1-Introduction:

When the information runs through Internet routers it needs to select the best way to reach its destination. These routers use routing protocols that are based on mathematical algorithms. These algorithms work well in small and little-changing systems. However, Internet is becoming a system of routers too extensive that changes extremely quickly, which is challenging traditional routing systems.

In seeking to enhance existing systems, researchers are looking at bio-inspired systems. These systems are inspired by nature to find an optimal algorithm. Nature has many more years of evolution and experimentation that any algorithm invented by man. These trends have already been incorporated into other areas of science and have given great results.

Among all the possible scenarios that may apply to Internet, I have focused on algorithms for mobile networks (MADHOC). These algorithms are characterized by supporting a one to one communication and are evolving continuously.

To check the difference between protocols I have used the tool NS-2 2.34. This tool is free and open source, and there are many algorithms already implemented for this simulator.

The tests carried out have compared a traditional protocol such as AODV with an ant-based protocol called WAntNet and proposed by J. Baras.

2-Literature Review

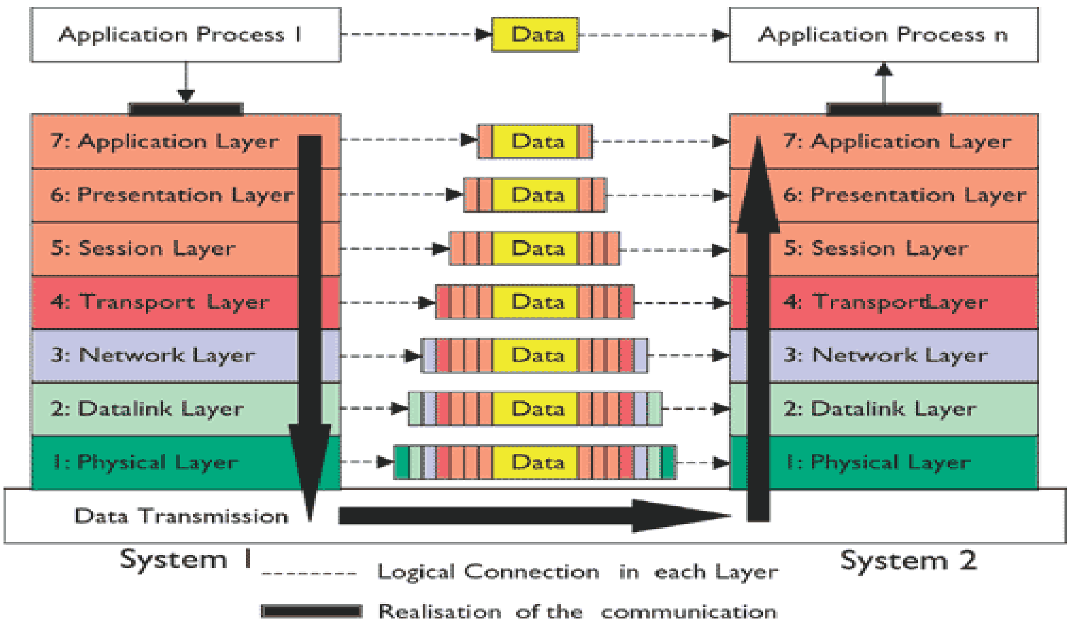
2.1- How the information will be transmitted along Internet?

2.1.1-How the information will be transformed in a signal?

Usually, the people use the computer to do daily works, but only some people understand the internal process that our computer does. It's easier than it looks.

When you use the computer to send an e-mail, surfing the web or looking the Facebook, it starts a sequence of process inside the computer and they finished sending a signal across the network interface of our computer.

To understand this sequence of process, we use the **Open System Interconnection (OSI)** model that describes a logical architecture into seven layers. The name of the seven layers is: Application (Layer 7), Presentation(Layer 6), Session(Layer 5), Transport(Layer 4), Network(Layer 3), Data-Link(Layer 2), and Physical(Layer 1).



Source: http://www.spottek.ca/Tutorials/images/ieb23protocol1.gif

The humans interact with the application software that provides us with a friendly interface.

The data produced in the application layer (layer 7), is deliberated to the low layer named presentation layer.

The presentation layer (layer 6) does the correct format to the data and sometimes compress and encrypted the data. Encapsulates the information and delivery to the low layer named session layer. E.g.: ASCII and SSL

The session layer (layer 5) provides the mechanism for opening, closing and managing a session between end-user application processes. e.g.: NetBIOS As this system layers, this layer encapsulates all the data and delivery to the layer 4 called transport layer.

The transport layer (layer 4) provides reliability of link, flow control, segmentation and error control.eg: TCP, UDP (Not strictly conforming)

The Network layer (layer 3) guarantees the connection end to end between the networks. Can perform the Quality of service, and fragmentation and reassembly of packets. This layer makes possible the Internet. Use logical address. e.g.: IP, ICMP

The Data Link Layer (layer 2) discriminates in MAC (Medium Access Control) and LLC (Logical Link Control). It creates and recognizes the limits of the frames; solve problems of deterioration, loss and duplication of frames. The MAC is concerned with physical addressing, topology of the network, network access, error notification, and flow control. The LLC allow a flow control and acknowledgment mechanisms. e.g.: ARP, Frame Relay

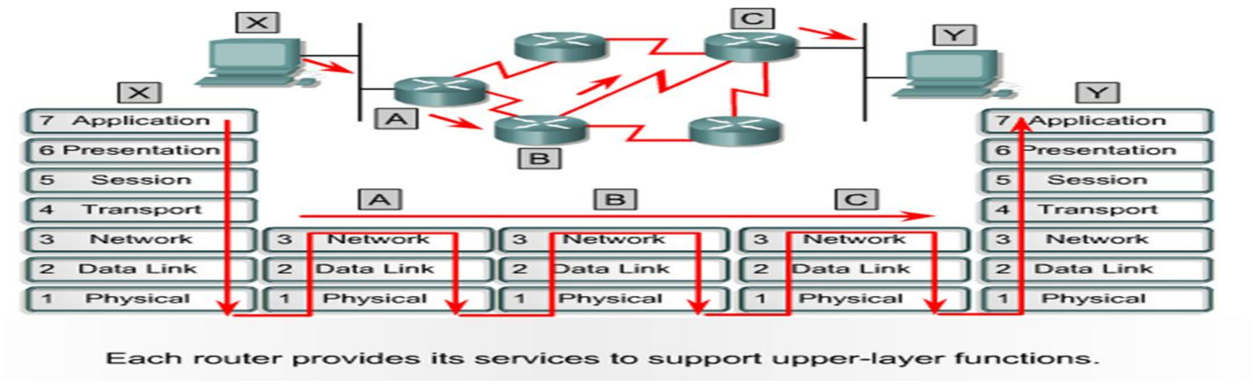
The Physical Layer (layer1) defines the electrical and physical specifications for divides. Define the relationship between a divide and a physical medium like: voltage, cable specifications, etc...e.g.: Ethernet

2.1.2-How the signal will be transmitted along Internet?

The signal that way out of our computer may be an electrical or lighting signal, and it's based in the protocol of Internet (layer 3). Internet is in essential a multitude of interconnected routers.

2.1.3-What is the routing and how it works?

The routing is the act to send packets (layer 3) along the net. The device that does this act is called router. In a normal communication between two internet users the information is forwarded by more of 30 routers. And all the routers do more of 2 possible ways to resend the message. To decide the best way the routers using routers tables. These routing tables will be configured manually or automatically wrote by any routing protocol.



Source: Cisco Systems

2.1.4-IPv4 packet structure

This is the most used protocol in the Internet layer, and essentially the routers read the header and act. Understand all the fields of the header are necessary to understand the routing job.

It was published on the RFC 791 in 1981

It is a best effort delivery model: doesn't guarantee delivery, good sequencing or duplicate packets.

IPv4 use four bytes separated with a dot for addressing.

The IPv4 packet header consists of 13 fields, of which 12 are required. The 13th field is optional (red background in table) and aptly named: options.

bit offset	**0–3**	**4–7**	**8–15**	**16–18**	**19–31**
0	Version	Header length	Differentiated Services	Total Length	
32	Identification			Flags	Fragment Offset
64	Time to Live		Protocol	Header Checksum	
96	Source Address				
128	Destination Address				
160	Options				
160or192+	Data				

Source: http://en.wikipedia.org/wiki/IPv4

2.2-Algorithms used in routing

2.2.1-Types of algorithms

The algorithm is a sequence of steps which when executed in order, performs a function.

Two types of networking algorithms: **off-line and real-time.** The off-line algorithms were used to design a network. The real-time algorithms running in the network devices.

In the real-time algorithms there are three types:

<u>Distance Vector:</u>

Know how many hops distance is the router objective, and in which direction is it.

Update the entire routing table in fixed intervals. This provides a slow convergence and expends a lot of bandwidth. E.g.: RIP

<u>Link State</u>

Know the complete topology of the network. Only update the network changes. Use the costs metrics. Can use a Dijkstra's algorithm. e.g.: OSPF

<u>Hybrids</u>

Is a Distance Vector protocol but use updates like a Link State protocol. Uses complex cost function based on bandwidth, delay, load, reliability, mtu. Uses complex algorithm to share this information around. Then calculates shortest paths by propagating known routes through the network. e.g.: EIGRP

2.2.2-Routing Limitations

The <u>Distance Vector algorithms</u> would be expensive to update because updates the entire IP table. Only knows the neighbour router.

The <u>Link State algorithms</u> only use the bandwidth to calculate the cost metric. Insufficient metrics.

The hybrids algorithms: Are better than DV and LS but isn't perfect. DSPA only calculates optimal routes in isolation, EIGRP appears to use dynamic cost but doesn't

2.2.4-TSP (Travelling Salesman Problem):

The shortest route visiting each member of a collection of locations and returning to your starting point.

Usually:

```
for  i=1 to n
        for
                for
                        loop?
```

Kruskal's algorithm:

Repeat: chose shortest link that doesn't create a loop

Until

Connected

E.g.: OSPF

Prim's algorithm:

Start at A node

Repeat

Connect in the doesn't node

Until

Connected

2.2.5-SPA: Shortest path between two vertexes

Dijkstra’s

Dijkstra's algorithm, also called shortest path algorithm is an algorithm for determining the shortest path given a source vertex to other vertices in a directed graph with weights on each edge. Its name refers to Edsger Dijkstra, who first described it in 1959.
The idea behind this algorithm is to explore all the shortest paths that start from the origin vertex and which lead to all other vertices, and when you get the shortest path from source vertex to the rest of vertices that make up the graph, the algorithm stops. The algorithm is a specialization of the uniform cost search, and as such, does not operate on graphs with negative cost edges (always choosing the node with shorter distance, can be excluded from the search nodes in future iterations would lower the overall cost the road to pass through an edge with negative cost).

