Indice

Capitolo 1

Ad hoc Sim 1.0

1.1 Simulator structure

The simulator i wrote is based on OMNeT++ v2.2. It has been written under a Linux environment but (as assured by OMNeT++ author) it should be portable on Windows systems.

The simulator depict an ad hoc network with a parameterizable number of hosts that move in a field free of obstacles.
Each host have a defined transmission power that affect the range within a communication is feasible. The signal power degradation is modelled by the *Free Space Propagation Model* which states that the received signal strength is inversely proportional to the node distance square.

Each mobile host is a compound module (see **??** on page **??**) which encapsulate the following simple modules:

- A physical layer;
- A MAC layer;
- A route layer;

- An application layer;
- A mobility layer;

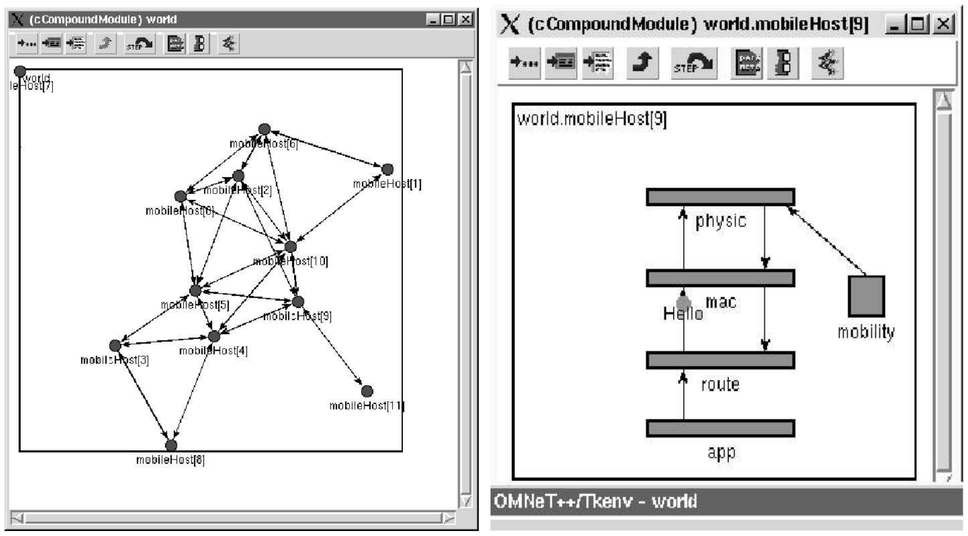

Figura 1.1: The simulation map with hosts and links between them the host internal structure.

1.1The communications between the modules is made via messages exchange.
Each module (simple or compound) can be replaced by other newly implemented one simply modifying the *omnetpp.ini* file. Then there is not the need of bothering on writing any new instruction in the the other simulator models.

Statical analysis are collected in text files and can be elaborated at the end of the simulation. Unfortunately due to a bug in the OMNeT++ kernel each data file is rewritten on every simulator new run so it is better to write a script file that copy the collected data in another file at the end of each simulator run.
Anyway in the newly released version of OMNeT++ this bug has been fixed

but due to the many simulator code modifications that the new release require, for this thesis purpose I will keep on using the 2.2 version.

To help the simulator debug I used the macro *d(string);*. The string passed as parameter has the same syntax of the OMNeT++ *"ev << string"* statement that prints out the sting passed. In addition to this d()prints out the name and ID of the calling module.

1.1.1 Physical model

It implements the physical layer of each host. In particular it cares about the on-fly creation of gates that allow the exchange of messages among the hosts. This dynamic capability represents an important contribute to the existing wireless module for OMNeT++ available in Internet. The other existing simulators like FraSiMo [**?**] are based on a central structure that cares about the inter-host messages delivery.

The FraSiMo-like structure represents a good choice for easy to implement and fast simulation but it doesn't allow the user to see the actual movements and behaviors of each node.

Every time a host moves from its position an inter-distance check on each node is performed. If a host gets close enough (depending on the transmission power of the moving node) to a new neighbor, these operations take place:

1. a new gate is created for both the compound modules (the two hosts modules);

2. a new gate is created on each of the physic simple module contained in the mobile host module;

3. a link is created between the newly created simple module gate and the compound module new gate;

4. a link is created between the two hosts modules. This last link uses the "etere" channel property. "Etere" is a channel type that I defined

and that gives to the link a delay, throughput and error probability characteristics;

When the two nodes get too far these gates are deleted braking so the link between these two hosts. Each node has its own transmission power so it can happen that a node has a link toward another host but there is not a reverse link. This, as it will be showed, will produce some data message lost that only a Transport protocol, such as TCP, can handle.

The physic module can receive messages from other hosts or from the mobility module. When this happen, if the message comes from outside and does not contains errors [1]it is sent directly to the higher levels. If the message comes from the mobility module, the host position, which is stored in this module, is updated with the values contained in the message.

When a higher level module needs to send a message, it send it to the physical level that will care about the correct delivery.
This module has a list of the current neighbors so scanning this list entries, it sends a new copy of the original message through the gates that connect the host to the other nodes.

