/	Sun SPARC Solaris 2.5.1 or higher with gcc/g++
./parsec/solaris-2.5.1-cc/	Sun SPARC Solaris 2.5.1 or higher with Sun C compiler
./parsec/x86-solaris-2.5.1/	Sun X86 Solaris 2.5.1 or higher with gcc/g++
./parsec/irix-6.4/	SGI IRIX 6.4 or higher with gcc/g++

In Unix-based systems, if we have permission to create */usr/local/parsec* directory, then we just copy the whole subdirectory with the name of target platform under */usr/local/parsec*. If we do not have permission to create a directory under */usr/local*, then create a directory anywhere else and set the designated directory to an environment variable "PCC_DIRECTORY." For instance, you can set the variable by typing "setenv PCC_DIRECTORY /home/example/parsec" if we are using (t)csh. Parsec environment variables and compiler options pcc checks the following environment variables when executed:

```
PCC_DIRECTORY      : Directory that pcc looks up
PCC_CC             : C compiler used for preprocessing and compiling
PCC_LINKER         : Linker used for linking
PCC_PP_OPTIONS     : Options for preprocessing
PCC_CC_OPTIONS     : Options for compiling
PCC_LINKER_OPTIONS : Options for linking
```

Several errors during installation can arise if these options are not set up correctly or if the */parsec/<platform>/bin* directory is not included in the PATH variable.

GloMoSim requires a C compiler to run and works with most C/C++ compilers on many common platforms. Nevertheless, in the case of Linux Red Hat 6.0, the default gcc 2.96 version does not support completely the ANSI standard. Therefore there are two options, either to replace it with the gcc 2.95 version or use the egcs version by defining the following variables:

```
setenv PCC_CC egcs
setenv PCC_LINKER egcs++
```

6.2 Visualization Tool

To install the visualization tool (VT), the Java Development Kit (JDK) version 1.2 or higher must be installed, and all the compiled GloMoSim source files must be present. Then the Java VT files are compiled by executing:

```
> cd $glomo/java_gui/
> javac *.java
```

A very common error when executing the VT is that once it is started either by the *Real Time* or *Write Trace* options, nothing is displayed on the screen, giving the impression that nothing happens. The usual solution to this problem is to copy the configuration files (nodes.input, mobility.in, app.conf and config.in) under the */java_gui* directory.

Another option is to specify in the VT the full path where the configuration file (CONFIG.IN) is located. However in this case, full paths of all secondary configuration files such as mobility, node and application configuration should also be specified in the CONFIG.IN file.

Another issue to have in mind when working with the VT is that the transmission range representation and the real transmission range defined by the power configuration parameters do not agree necessarily. This is a possible bug to be corrected in further developments.

References

[1] *GloMoSim: Global Mobile Information Systems Simulation Library.* http://pcl.cs.ucla.edu/projects/glomosim/.

[2] Mario Gerla Lokesh Bajaj, Mineo Takai, Rajat Ahuja, Rajive Bagrodia. GloMoSim: A Scalable Network Simulation Environment. Technical Report 990027, University of California, 13, 1999.

[3] Addison Lee - Kaixin Xu. GloMoSim Java Visualization Tool. documentation version 1.1, Software Distribution.

[4] Rajive Bagrodia. README file - GloMoSim Software. University of California, Los Angeles - Department of Computer Science. Box 951596, 3532 Boelter Hall, Los Angeles-CA 90095-1596 / rajive@cs.ucla.edu.

[5] Theodore S. Rappaport. *Wireless Communications: Principles and Practice.* Prentice Hall, New Jersey, 1999.

[6] William Stallings. *Wireless Communications and Networks*. Prentice Hall, New Jersey, 2000.

[7] David B. Johnson and David A. Maltz. "Dynamic Source Routing in Ad Hoc Wireless Networks". In Imielinski and Korth, editors, *Mobile Computing*, volume 353 of *The Kluwer International Series In Engineering And Computer Science*. Kluwer Academic Publishers, 1996.

[8] Kevin Fall and editors Kannan Varadhan. "NS notes and documentation". The VINT Project, UC Berkeley, LBL, USC/ISI and Xerox PARC, November 1997. Available at http://www.isi.edu/nsnam/ns.

[9] Zygmunt J. Haas et al. Wireless Ad Hoc Networks. In *Encyclopedia of Telecommunications*, 2002.

[10] Sung-Ju Lee Elizabeth M. Royer and Charles E. Perkins. The Effects of MAC Protocols on Ad hoc Network Communication. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, Chicago, Illinois, September 2000.

[11] Shree Murthy and J. J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *Mobile Networks and Applications*, 1(2):183–197, 1996.