Algorithm

Given a weighted directed graph of N nodes are not isolated, the initial node x is a vector D of size N stored at the end of the algorithm the distances from x to all other nodes.
Initialize all distances in D on an infinite value and that are unknown at first, except that of x to be placed at 0 because the distance from x to x would be 0.
Let a = x (as we take the current node).
We travel all nodes adjacent to, except the marked nodes, we call these vi.
If the distance from x to vi stored in D is greater than the distance from x to a plus the distance from a to vi, this is replaced with the second name, that is:
if (Di> Da + d (a, vi)) then Di = Da + d (a, vi),
We mark the node as complete a.
We take as the next current node with the lowest value in D (may be storing the values in a

priority queue) and return to Step 3 until there are unmarked nodes.
Upon completion of the algorithm, D is completely full.

2.3- Ad-hoc networking protocols:

2.3.1-AODV (Ad-Hoc On Demand Distance Vector Routing)

The Ad hoc On Demand Distance Vector (AODV) routing algorithm is a routing protocol designed for ad hoc mobile networks. AODV is capable of both unicast and multicast routing. It is an on demand algorithm, meaning that it builds routes between nodes only as desired by source nodes. It maintains these routes as long as they are needed by the sources. Additionally, AODV forms trees which connect multicast group members. The trees are composed of the group members and the nodes needed to connect the members. AODV uses sequence numbers to ensure the freshness of routes. It is loop-free, self-starting, and scales to large numbers of mobile nodes.

AODV builds routes using a route request / route reply query cycle. When a source node desires a route to a destination for which it does not already have a route, it broadcasts a route request (RREQ) packet across the network. Nodes receiving this packet update their information for the source node and set up backwards pointers to the source node in the route tables. In addition to the source node's IP address, current sequence number, and broadcast ID, the RREQ also contains the most recent sequence number for the destination of which the source node is aware. A node receiving the RREQ may send a route reply (RREP) if it is either the destination or if it has a route to the destination with corresponding sequence number greater than or equal to that contained in the RREQ. If this is the case, it unicast a RREP back to the source. Otherwise, it rebroadcasts the RREQ. Nodes keep track of the RREQ's source IP address and broadcast ID. If they receive a RREQ which they have already processed, they discard the RREQ and do not forward it.

As the RREP propagates back to the source, nodes set up forward pointers to the destination. Once the source node receives the RREP, it may begin to forward data packets to the destination. If the source later receives a RREP containing a greater sequence number or contains the same sequence number with a smaller hop count, it may update its routing information for that destination and begin using the better route.

As long as the route remains active, it will continue to be maintained. A route is considered active as long as there are data packets periodically travelling from the source to the destination along that path. Once the source stops sending data packets, the links will time out and eventually be deleted from the intermediate node routing tables. If a link break occurs while the route is active, the node upstream of the break propagates a route error (RERR) message to the source node to inform it of the now unreachable destination(s). After receiving the RERR, if the source node still desires the route, it can reinitiate route discovery.

Multicast routes are set up in a similar manner. A node wishing to join a multicast group broadcasts a RREQ with the destination IP address set to that of the multicast group and with the 'J'(*join*) flag set to indicate that it would like to join the group. Any node receiving this RREQ that is a member of the multicast tree that has a fresh enough sequence number for the multicast group may send a RREP. As the RREPs propagate back to the source, the nodes forwarding the message set up pointers in their multicast route tables. As the source node receives the RREPs, it keeps track of the route with the freshest sequence number, and beyond that the smallest hop count to the next multicast group member. After the specified discovery period, the source node wills unicast a Multicast Activation (MACT) message to its selected next hop. This message serves the purpose of activating the route. A node that does not receive this message that had set up a multicast route pointer will timeout and delete the pointer. If the node receiving the MACT was not already a part of the multicast tree, it will also have been keeping track of the best route from the RREPs it received. Hence it must also unicast a MACT to its next hop, and so on until a node that was previously a member of the multicast tree is reached.

AODV maintains routes for as long as the route is active. This includes maintaining a multicast tree for the life of the multicast group. Because the network nodes are mobile, it is likely that many link breakages along a route will occur during the lifetime of that route. The papers listed below describe how link breakages are handled. The WMCSA paper describes AODV without multicast but includes detailed simulation results for networks up to 1000 nodes. The Mobicom paper describes AODV's multicast operation and details simulations which show its correct operation. The internet drafts include descriptions of both unicast and multicast route discovery, as well as mentioning how QoS and subnet aggregation can be used with AODV. Finally, the IEEE Personal Communications paper and the Infocom paper details an in-depth study of simulations comparing AODV with the Dynamic Source Routing (DSR) protocol, and examine each protocol's respective strengths and weaknesses.

2.3.2-DSR (Dynamic Source Routing)

The Dynamic Source Routing protocol (DSR) is a simple and efficient routing protocol designed specifically for use in multi-hop wireless ad hoc networks of mobile nodes. DSR allows the network to be completely self-organizing and self-configuring, without the need for any existing network infrastructure or administration.

DSR has been implemented by numerous groups, and deployed on several test beds. Networks using the DSR protocol have been connected to the Internet. DSR can interoperate with Mobile IP, and nodes using Mobile IP and DSR have seamlessly migrated between WLANs, cellular data services, and DSR mobile ad hoc networks.

The protocol is composed of the two main mechanisms of "Route Discovery" and "Route Maintenance", which work together to allow nodes to discover and maintain routes to arbitrary destinations in the ad hoc network. All aspects of the protocol operate entirely on-demand, allowing the routing packet overhead of DSR to scale automatically to only that needed to react to changes in the routes currently in use.

The protocol allows multiple routes to any destination and allows each sender to select and control the routes used in routing its packets, for example for use in load balancing or for increased robustness. Other advantages of the DSR protocol include easily guaranteed loop-free routing, support for use in networks containing unidirectional links, use of only "soft state" in routing, and very rapid recovery when routes in the network change. The

DSR protocol is designed mainly for mobile ad hoc networks of up to about two hundred nodes, and is designed to work well with even very high rates of mobility.

2.3.3-OLSR (Optimized Link State Routing Protocol)

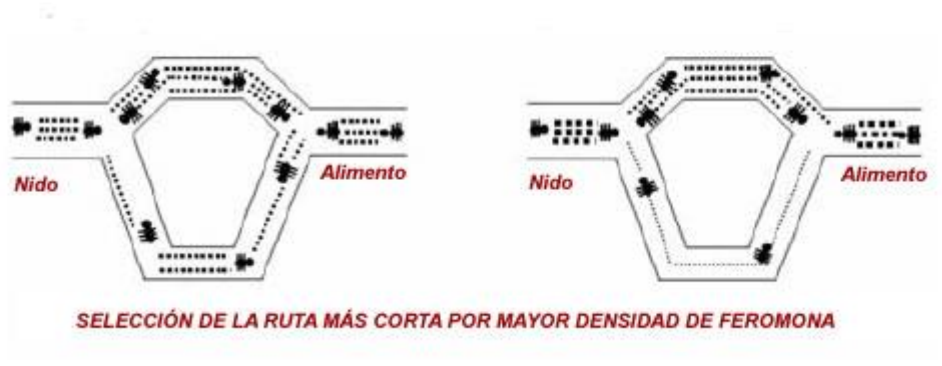
The Optimized Link State Routing Protocol (OLSR) is developed for mobile ad hoc networks. It operates as a table driven and proactive protocol, thus exchanges topology information with other nodes of the network regularly. The nodes which are selected as a multipoint relay (MPR) by some neighbour nodes announce this information periodically in their control messages. Thereby, a node announces to the network, that it has reachability to the nodes which have selected it as MPR. In route calculation, the MPRs are used to form the route from a given node to any destination in the network. The protocol uses the MPRs to facilitate efficient flooding of control messages in the net- work. OLSR inherits the concept of forwarding and relaying from HIPERLAN (a MAC layer protocol) which is standardized by ETSI.

2.4- Protocols biological inspired:

Recently the new networks algorithms were based in a Swarm Intelligence (SI). This intelligence technique studies the collective behaviour in decentralized systems. Not exist a centralized control of the individuals. The individuals interact with other individuals and with the environment producing a global pattern. Examples of these systems are the ant colonies, honey bees, bird flocking, animal herding, etc...

2.4.1-Ant Colony Optimization:

In the conduct of the ants finding differs food, tracing the path short between the nest and location of food. The focus of that search is the deposit pheromone as trail guides travel. Ants prefer follow the highest concentration of pheromone, the same is achieved for the shortest tour made by ants (which for this reason may repeated at most once). In principle routes are random, but who have used the shorter length can return more quickly (for passing at a uniform rate), and repeat the way, leaving a higher density of traces of pheromone. The deposited on the other routes will evaporate, and cease to be relevant in the new routes. This is illustrated in the figure below, adapted from classic images for this topic.


Nido
Alimento
Nido
Alimento
SELECCIÓN DE LA RUTA MÁS CORTA POR MAYOR DENSIDAD DE FEROMONA

2.4.2-The algorithm based on ant colonies

It simulates a colony of artificial ants working in groups and communicating through artificial pheromone trails. Each ant artificial constructs a solution to the problem of crossing a graph construction. Each branch or section of the graph has two types of information guiding the movement of the ant.

•The preference of moving from node r to node s in the edge rs. This preference is denoted by η rs. The value is not changed in the process and is a function of distance between nodes.

•The desirability of movement learned to r s. It simulates the pheromone or trace deposited, and is modified. It is designated by τ rs.

Ant colony optimization code

The basis of the algorithm is the simulation of the reaction exhibits a colony of ants. The substance is called virtual tra as similar to the pheromone. The algorithm is the computational structure below basic.

Top of routine or cycle tracks:

```
Do While (until the criterion is not satisfied)
    Do Until (each ant completes a tour)
        Decision mechanism of the ant
        Update local trail
    End Do
    Updating the global cycling of trace
End Do
```

The decision is based on the intensity of trail present on each way between two adjacent points.

•He who has more tracks has the highest probability of being selected.

•If there is no trace, the election probability is zero.

•If all paths have the same number of traces, the decision is random.