The simulator kernel, accordingly to the gate settings and the message length, will care about the correct delivery time of the message to the neighbor.

1.1.2 Mobility models and analysis

The mobility model is the only module that doesn't fit in the submodule hosts's ISO/OSI-like structure .
As the name states this module cares about the mobility of the node in which is encapsulated. Each host has its own mobility module that can be one of the various algorithm implemented. Just changing a string value in the *omnetpp.ini* file it is possibile to assign one of these mobility algorithm:

[1]OMNeT++ messages are endowed of a boolean flag that is set by the simulator kernel according to the channel error rate

- Random Walk mobility model
- Restricted Random Walk mobility model
- Random Waypoint mobility model
- Random Direction mobility model
- Normal Markovian mobility model

Once chosen one of these models it is possible to choose the behavior of the map borders. There are two available choices:

- Rebounding behavior: makes the nodes to change the move direction accordingly to the elastic impact theory.
- Toroidal behavior: make the node to leave the map from one side and re-enter from the opposite side.

Each module whichever algorithm it implements works as follows:
At the beginning the mobility module schedule a self message. When this is delivered the module elaborate a new $< x, y >$ position accordingly to the host actual position and the implemented algorithm . These values are stored in two parameters of the newly created *MOVE* message. Then the module send this message to the physic module and reschedule the self message to be delivered in *moveInterval*[2] seconds .

Random Walk mobility model

The random Walk in the discrete interpretation of the Brownian mobility model. It is characterized by a completely unpredictable motion pattern where the speed and direction are not correlated. Due to its randomness it produces

[2]a module parameter

quite an unrealistic motion. Anyway it is often used as a worst case motion algorithm.

In letterature there are many different implementation of this model. The one implemented in the simulator follow these steps

- generate an angle uniformly distributed in [0,2π[;
- generate a speed uniformly distributed in between the parameters *[minSpeed, maxSpeed]*;
- the chosen speed and direction are maintained until a predefined distance is not covered;
- after the direction has been covered new direction and period are chosen;

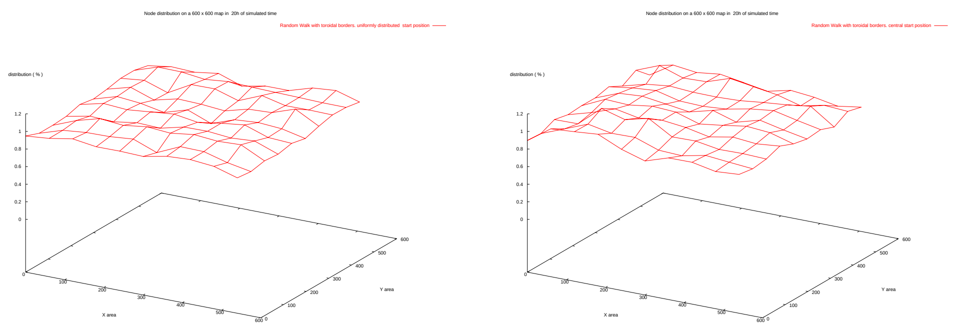

Figura 1.2: Random Walk mobility distribution of sixteen host moving on a 600x600 toroidal border map in twenty hours of simulated time. The hosts initial position is in the map center (1.1.2) or uniformly distributed (1.1.2, 1.1.2)

.

The graphics above show this mobility model behavior. The host motion is not affected by the initial position or by the borders policy so the map is uniformly covered.

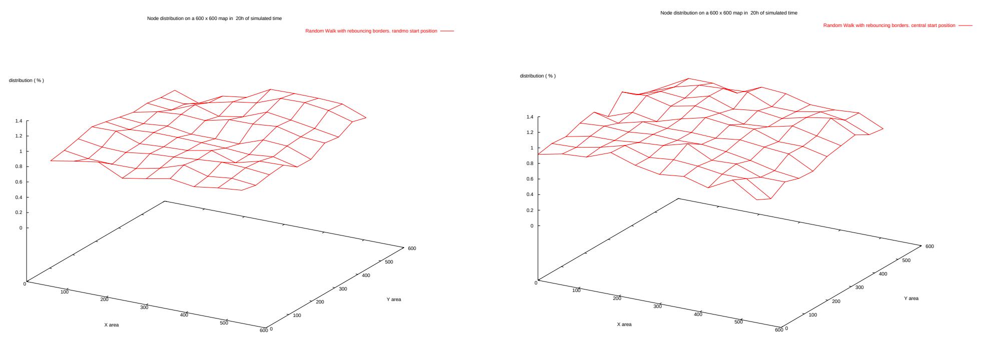

Figura 1.3: Random Walk mobility distribution of sixteen host moving on a 600x600 bouncing border map in twenty hours of simulated time. The hosts initial position is in the map center (1.1.2) or uniformly distributed (1.1.2)

.