Each ant chooses a route using a decision mechanism. Some algorithms use a local update in testing, reducing the trace amount in the path chosen by the ant (which will less likely to choose the same path as the above). The ant continues the routes of travel between points reaching every point to have visited all, and returns to its point of origin. When back to top, the ant has completed a journey. By completing the trip, I analyze how well it solves the problem. The trail intensity on each route in the journey is fitted through overall upgrade process. The courses that best solve the problem are more trails. Thus, when all the ants have completed a trip and all trips are discussed, as are current traces, the optimization cycle is complete. A new cycle begins and the process repeated. Almost all ants follow the same journey in each cycle and the solution converges. Compared the best solution in the last cycle for the global solution.

2.4.3-Ant colony system

This system (ACS in English) is based on the original proposal for the case of the passenger (Marco Dorigo, Milano, Italy in 1992 and following), who must travel from home all the cities looking for more travel short.

The distance between cities i and j, with coordinates (Xi,Yi) (Xj,Yi)

Respectively, is calculated as:

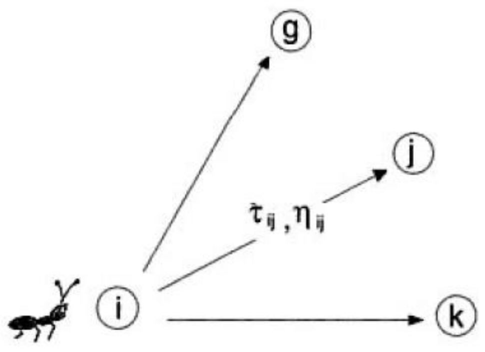
$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Where n is the number of cities, I any city bi(t) the number of ant i at time t, the total number of ants m is:

$$m = \sum_{i=1}^{n} b_i(t)$$

Each ant has the following characteristics:

•Select what will be the next city to be visited. This selection is based on a probability that is a function of the distance city (or visibility) and trace amount present in the stretch connects the cities.



PARÁMETROS DE DECISIÓN

•You report on the cities you have visited (according to a taboo list) and he has not visited.

•When you have completed the journey, or leave a trace substance in each tranche.

♦ **Trail**

The intensity of trail on the road in each section that connects i to j in time t, is denoted by τij(t). At time 0, τij(0) is as small as small positive constant τ0.

At time t, each ant chooses a path and travels to the next city time t +1.

In each iteration there are m movements caused by the m ants (each one making a move) in the interval (t, t +1).

A cycle is defined as n iterations, and means that the m ants have completed a trip.

After each iteration, the algorithm implements a local update:

Reduces the level of trace in the sections selected by the colony in previous iteration. When an ant travels from city j city i, the update rule adjusts the intensity of the stretch of trail connected by

$$\tau_{ij}(t) \leftarrow (1-\varphi)\tau_{ij}(t) + \varphi\tau_0$$

Where φ is an adjustable parameter between 0 and 1 that represents the persistence of the trail. Plus:

$$\tau_0 = \frac{1}{nL_{nn}}$$

Where Lnn length is calculated with the neighbour heuristic algorithm close (it starts in a randomly selected city and following based on the shortest distance to visit all cities).

At the end of each cycle trail intensity is adjusted using a global update. The first step is to calculate the quantity of trace left in each section, including:

$$\Delta\tau_{ij}^{k} = \frac{1}{L_k}$$

Where k represents some ant (1 to m). Y Lk is the length of the cycle k chosen by an ant.

The total value of updating the trail for each section is the sum of the left by each ant quantities you have selected this segment ij, and will:

$$\Delta\tau_{ij} = \sum_{i=1}^{m} \Delta\,\tau_{ij}^{k}$$

The value of the global update is determined by the following expression:

$$\tau_{ij}(t+n) = (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}$$

Where ρ is a constant between 0 and 1, such that (1-ρ) represents the evaporation of trail between t and t + n (the time required to complete a cycle).

♦ **From the visibility**

The system supports the ants have a lot of visibility ηij associated with each tranche, a value that depends on the distance of it (the closest are most preferred that the remote).

$$\eta_{ij} = \frac{1}{d_{ij}}$$

♦ **Probability of preference**

The combination of visibility to the intensity of trail, resulting in:

$$a_{ij}(t) = \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \epsilon permitidas}^{n}[\tau_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}}$$

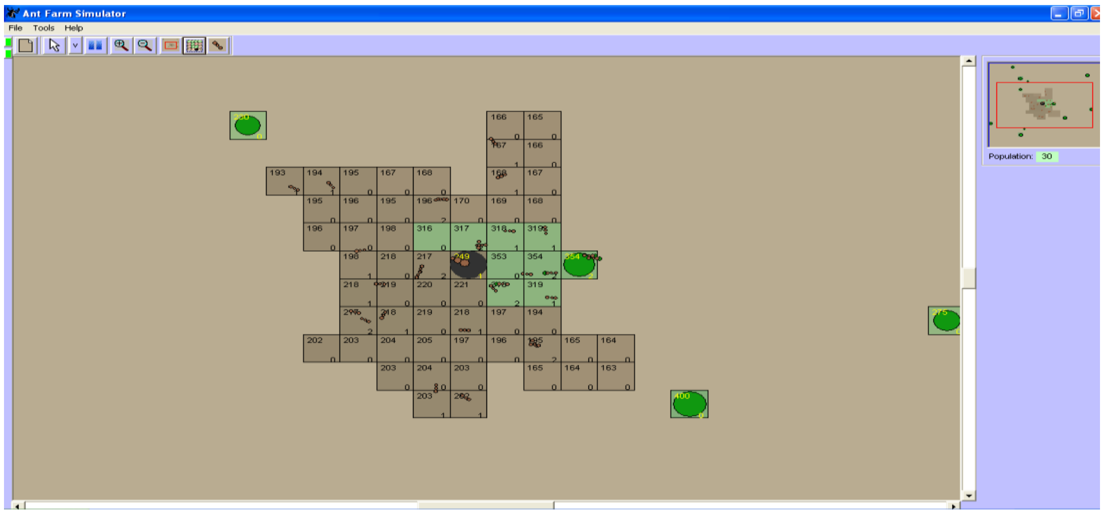
This is the proportion to the neighbouring cities permitted from i City. The parameters α and β are constants used to control importance of the values of local trails and visible.

The probability p that the ant k choose to go from i to j at time t is:

$$p_{ij}^{k}(t) = \frac{a_{ij}(t)}{\sum_{l \epsilon permitidas}^{n} a_{il}(t)}$$

If the ant is not allowed to travel to the city j, then $p^{k}_{ij}=0$.

Example of Ant Farm Simulator:


Ant Farm Simulator
File
Tools
Help
Population: 30

2.4.4-AntHocNet4 algorithm: *Based in ACO (Ant Colony Optimization)*

AntHocNet is the name of one adaptive routing algorithm for Mobile Ad hoc NETworks (MANET) inspired in the Ant Colony Optimization (ACO).

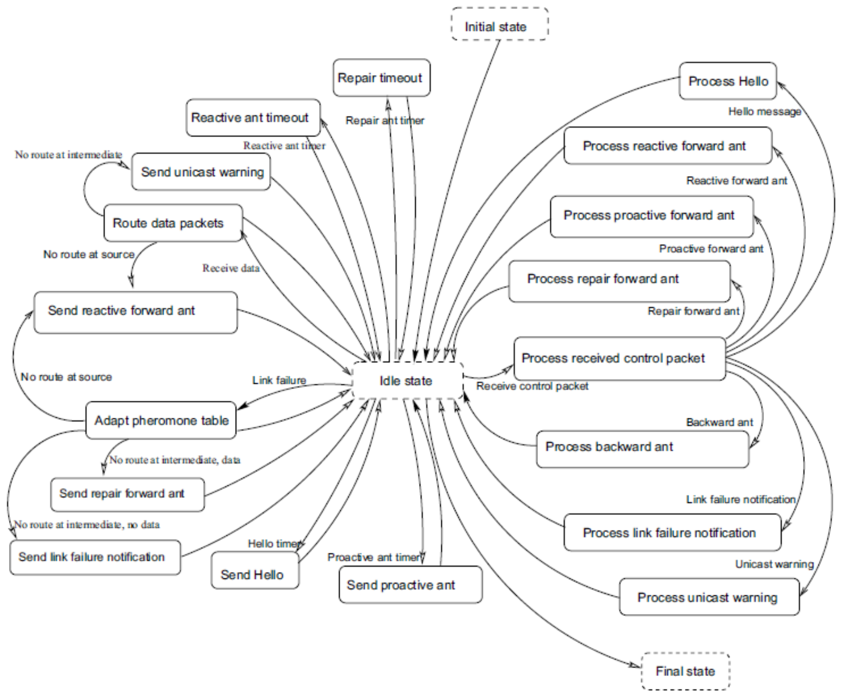
AntHocNet is a hybrid algorithm because combine a reactive and proactive routing strategies.

It’s reactive because doesn’t try to maintain routing information between all the nodes in the network, only update info in the end nodes.

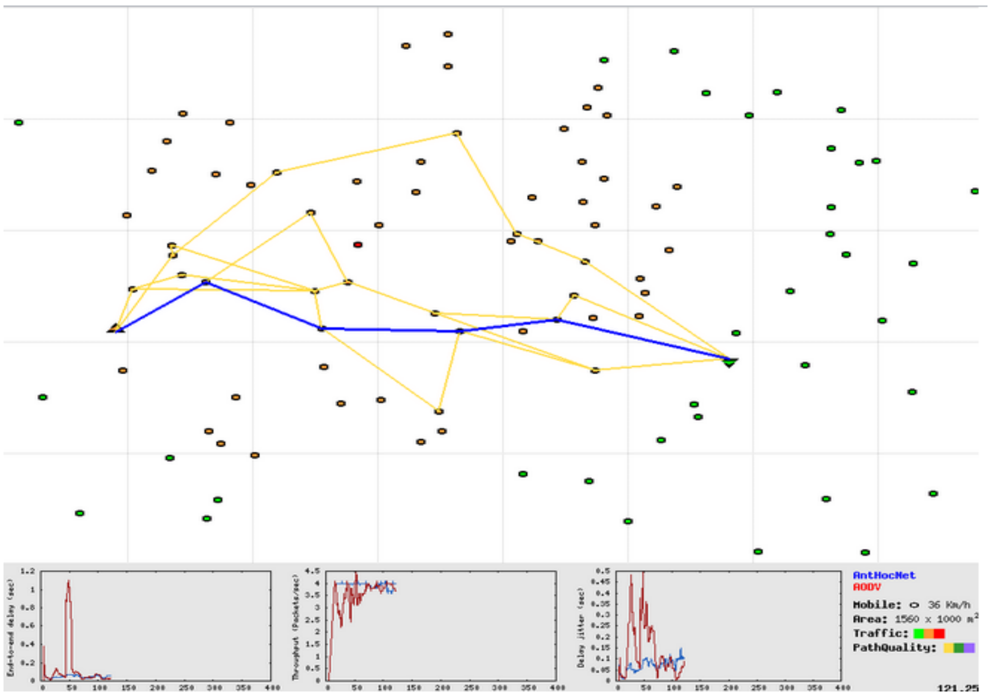
It’s proactive because have information of the network using two complementary methods. One process updates data along the end path using the artificial ants. The other process used the pheromone diffusion to update information along the net. This way to update information is not essentially ACO, but is used in Bellman-Ford routing algorithms.