Restricted Random Walk mobility model

The Restricted Random Walk mobility model is derived from the standard model discussed above and differs from it only when there is the need to choose a new direction and angle.RRW[3] tries to smooth the randomness of the move by choosing these new values within a limited range around the former ones. The speed is uniformly chosen between [s - k, s - k] where s is the former speed and k is a module parameter. The angle is chosen uniformly between $[\alpha - \pi/4, \alpha + \pi/4[$ where α is the former direction angle.

As shown in the graphics **??** and **??** this mobility model produce a different kind of global behavior from its ”cousin” Random Walk.
The map coverage is a little bit less uniform and the hosts initial position affect much more the map coverage. This second point is proved by the need of executing many different runs of the simulator to produce a quite uniform graph. With the Random Waypoint instead the graph was uniform even with a single simulation run.

[3]Restricted random Walk

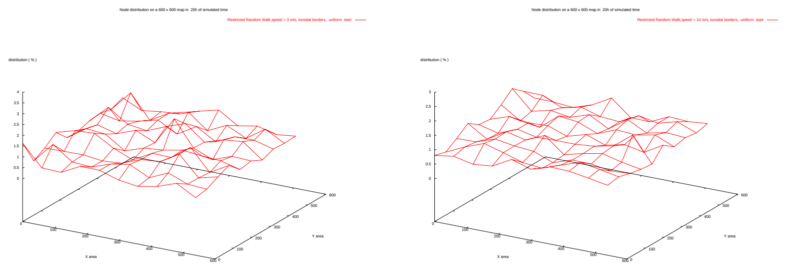

Figura 1.4: Restricted Random Walk mobility distribution of sixteen host with a speed of 3m/s (left) and 10m/s (right)on a 600x600 toroidal border map.

.

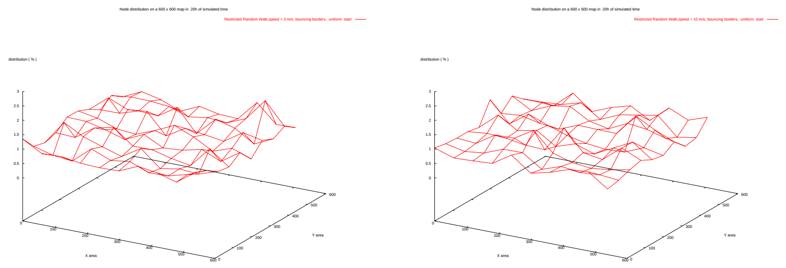

Figura 1.5: Restricted Random Walk mobility distribution of sixteen host with a speed of 3m/s (left) and 10m/s (right)on a 600x600 bouncing border map.

.

Random Waypoint mobility model

The Random Waypoint model is one of the most used mobility pattern in the ad hoc network simulations. This is due to its simplicity and its quite realistic mobility pattern.

A host whose mobility model is modelled with this algorithm, choose a destination point within the movement court, a speed uniformly distributed between the parameters *[minSpeed, maxSpeed]* and moves toward its destination for the needed time. Once the node has reached the chosen point it stays there for a given time called *pause time* and then a choose a new destination.

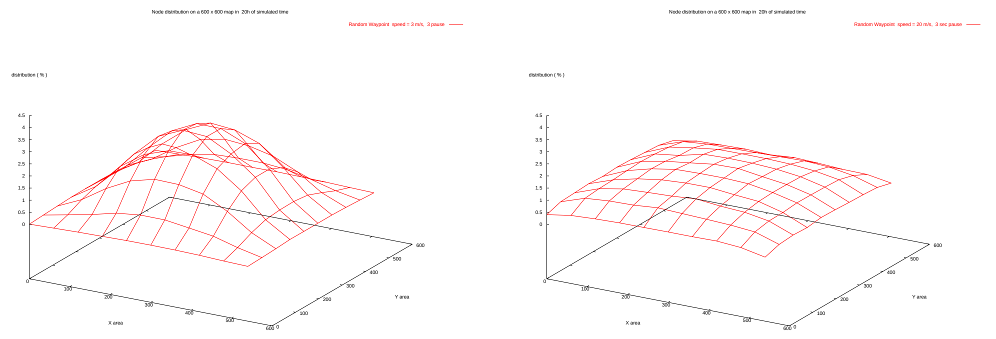

Figura 1.6: Random Waypoint mobility distribution of sixteen host moving on a 600x600 map in twenty hours of simulated time waiting. The host move with a speed of 3m/s 1.1.2 and 1.1.2wait 3 at epoch end.

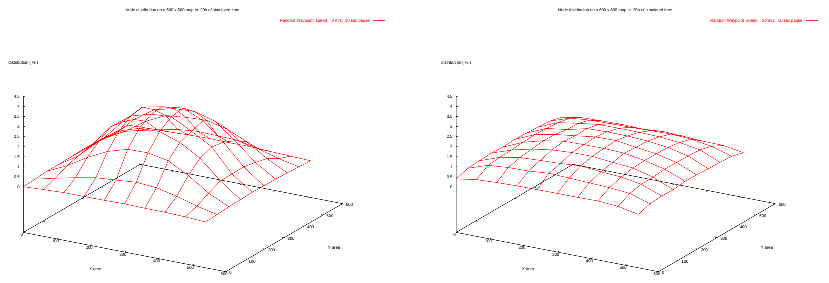

Figura 1.7: Random Waypoint mobility distribution of sixteen host moving on a 600x600 map in twenty hours of simulated time waiting. The host move with a speed of 3m/s 1.1.2 and 1.1.2 wait 10 at epoch end.

This algorithm is currently subject of criticism because of the high node density in the center of the map that this algorithm produce. In [**?**] the author explains this fact saying: "*The described effect occurs because the random waypoint model does not use an angle for direction control but chooses a destination point in the system area. Nodes in the middle of the area have a uniformly distributed angle, but nodes at the border are more likely to move back to the middle. The resulting spatial node distribution is not uniformly distributed*". As the graph shows, when the hosts average speed is higher, more new destination can be chosen and reached by the nodes. The result is a more uniformly distributed graph(1.1.2,1.1.2).

Random Direction mobility model

The Random Direction mobility model is a mix of the Random Waypoint and Random Walk models.
In order to avoid node concentration in the map center each node choose a speed and a direction distributed uniformly and goes on until it reaches a map border. Here the node wait a *pause time* and then choose a new direction and speed.

As the graph show in 1.1.2 1.1.2 1.1.2, the node speed affect very much their distribution on the map. If a node is very fast it reaches a border speedily and then it has to wait a 3 seconds pause time. The last graph shows that

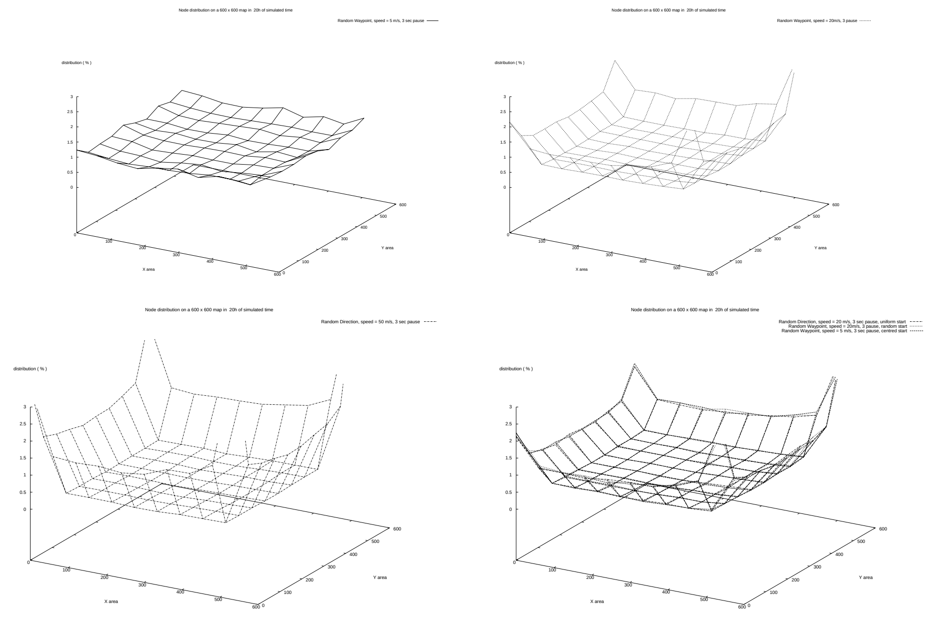

Figura 1.8: Random Direction mobility distribution of sixteen host moving on a 600x600 map in twenty hours of simulated time. The host move with a speed of (1) 3m/s, (2) 20m/s and(3) 50m/s. (4) shows the node mobility distribution with uniform, central and random initial position.

the initial distribution of the nodes does not affect their behavior. Obviously a longer pause time will sharp the map edge.

Normal Walk mobility model

The Normal mobility model [**?**, pp. 6-7] is a different kind of algorithm derived from *Markovian* models that are often used to model cellular networks instead of ah hoc networks. I decided to add it here to make a comparison with the other algorithms presented before.

As every Markovian mobility model do, the Normal walk model on each step choose a new direction and speed picking a value from a fixed range of values surrounding the former speed and direction value.

This algorithm gives the nodes quite uniform motion that keeps on "shaking" around the initial direction angle.