The combination of two processes does an efficient, adaptative and robust system especially in highly dynamic environments.

A finite state machine representation of the AntHocNet routing algorithm:

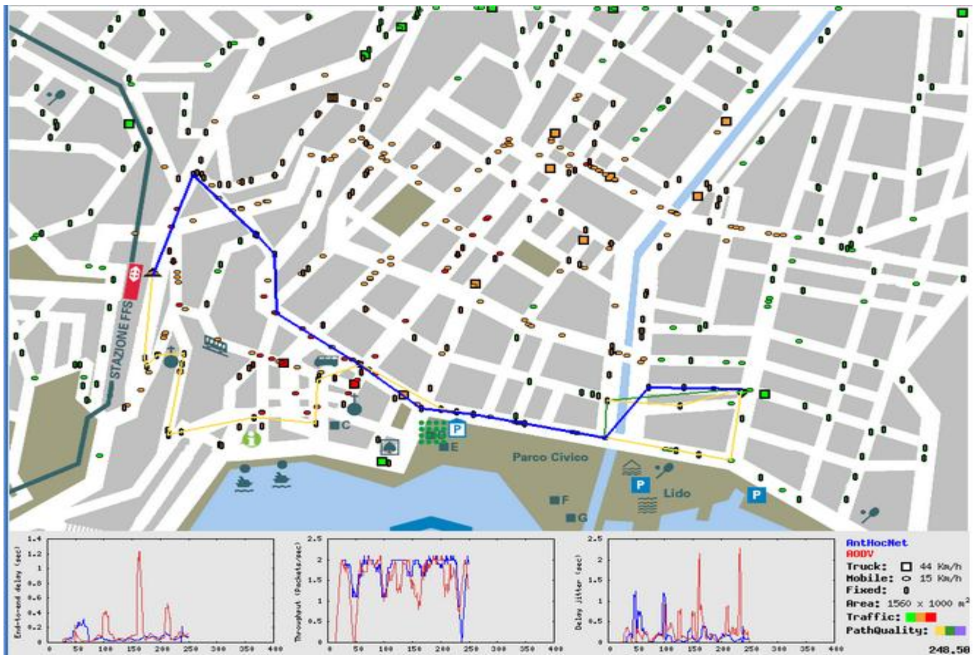


AntHocNet at work in an open space scenario:



Source: http://www.idsia.ch/~frederick/anthocnet/OpenSpace.gif

AntHocNet at work in an urban scenario:



Source: http://www.idsia.ch/~frederick/anthocnet/Urban.gif

2.4.5-Artificial Bee Colony (ABC) Algorithm

This algorithm was defined by Davis Karaboga in 2005. In essential this algorithm is based in the searching of food sources by the bees. The artificial bees flying in a multidimensional area searching food. One type of bees flying randomly searching the food (scouts), when it's discovered memorizes the position. Other beers named onlooker bees look the source of food and going to eat. Every time the bees searching the source of highest nectar. When it's discovered, memorize the new position and forget the previous one.

This algorithm combines local search methods (using the employed an onlooker beers), and a global searching (using the onlookers and scouts). The effect is a balance between exploration and exploitation.

Detailed Pseudocode of the ABC Algorithm

```
1: Initialize the population of solutions xi,j
2: Evaluate the population
3: cycle=1
4: repeat
5: Produce new solutions (food source positions) υi,j in the neighbourhood of xi,j for the employed bees using the formula υi,j = xi,j + Φij(xi,j - xk,j) (k is a solution in the neighbourhood of i, Φ is a random number in the range [-1,1] )and evaluate them.
6: Apply the greedy selection process between xi and υi
7: Calculate the probability values Pi for the solutions xi by means of their fitness values using the equation (1)
```

$$P_i = \frac{fit_i}{\sum_{i=1}^{SN} fit_i}$$

In order to calculate the fitness values of solutions I employed the following equation :

$$fit_i = \begin{Bmatrix} \frac{1}{1+f_i} & if\ f_i \geq 0 \\ 1+abs(f_i) & if\ f_i < 0 \end{Bmatrix}$$

Normalize Pi values into [0,1]

8: Produce the new solutions (new positions) υi for the onlookers from the solutions xi, selected depending on Pi, and evaluate them

9: Apply the greedy selection process for the onlookers between xi and υi

10: Determine the abandoned solution (source), if exists, and replace it with a new randomly produced solution xi for the scout using the equation (3)

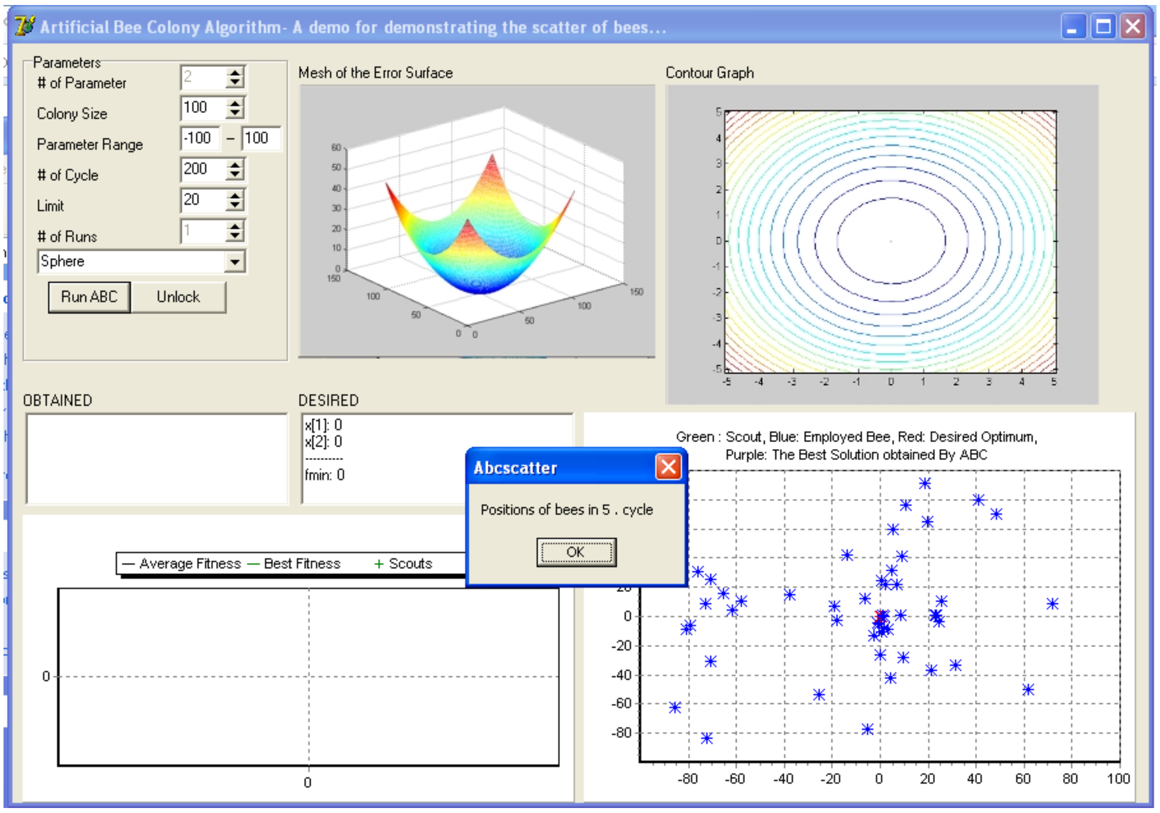
xij=minj+rand(0,1)*(maxj-minj) (3)

11: Memorize the best food source position (solution) achieved so far

12: cycle=cycle+1

13: until cycle= Maximum Cycle Number (MCN)

Screenshot of the program Artificial Bee Colony Algorithm


Artificial Bee Colony Algorithm- A demo for demonstrating the scatter of bees...

Parameters
# of Parameter 2
Colony Size 100
Parameter Range -100 – 100
# of Cycle 200
Limit 20
# of Runs 1
Sphere

OBTAINED

DESIRED
x[1]: 0
x[2]: 0
----
fmin: 0

Abcscatter
Positions of bees in 5 . cycle

Green : Scout, Blue: Employed Bee, Red: Desired Optimum,
Purple: The Best Solution obtained By ABC

Source: http://mf.erciyes.edu.tr/abc/pub/ABCScatter.exe

3-Design of Ant Algorithm

The first step in any project must to be the research of the information in the area of study.

In my case, the first step was the research of information about the protocols used in the routing and check witch algorithms are based each one. The protocols studied are AODV, DSR, OLSR, AntHocNet and ABC.

The second step was finding information about bioinspired algorithms, specifically about Swarm Intelligence. Swarm Intelligence describes the collective behaviour of decentralized, self-organized systems. Among all possible types of use of collective intelligence, I chose those related to the organization that has ants and bees.

The way they communicate with each other ants to find food was the inspiration for Marco Dorigo in 1992 to publish his thesis focused on finding the ant colony optimization algorithm (ACO). ACO is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs.

The bee's algorithm is a population-based search algorithm first developed in 2005. It mimics the food foraging behaviour of swarms of honey bees. In its basic version, the algorithm performs a kind of neighbourhood search combined with random search and can be used for both combinatorial optimisation and functional optimization.

To do the implementation and test, I used the Network Simulator (NS-2.34) because this is an open source program with a lot of libraries and with an easy programming in C++ or TCL. I discard other simulators (QualNet, OPNET and GloMoSim) because they are not free or have a difficult implementation.

There are a some algorithms based in Ant Colony Optimization. The most important are PERA, AntHocNet and ARA. To do my test I used the code of AntNet (J.Baras) and make some modifications needed because AntNet has been designed for fixed networks and I will use it in ad-hoc networks, where the topology is highly dynamic.