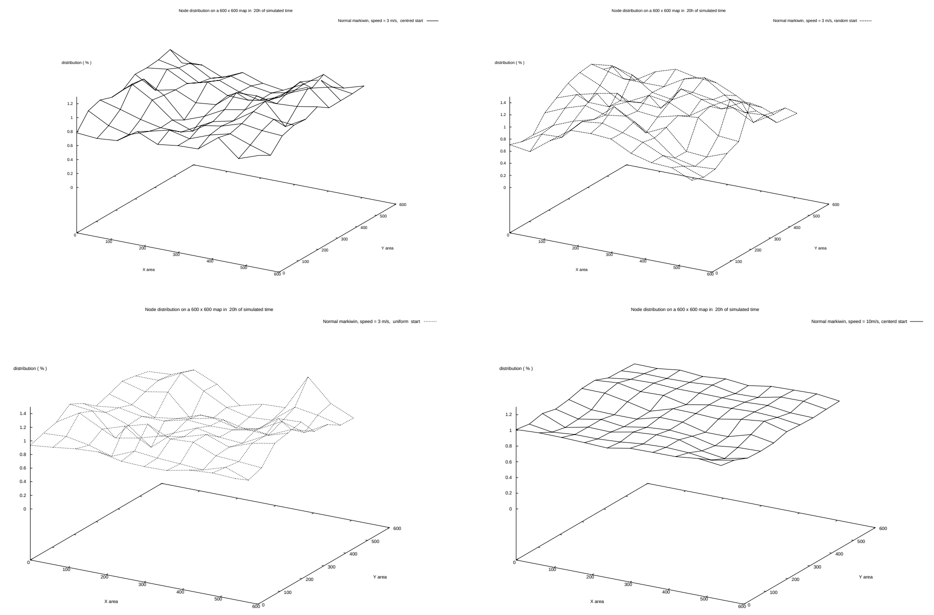

Figura 1.9: Normal Walk mobility distribution of sixteen host moving on a 600x600 map in twenty hours of simulated time waiting. The host move with a speed of 3m/s.The node mobility distribution is uniform (1), central(2) and random(3); (4) shows the higher speed movement affect on the central initial position.

The graph 1.1.2, 1.1.2, 1.1.2 show that the initial position affect considerably the node behavior that anyway become smoother when the node speed increases 1.1.2(4) .

1.1.3 Mac Layer

This module depict the ISO/OSI MAC layer. Here it is possible to insert different channel contention protocols such as CSMA/CA, MACA, MACAW and any other existent algorithm.
All of these protocols are very complex and their implementation is a new project worth.

The layer implemented is much a simpler one. The outgoing messages are let pass through. The incoming one instead are delivered to the higher levels with a *MM1 queue* policy. When a in-coming message arrives the module check a flag that advise if the higher level is busy. If it is the message is put in buffer or, if the buffer is full, it is dropped. When the higher level is no more busy, the MAC module picks the first message in the buffer, send it upward and schedule to itself an *end of service* message that will trigger a new pick from the buffer or set the busy-flag as free.

This level check all the incoming messages and watching their *mac* address[4] it let pass only those who are addressed to this module or are broadcast one. The node can even work in *promiscue mode*[5] meaning that all the messages, even the other module's one, are allowed to be elaborated by the higher levels.

This is a dangerous thing for the security of the network communication but it is as well a very important resource for all the on-demand routing protocols.

1.1.4 Routing model

The routing model is the simulator heart. This model depict the AODV routing protocol and it is set between the MAC module and the application module[6].

It receives *DATA* messages from the higher layers and tries to find a route to the chosen destination looking in its routing table or sending control messages (a *RREQ*) to get a new route.

All the AODV control messages are 512 byte long and all the control fields are stored in the message attachable parameters. The Data message size instead can be chosen by setting the correct parameter in the application layer.

[4]stored as a message parameter

[5]this is the default setting

[6]The transport layer that should be set between the application and the routing module has not been implemented because it is out of this thesis purpose and because it would have slowed down the simulator speed

OMNeT++ measures the messages size in bit, so it is necessary to multiply the wanted size by 8.

AODV was engineered to be as adaptable and flexible as possible. This make the protocol to have many options that the user can choose to implement depending on the system characteristics.
This simulator implements those options suitable for an ad hoc network and in particular :

- *HELLO* message exchange between neighbor nodes. As stated by the standard, when the underlying layer does not provide any information about the link status, HELLO messages are used to check the neighbor status;

- The *expanding ring search* optimization is used to regulate RREQ broadcast. This means that initially a route request is sent using a small *ttl* hoping that some neighbor knew a route toward the chosen destination. If the request fails, a new one with a bigger *ttl* is sent. A fixed number of retrials is allowed after which the transmission trial is aborted;

- Due to the asymmetrical nature of the wireless link, *ACK* messages are used to confirm the correct delivery of a *RREP* message. AODV standard draft[7]uses these kind of messages only for route reply confirmation. Data message acknowledgment is referred to the transport layer;

- A *black list* is used to avoid unreliable neighbor nodes. A node, let's call it X, inserts a neighbor Y, in the black list when X , after trying a number of times to send a *RREP* message to Y, it does not receive any acknowledgment. When this happen it means that the link between X and Y is unidirectional, X "hears" the messages sent by Y but Y doesn't do the same with X's messages.
  AODV standard says that when a node X put a neighbor Y in the black

[7]the latest release is the version 10

list, X will not consider any new RREQ messages coming from Y for a fixed amount of time.
Hello message coming from Y are still processed by X. This may bring some problems to this node that might wish to comunicate directly with to Y and because Y is in X's route table, no RREQ will be sent and the straight route will be used.

The AODV model uses many kind of messages, each one defined with a constat value and with it own parameters. Self messages are scheduled to trigger some future actions like a route expiration or a route request time out. A brief list of these messages is reported here to ease the understanding of the model's code.

- HELLO: a Hello message;
- RREQ: a Route Request message;
- RREP: a Route Reply message;
- RERR: a Route Error, it contains a list of all the no more valid destinations;
- DATA: a Data message;
- RREP_ACK: a RREP acknowledgment message;
- DELETE: a self message scheduled to trigger a route expiration event. As AODV standard states, the first time a DELETE message is processed, it is scheduled to occur again in the future. In this way a "last chance" is given to this invalid route before deleting it;
- FLUSH: this message handle the RREP time out. after occurring a fixed number of time all the messages stored in the data messages directed to the unknown destination are deleted from the output buffer;

- SEND_HELLO: a self message that triggers the host to send a new Hello message;
- BLK_LIST: a self message that triggers the exit of a neighbor node from the black list;

Each of these messages have parameters used to exchange control information among the nodes. It follows a short list with the most important one, often common to every AODV control messages.

originator : used in the RREQ and RREP messages, it stores the ID of the RREQ generator host. Its value is copied in the same RREP field so when a host process the message can understand if he is the final message destination;

dest : used in the RREQ to specify the host that the originator wants to comunicate with;

seqNumS : stores the sequence number of the message originator. If the message makes many hops, this field will remain the same(it is not modified by the intermediary nodes);

seqNumD : stores the last known sequence number of the destination node. The value 0 is used if this host was unknown;

source : stores the ID of last hop host. Reading its value a node can understand from which neighbor the message comes from;

mac : stores the ID of next hop node. It has the same role of the MAC address in a wired LAN;

ttl : the *time to live* value expressed in number of hops;

hopNum : count the number of hops performed by a message so far;

Moreover each message kind has its own specific parameter that don't need to be explained in in order to understand the simulator outline.

When a Data message is received various information are collected to generate statistics analysis. In particular these kind of operation are undertaken:

- the number of hops performed by the packet is stored in a histogram that can be inspected at run-time;
- the latency of the message delivery is measured and added to a vector cell counter that will be used to compute the latency mean according to the number of hops performed by the packet;
- the same operation are performed for the throughput value that is computed with the formule : latency / message size;
- a delivered message counter is incremented to compute the protocol delivery ratio;

Each time a message of any kind is received from outside, its kind is added to a histogram object so at runtime it is possible to the number and the kind of messages that the host has processed so far;

1.1.5 Traffic model

This module generates the data traffic that trigger all the routing operations. Each host has its own traffic generator that can be switched on/off setting the *active* parameter in the omnetppp.ini file. This module schedules a self message to trigger the data sending operation.

The traffic is modelled by generating a packet burst of sixtyfour messages sent to a randomly chosen destination that stays the same for all the burst length. The rate of each burst sending messages is defined by the *rate* parameter.

The time elapsed between two application bursts is defined by *burstInterval* parameter instead.

As previously said a host is identified by its ID number that the OMNeT++ kernel assigns at the simulation beginning.
These IDs are not in sequence and may vary depending on the total number of modules that work in a simulation. To generate a correct destination number, avoiding the burden of scanning all the module vector of pointers kept by the simulator kernel, the module uses a pointer to the physic layer that already has a list of all available destinations.
This optimization might look not very flexible but it saves memory and CPU cycles.

1.2 Aodv analysis

To study the behavior of AODV on an ad hoc network the following test have been performed:

- Packet delivery ratio;
- Hop distance: Average number of hops travelled by data packets that reached their destinations. Since only data packets that survive all the way to destinations are considered, low hop count means that most of the data packets delivered are destined for nearby nodes, and packets sent to remote hosts have been probably dropped. Thus, the hop count measure provides us with information about the survivability of the protocol.
- per-hop latency;
- per-hop throughput;

1.2.1 Simulator settings

All the simulation analysis have been performed with the following setting:

Map and Hosts

Map size	700m x 700m
Number of hosts	25
Host enabled to transmit	5

Physical Layer

Transmission power[8]	25000ρWatt
Receive Threshold	1ρWatt
Channel Bandwidth	11Mb/s (IEEE 802.11a)
Channel Delay	10μsec
Channel Error probability	1 bit on 10^6

Mac Layer

Busy time	normally distributed whit a λ average and a γ variance
λ	15μ sec
γ	7.5μ sec
Input Buffer size	1MB

Routing

Control Message Size	64 byte
HELLO interval	1sec
Allowed HELLO loss	2
Delete pertiod[9]	4sec
RREQ max trials	3

Application

Enabled Node	5
Message packet size	512byte
Burst length	64 packets
Send Packet Rate	3/sec
Burst Interval	normally distributed in [0.1,3]sec

1.2.2 Packet delivery ratio

The packet delivery ratio is the number of the messages delivered to their destination with respect to the sent data message. Only the messages sent by their sources are considered, then the number of hops a message can have done is not included in the count.