The test scenarios are a mobile wireless scenario and a large scenario described on chapter 5.1.1 and 5.1.2

3.1-Algorithm Description

The WAntNet algorithm, which is basically the same as the AntNet algorithm1, can be described in detail as follows:

1. At regular intervals, from every network node s, a forward ant $F_{s\rightarrow d}$ is launched towards a randomly selected destination node d. Destinations are chosen to match the current traffic patterns i.e., if f_{sd} is a measure (in bits or in the number of packets) of the data flow $s\rightarrow d$, then the probability y_d of creating at node s a forward ant with node d as destination is:

$$y_d = \frac{f_{sd}}{\sum_{d'=1}^{N} f_{sd'}}$$

2. While travelling towards their destination nodes, the forward ants store the address of each visited node N_k and the departure time to the next hop in a memory stack. Forward ants share the same queues as data packets, so they experience the same traffic delays.

3. At each node k, each forward ant chooses the next node as follows:

   • If not all the neighbouring nodes have been visited, then the next neighbour is chosen among the nodes that have not been visited as:

$$P'_{nd} = \frac{P_{nd} + \ \alpha l_n}{1 + \ \alpha \ (|N_k| - 1)}$$

   N_k represents the set of neighbours of the current node k and $|N_k|$ the cardinality of that set, i.e., the number of neighbours while the heuristic correction l_n is a normalized value [0, 1] such that $1-l_n$ is proportional to the length q_n of the queue of the link connecting the node k with its neighbour n:

$$l_n = 1 - \frac{q_n}{\sum_{n'=1}^{|N_k|} q_{n'}}$$

The value of α weighs the importance of the instantaneous state of the node's queue with respect to the probability values stored in the routing table.

• If all the neighbouring nodes have been visited previously, then the next node is chosen uniformly among all the neighbours. In this case, since all the neighbours have been already visited, the forward ant is forced to return to a previously visited node. Thus, irrespective of which neighbour is chosen as the next node, the forward ant is in a loop (cycle).

• With some probability ε the next node may be also chosen uniformly among all the neighbouring nodes. The parameter ε is deliberately incorporated in W AntNet to overcome the problem where one of the entries in the routing table is almost unity, while the other are vanishingly small. In such a situation, the forward ants always choose the same link and thus stop exploring the network for other routes. The parameter ε ensures that the network is being constantly explored, though it introduces an element of inefficiency in the algorithm.

4. If a cycle is detected, that is, if the ant is forced to return to an already visited node, the cycle's nodes are popped from the ant's stack and all memory about the cycle is destroyed. If the cycle lasted longer than the lifetime of the forward ant before entering the cycle, the ant is destroyed. The lifetime of a forward ant is defined as the total time since the forward ant was generated.

5. When the destination node d is reached, the forward ant $F_{s\rightarrow d}$ generates a backward ant $B_{d\rightarrow s}$. The forward ant transfers all the memory contained in the stack $S_{s\rightarrow d}$ to the backward ant, and dies.

6. The backward ant takes the same path as the corresponding forward ant, but in the opposite direction. At each node k, the backward ant pops the stack $S_{s-!d}$ to move to the next node. Backward ants do not share the same queues as data packets and forward ants; they use high priority queues to quickly propagate to the routing tables the information collected by the forward ants.

7. Arriving at a node k coming from a neighbour node h, the backward ant updates the two main data structures of the node, the local model of the traffic M_k and the routing table T_k, for all the entries corresponding to the destination node d.

• The mean μ_d and variance σ_d^2 entries in the local model of traffic M_k are modified. The best value t_{bestd} of the forward ants trip time from node k to the destination d stored in the moving observation window W_d is also updated by the backward ant. If the newly observed forward ant's trip time $t_{k\rightarrow d}$ from the node k to the destination d is less then t_{bestd}, then t_{bestd} is replaced by $t_{k\rightarrow d}$.

• The routing table *Tk* is changed by incrementing the probability $p_{hd'}$ (i.e., the probability of choosing neighbour *h* when destination is d_0) and decrementing, by normalization, the other probabilities $p_{nd'}$. The probability $p_{hd'}$ is increased by the reinforcement value *r* as:

$$p_{hd'} \leftarrow p_{hd'} + r\,(1 - p_{hd'})$$

The probabilities $p_{nd'}$ of the other neighbouring nodes n for destination d' are decreased by the negative reinforcement as:

$$p_{nd'} \leftarrow p_{nd'} - rp_{nd'}\ , \forall n \neq h,\ n \in N_k$$

Thus, in W AntNet, every path found by the forward ants receives a positive reinforcement.

• The reinforcement value r is a dimensionless constant (0, 1] and is calculated as:

$$r = c_1 \frac{t_{best_d}}{t_{k \longrightarrow d}} + c_2 \frac{t_{sup} - t_{best_d}}{(t_{sup} - t_{best_d}) + (t_{k \longrightarrow d} - t_{best_d})}$$

$t_{k \to d}$ is the newly observed forward ant's trip time from node *k* to the destination *d* and $t_{best d}$ is the best trip time experienced by the forward ants travelling towards the destination d over the observation window W_d. The value of t_{sup} is calculated as:

$$t_{sup} = \mu_d + \frac{\sigma_d}{\sqrt{1-\gamma}\sqrt{|W_{max}|}}$$

where γ is the confidence level 2 In my experiments, it was set to = 0.95. Represents the upper limit of the confidence interval for the mean μ_d, assuming that the mean μ_d and the variance

σ_d^2 are estimated over W_{max} samples.

The first term evaluates the ratio between the current trip time and the best trip time observed over the moving observation window. The second term is a correction factor and indicates how far the value of $t_{k \to d}$ is from $t_{best d}$ in relation to the extension of the confidence interval. The values of c_1 and c_2 indicate the relative importance of each term. It is logical to assume that is more important than the second term. Hence, the value of c_1 should be chosen larger than the value of c_2. The value *r* calculated in is finally transformed by means of a squash function *s(x)* defined by:

$$s(x) = \frac{1}{1 + \exp\left(\frac{a}{x|N_k|}\right)}\ , \text{ where } x \in (0,1],\ a \in \mathbb{R}^+$$

$$r \longleftarrow \frac{s(r)}{s(1)}$$

The squash function *s(x)* is introduced in the AntNet algorithm so that small values of r would have negligible effect in updating the routing tables. Due to the squash function *s(x)*, the low values of *r* are reduced further, and therefore do not contribute in the update of routing tables. The coefficient $\frac{a}{N^k}$ determines the dependence of squash function s(x) on the number of neighbours N_k of the node k. If the value of coefficient $\frac{a}{N^k}$ is less than 1, then even low values of r get incremented due to the squash function *s(x).* Thus, the value of parameter a should be chosen such that the coefficient $\frac{a}{N^k}$ is greater than 1.

In my experiments, I set them to c1 = 0.7 and c2 = 0.3 respectively.

Data packets use different routing tables than the forward ants for travelling from the source node to the destination node. The routing table values for data packets are obtained by re-mapping the routing table entries used by forward ants by means of a power function $g(v) = v^{\beta}, \beta > 1$ that emphasizes the high probability values and reduces the lower ones. Thus, I prevent data packets from choosing links with very low probability. In addition, data packets have a fixed time to live (TTL). If they do not arrive at the destination within the TTL, they are dropped.

3.2-Packet classes:

3.2.1-Forward ants (FANT):

Used in the “Route Discovery Phase” to update the routing tables and distribute information about the traffic load in the network.

This kind of packets establishes the pheromone track to the source node. These packets are sending via broadcast to all its neighbours. Each packet has a different number to be identified. When any node receive for first time a FANT record in his table the “destination address”, “next hop” and “pheromone value”.

The node interprets the sender's address as “destination address". The address of the previous node's "next hop". Calculate the value of pheromone depending on the number of hops remaining to your destination. Then the packet is sent to its neighbours. If the package does not find its destination, the packet is sent to the source node and there is destroyed.

3.2.2-Backward ants (BANT)

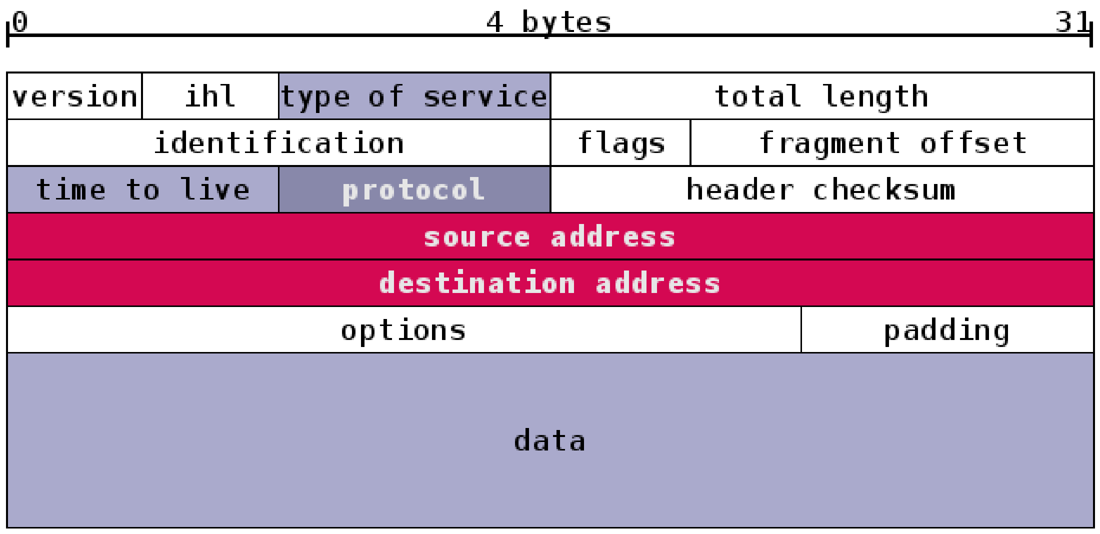
Used in the “Route Discovery Phase”. This kind of packets establishes the pheromone track to the destination node. When a FANT packet arrive to his destine, a BANT packet is created and sent to the source node. In this state, the path is established a data packets can be sent.

3.2.3-Neighbour control parquets

They are used to maintain the list of available nodes. These packages are HELLO messages that are sent via regular broadcast to all neighbouring nodes to indicate the presence of the source node. These packages only contain the information from the source address and an ID number to distinguish it from other agents.

3.2.4-Data packets

These packets contain the information that the end user wishes to transmit from one point to another. They contain no information concerning the routing protocol used. IP packets are called datagram, and its structure is as follows:



Routing agents, both forward and backward ants, contain, in addition to the IP header, the following data fields:

Final destination address towards where the original forward ant was sent. It is necessary to check if the ant has arrived or not, as the IP destination address will change at every visited node.

Birthtime is the time when the ant has been generated.