The graph here reported shows the behavior of AODV with reference to the hosts speed average.

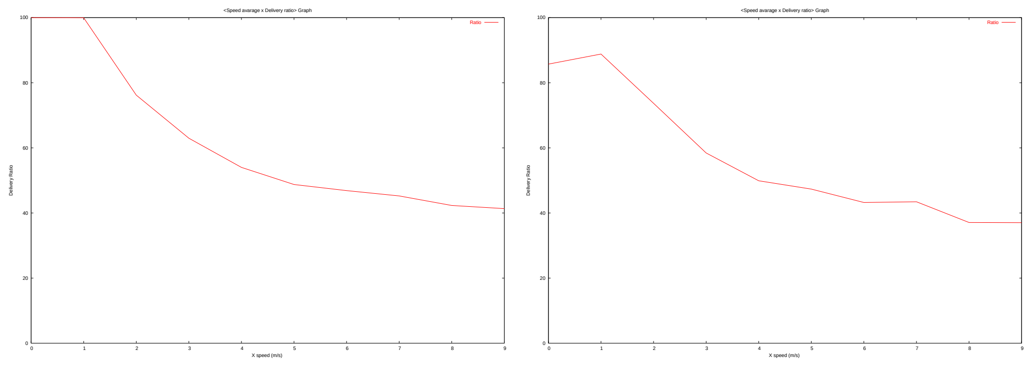

Figura 1.10: AODV delivery ratio in function of the hosts speed average. In (1) all the hosts have the same transmission range, in (2) each node has a different one.

These slightly poor results, shown in 1.2.2 are due to the lack of a reliable MAC layer that would provide a quicker reaction to link failure than the *HELLO* message system do.
This unreliable system is made even worse by the AODV specific stating that

a *route timeout*has to be shifted in the future each time a data message is sent on it.Whit such a MAC layer this is a very bad role that makes the routes survive for much more time than they should.

The 1.2.2 second graph shows the problem derived from the different transmission range that each host has from the others. The *black list* technic [10] tries to deal with this problem but does not prevent a host to send data messages to a neighbor, unreachable from this host side, without trying to find an alternative working route. Whit unmoving hosts this problem is not self-resolving and this is the reason that make the graph *1m/s* point to have a higher ratio than *0 m/s*.

Anyway the graph reported shows that the performance of this algorithm can be comparable with those almost similar reported in [**?**, p.42].

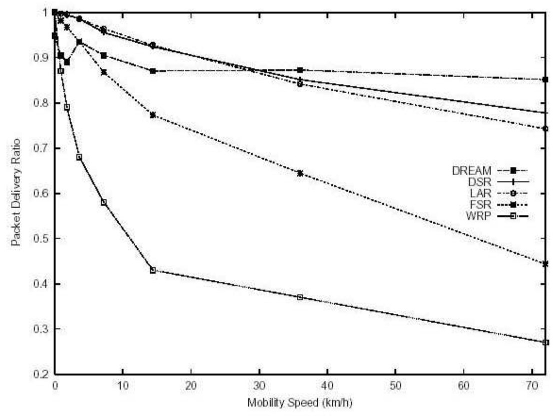

Figura 1.11: Other algorithm delivery ration. Image taken from [**?**, p.42].

[10]see 1.1.4 on page 15

1.2.3 Latency

The latency is the average of time that a packet "moves" on the network. It includes the "on air" time and the time spent in the intermediate host's buffers. It is measured considering the number of hops performed by each message;

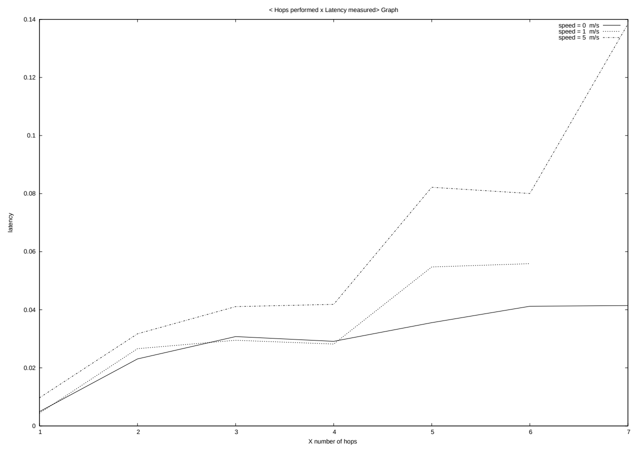

Figura 1.12: Data message latency vs number of hops performed

As expected the latency increases as the number of hops increases too. Anyway the line is not uniform because of the MM1 queue that make each message to wait a time randomly chosen. The latency is longer when the node speed average increases because of the grater number of control messages that need to be exchanged among the hosts that affect the general network performance.

1.2.4 Throughput

The throughput data reflects the effective network capacity. It is computed by dividing the message size with the time it took to arrive at its destination.

It is measured considering the hops performed by each packet.
The graph shows this data behavior with different host speed average.