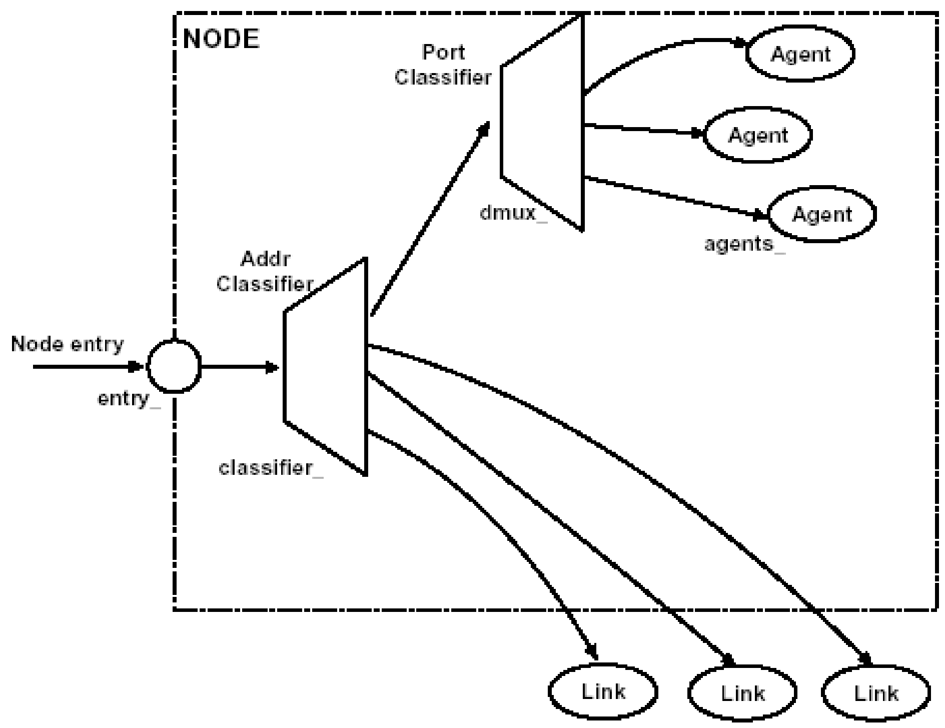
Arrival time to the final destination. This value is used to calculate the trip time.

Memory stacks with the addresses of the visited nodes and the departure time towards the next hop.

3.3-Node

3.3.1-Logical node structure:

The logical structure of a node in Network simulation consists in two TclObjects: an address classifier (classifer_) and a port classifier (dmux_). The function of these classifiers is to distribute incoming packets to the correct agent or outgoing link.



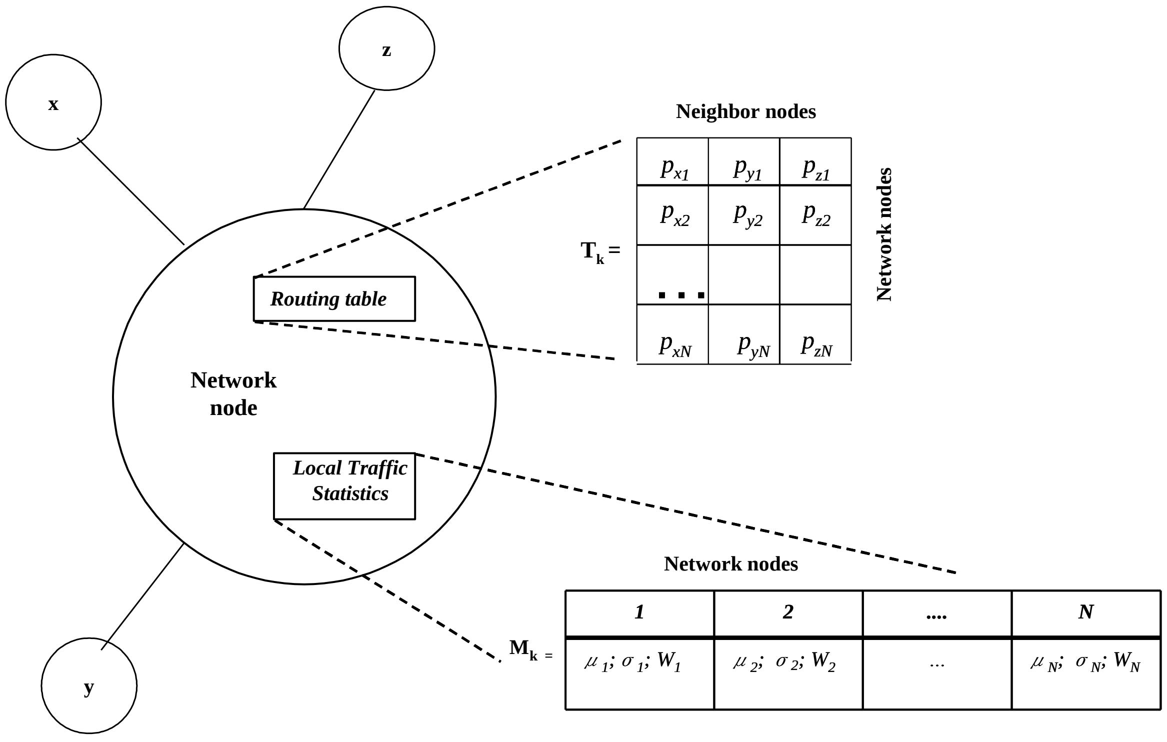
In addition of the typical structure of the node in NS-2 there are an input buffer composed of a single queue and an output buffer composed of a high priority queue and a low priority queue for each neighbour or outgoing link.

The operation of these buffers is as follows. When the node receives a packet from a neighbour node, stores it in the input buffer is FIFO. Then the packet goes to the output buffer where you can find two types of priorities: High priority and low priority. In the high priority, all packages will create that need to be routed quickly such as Forward ants. In the high priority packets will Backward ants and data packets. The buffers have a finite storage capacity. The buffers of the nodes in real life have a storage capacity, but the nodes in NS-2 I can configure according to the needs of each test. In all cases there is a maximum capacity exceeded if the packets are lost. Place a large buffer maximum size will save us problems because our protocol sends a large amount of control packets.

3.3.2-Data structure for a node with neighbours

In the example node is supposed that this node have tree neighbour nodes called x, y z.

Each node with neighbours in her data structure must have two different tables. Routing table (Tk) and another table with the statistics named Local Traffic Statistics (Mk)



p_{x1}	p_{y1}	p_{z1}
p_{x2}	p_{y2}	p_{z2}
. . .		
p_{xN}	p_{yN}	p_{zN}

1	2	....	N
$\mu_1; \sigma_1; W_1$	$\mu_2; \sigma_2; W_2$	...	$\mu_N; \sigma_N; W_N$

Routing table:

In the routing table appears the probability to arrive at each neighbour node between each network.

Local Traffic Statistics:

This table contains the statistics about the network topology and traffic distribution by means of the measured delay.

The model is adaptative and described by means and variances computed over the trip times experienced by the agents. A windowing mechanism is used to limit the number of samples used. For each destination d in the network, the table *Mk* contains a moving observation window *Wd*, an estimated mean μd and an estimated variance σ_d^2.

4-Implementation of protocol WAntNet

4.1-Installing the environment

The simulator NS-2 works under Linux, Mac Os, and a shortened version for the Windows emulator Cygwyn.

I have chosen to install the Linux as the operating system that supports more versions of NS-2 and allows full implementation of all its features.

The operating system has been chosen ubuntu-9.10-desktop-i386. The operating system of choice has proven to be robust and has many support manuals.

The selected version of NS2 has been the release 2.34. This improves the previous version had a bug that did not allow its full functionality. Later versions have also been more volatile this being the most valued by developers.

To install the NS2 is necessary to download the code (http://www.isi.edu/nsnam/ns/ns-build.html), and then add a series of libraries that will be useful in the performance of our code.

Then, I download the code of Ant Net of Lavina from http://code.google.com/p/antalgorithm/downloads/list

The code of AODV is included in the NS-2 package.

4.2-Code

Based on the study by J. Baras I have generated a set of libraries written in C + + to create the algorithm WAntNet. The scripts will be modified to be applied in a mobile scenarios generating pauses and reconnect in the different scenarios. TheAll of this codes are in the appendices.

The scripts are the next:

WAntNet.cc

WAntNet.h

WAntNet-packet.cc

WAntNet-packet.h

WAntNet-rtable.cc

WAntnet-ltm.cc

Ndp.cc

Ndp.h

To make the test more easy I had to create some scripts. These escripts keep the parameters used in the test and collect the solutions for further evaluation.

The scripts are the next:

config-experiment.sh:

The script config-experiment.sh creates a folder tree and a template config file. This file should be edited to set all the parameters of the experiment:

```
$config-experiment.sh [experiment]

$vim [experiment]/config
```

All the parameters and settings are self-explained in the config file and will be discussed later in this section.

run-experiment.sh:

If a similar experiment has to be run, one can use the script copy-experiment.sh, which copy the movement and traffic patterns and configuration file from the experiment passed as first argument to the second one. Then, it asks the user to change whatever needed. Usage:

```
$copy-experiment.sh [source experiment] [new experiment]
```

average-plots.sh:

In this scripts the information of the plots have been saved.

gen-gnuplot-commands.sh:

This script generate the commands to show the graphical plot.

purge-plots.sh:

This script purge all the data saved in the plots.

optimal-hop-count.sh

The counting to the optimal hop have been saved in this script.

error-rate-plot.sh:

The data of error rate of the plots have been saved in this script.

5-Evaluation and simulations

5.1-Experiments

I have made many experiments to evaluate the protocols AODV and WAntNet each other. The most important test I have been made on a wireless mobile scenario of reduced dimensions, and performed on a stage large (2km x 2km)

The initial parameters used are:

Nodes =	50
Area =	1500x300m2
Speed =	1 − 20m/s
CBR sources =	10 20 and 30
Packet size =	64bytes
Data rate =	4.0pkts/s
Pausetimes =	0 30 60 120 300 600 900 s

Channel type :	WirelessChannel
Radio propagation model :	TwoRayGround
Network interface type :	WirelessPhy
MAC type :	802.11
MAC data rate :	2Mbps

Time between retransmitted Route Requests:	500ms
Size of source route header carrying n addresses :	4n + 4bytes
Timeout for non propagating search :	30ms
Time to hold packets awaiting routes :	30s
Max rate for sending gratuitous Replys for a route :	1/s

Time for which a route is considered active :	300s
Lifetime on a Route Reply sent by destination node :	600s
Number of times a Route Request is retried :	3
Time before a Route Request is retried :	6s
Time for which the broadcast id for a forwarded Route Request is kept :	3s
Time for which the reverse route information for a Route Reply is kept :	3s
Time before broken link is deleted from routing table :	3s
MAC layer link breakage detection :	on

Time of simulation end :	900
X dimension of topography :	1500
Y dimension of topography :	300
Number of mobile nodes :	50
Ad-hoc routing protocol :	W AntNet
Scenario file :	nn50-1500x300-m1-M20-s8-p600.scen
Connection pattern file :	nn50-mc20-r4.0.cbr
Trace output file :	nn50-mc20-r4.0-1500x300-m1-M20-s8-p600.tr

5.1.1-Mobile wireless scenario

To perform this test I built a mobile stage of an area of 1500m x300m. To simulate the motion of the nodes I have used a feature called setdest created by EECS from the University of Michigan.

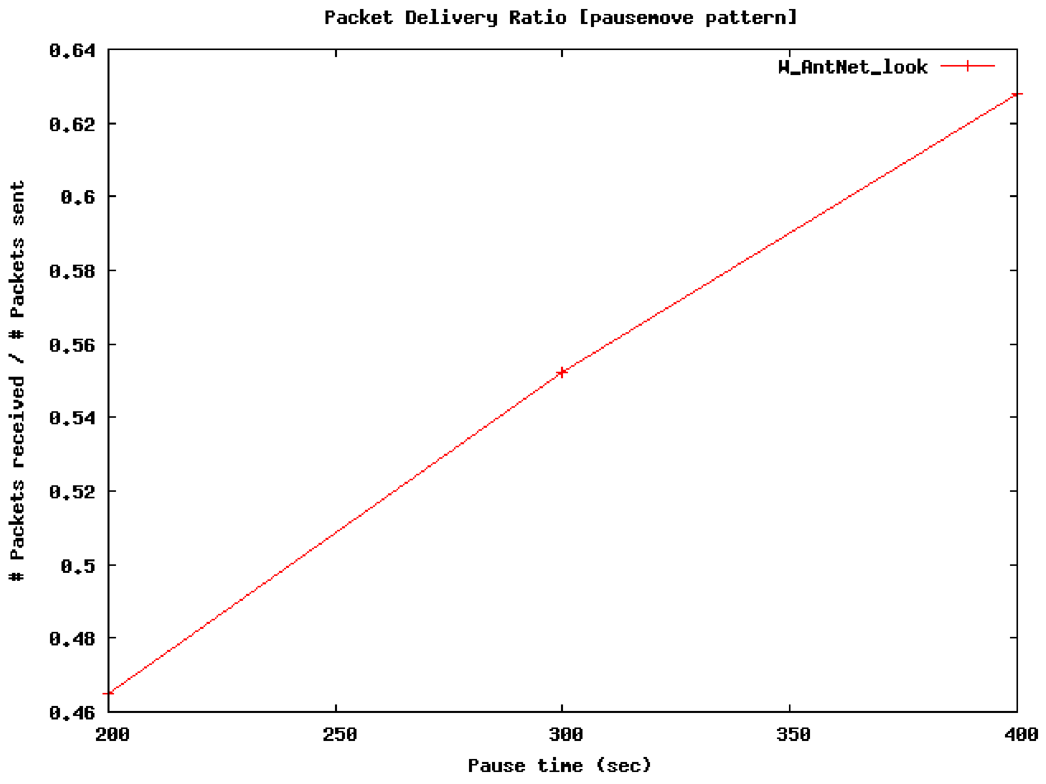
To give a more realistic approach I tried with different pause times and movement. At the time of pause nodes stay static while moving at the time of the nodes are moving.

I tried the following:

Pause: 200s y move: 300s

Pause: 300s y move: 200s

Pause: 400s y move: 100s



In the graph we can see that the packages that time exceeding 300 seconds are fewer delivered. This is because the algorithm needs more time to converge and find a suitable path.

ForwardAnts only knew neighbouring nodes:

In previous experiments have found that sending a large amount of ForwardAnts (control packets that are reactively route) makes the protocol significantly decrease performance.

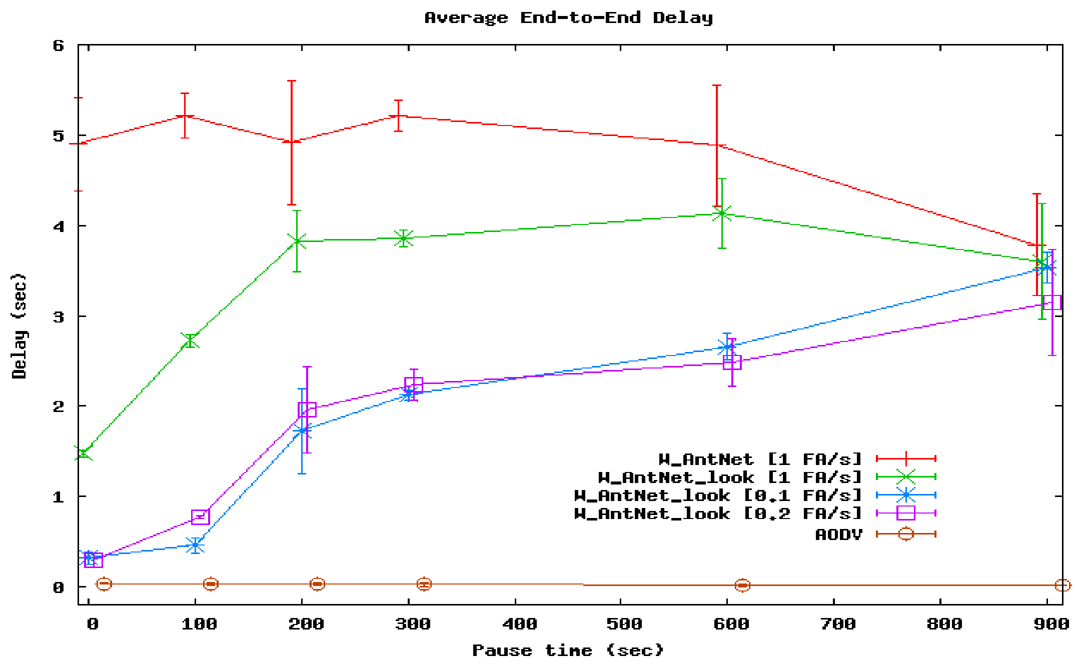
To prove I do an experiment only sent to neighbouring known nodes ForwardAnts (lookahead). I have done this experiment with different shipping rates Forward Ants per second.

The tests were the following values:

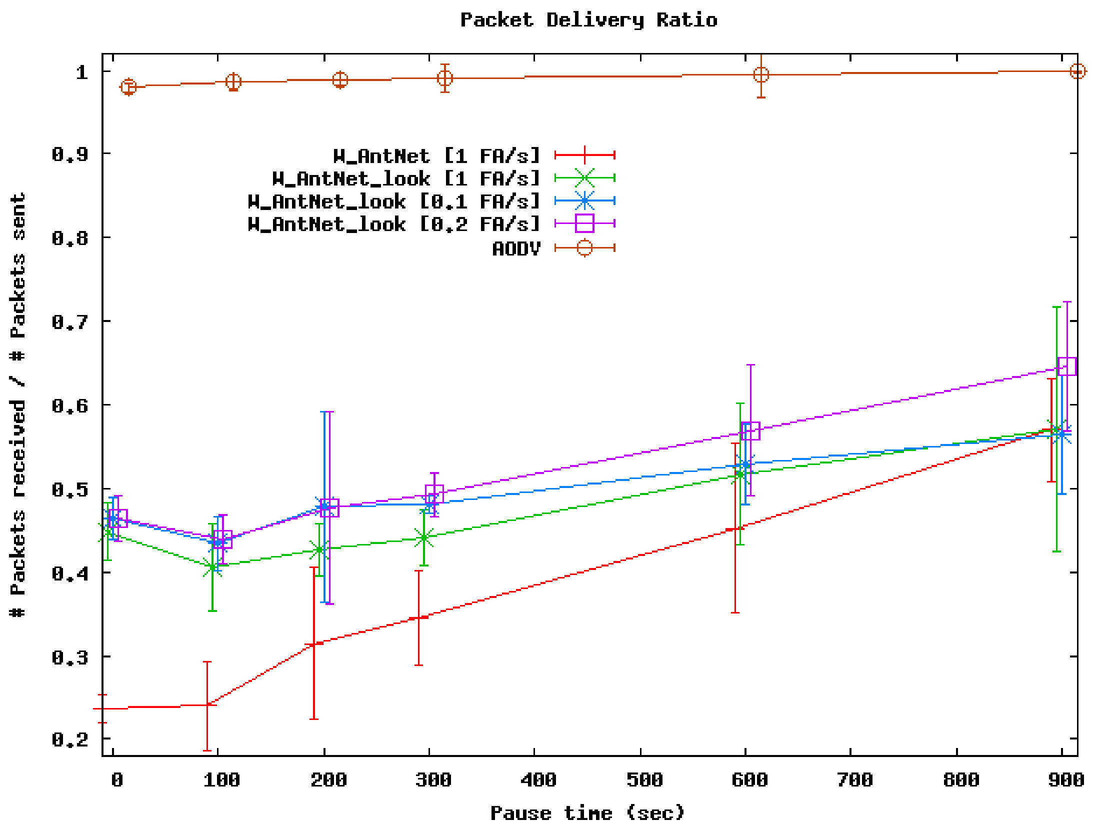
1 ForwardAnt / second

0.1 ForwardAnt / second

0.2 ForwardAnt / second



In this graph we can observe que inicialmente el protocol W_AntNet has more delay because in the first moments flood the network with broadcast packets sent to all routers. If we lower the proportion of packets sent per second to 0.2 FA / s get better performance but far from the performance observed by AODV.



In this graph we can see that initially W_AntNet based protocols lose many packets. If we send forward ants only to neighboring routers, we get better results, almost to converge to 900sec.