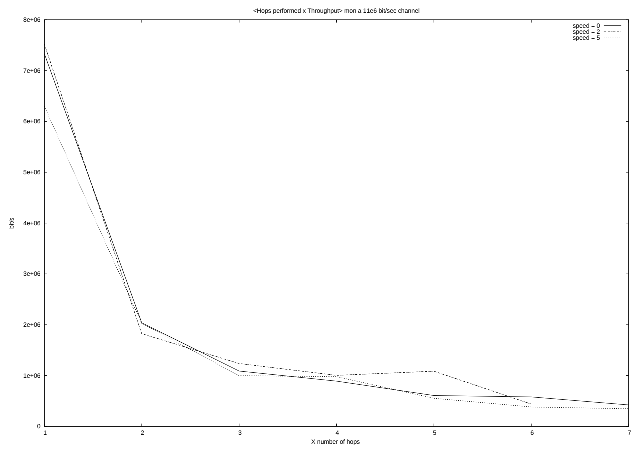

Figura 1.13: Data message throughput vs number of hops performed

As expected this value decreases when the number of hops performed gets higher. This is due to the longer latency that a message has to reach its destination. The speed

1.2.5 Hops

This metric shows the hops average that a packet do to get to its destination. Only the correctly delivered messages take part in the count.

A node enabled to transmit chooses a destination randomly. The low average is due to the higher probability of errors that the long distance message have respect to the short one.

Again some other hops averages of ad hoc protocols are reported. The graph is taken from [?] and are comparable with the data collected with my simulator .

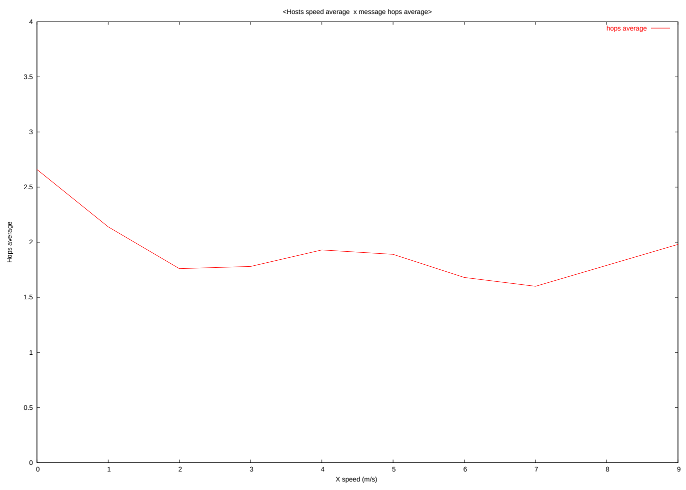

Figura 1.14: Hops average performed by the messages.

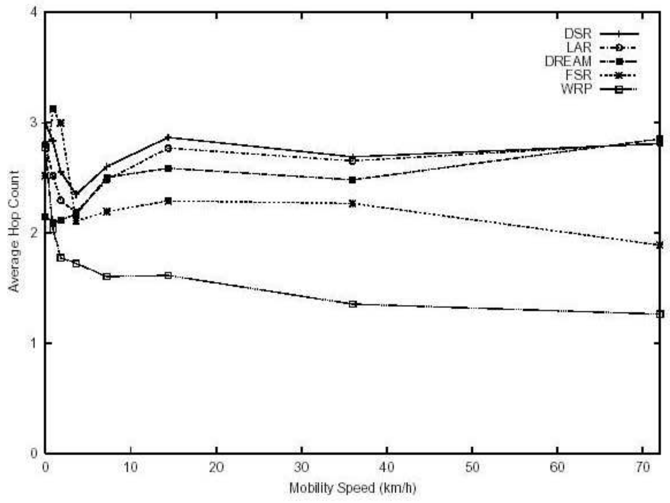

Figura 1.15: Other algorithms hops average. Image taken from [?].

1.3 Expanding the model

It is possible to expand the simulator in many ways. One can choose for example to add a new layer or to implement a new version of an existing one.

In the first case a new layer can be added for example to simulate the transport layer. This will slow down the simulator speed but at the same time will produce more accurate results. To add a new layer one should edit the "mobilehost.ned" file that contains the connections between the host layers and insert here the new connection to the brand new layer.

Implementing a new version of an already existing layer means that a certain layer will keep the same connections specified in the "mobilehost.ned" file, but it will change its behavior. For example one can implement a new routing protocol or a better MAC protocol. To implement a new version of a layer one should follow these steps:

- look the "simple.ned" file to understand what kind of parameters and gates the layer has;
- read the an existing version of the layer to understand what kind of messages the layer has to handle;
- look the "mobilehost.ned" file where there are specified the connection that the module has;
- implement its own version of the layer;
- specify to the simulator kernel to use the new layer editing the "omnetpp.ini" file;
- call the "opp_makemake" and "make" statements;

This is a useful feature of OMNeT++ that allows the user to simulate many combinations of protocol to see the differences among them. I used this feature to implement the different mobility algorithms.