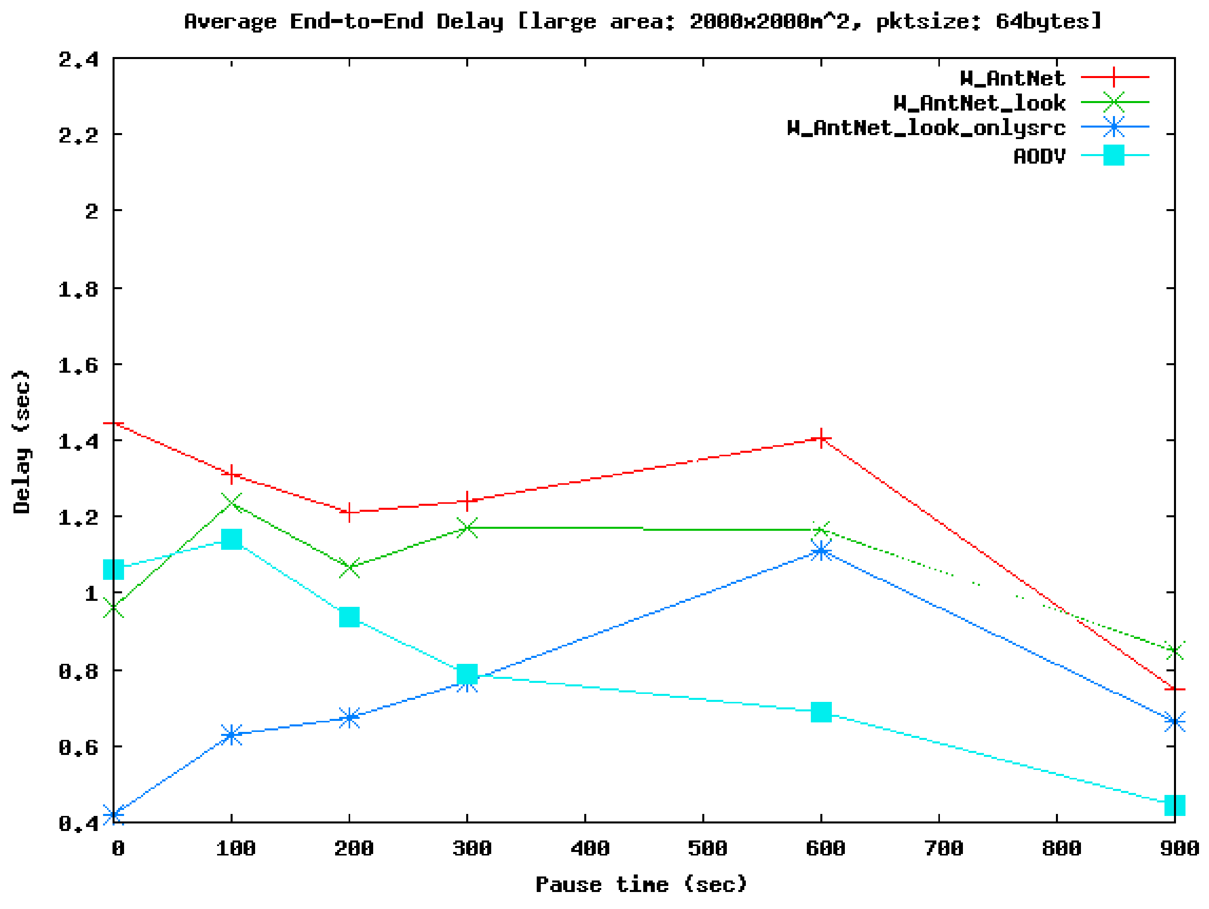
We can see how the protocol performance WAntNet using very small shipping rates Forward Ants (0.2 FA / S) is very similar to the performance of AODV.

But anyway, AODV remains clearly superior in this aspect.

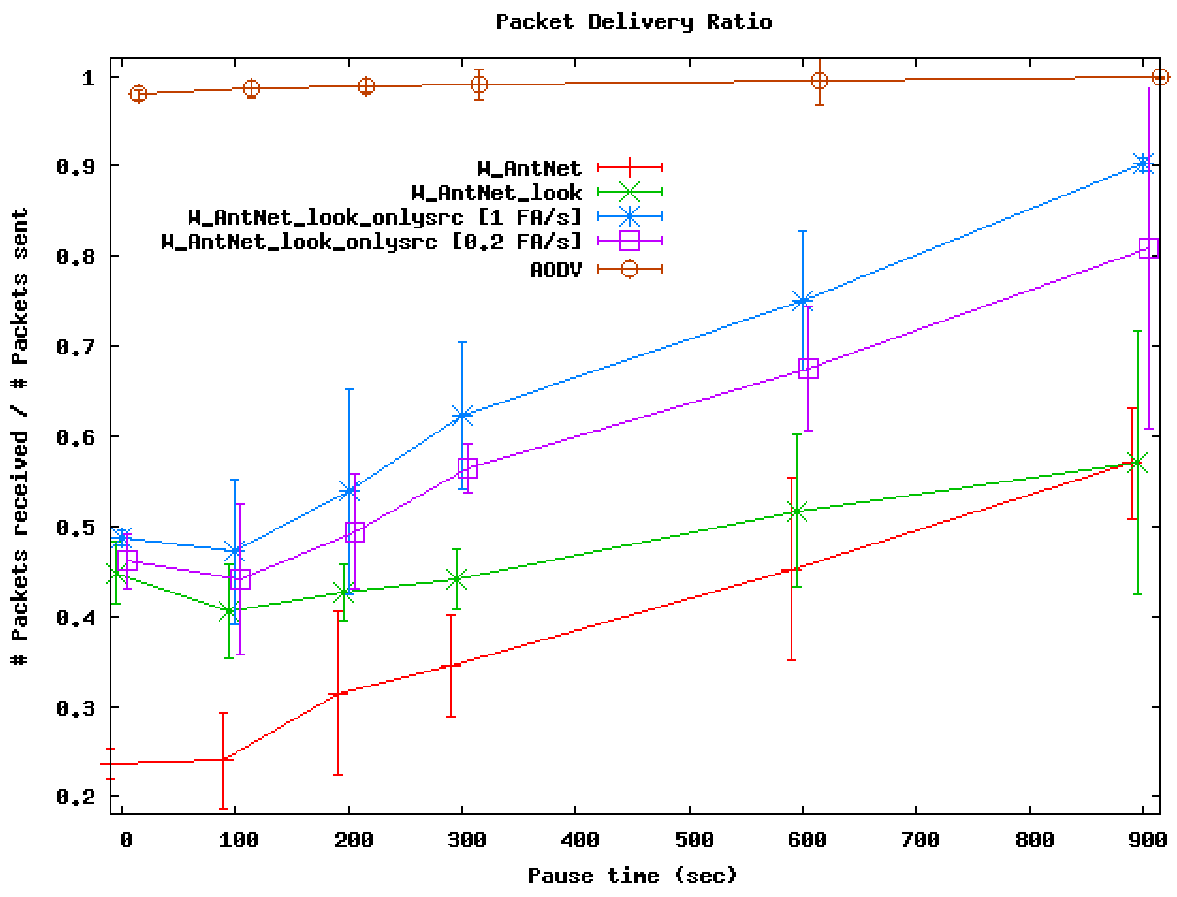
5.1.2-Large scenario

This scenario keeps the initial values described earlier in this chapter but the area in which 50 nodes are distributed is 2km x 2km.

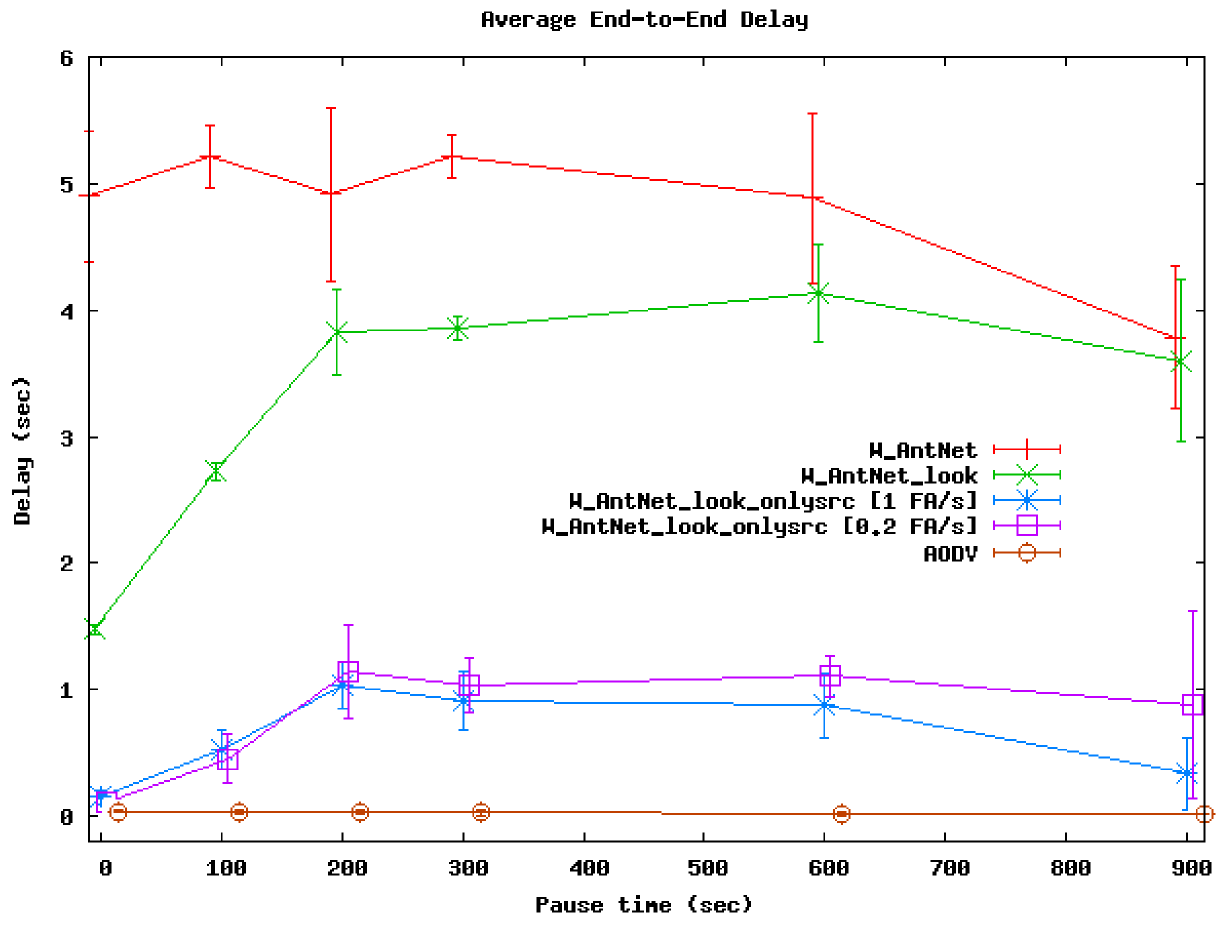
We can observe when the nodes are in motion has a greater delay than when they are static, but the package delivery is more effective. The fact that delivery is more effective when the nodes are in motion and explained that failure to find the destination and this is still never finding it. In the other side if it is not a destination, and this change of position, sooner or later find it.



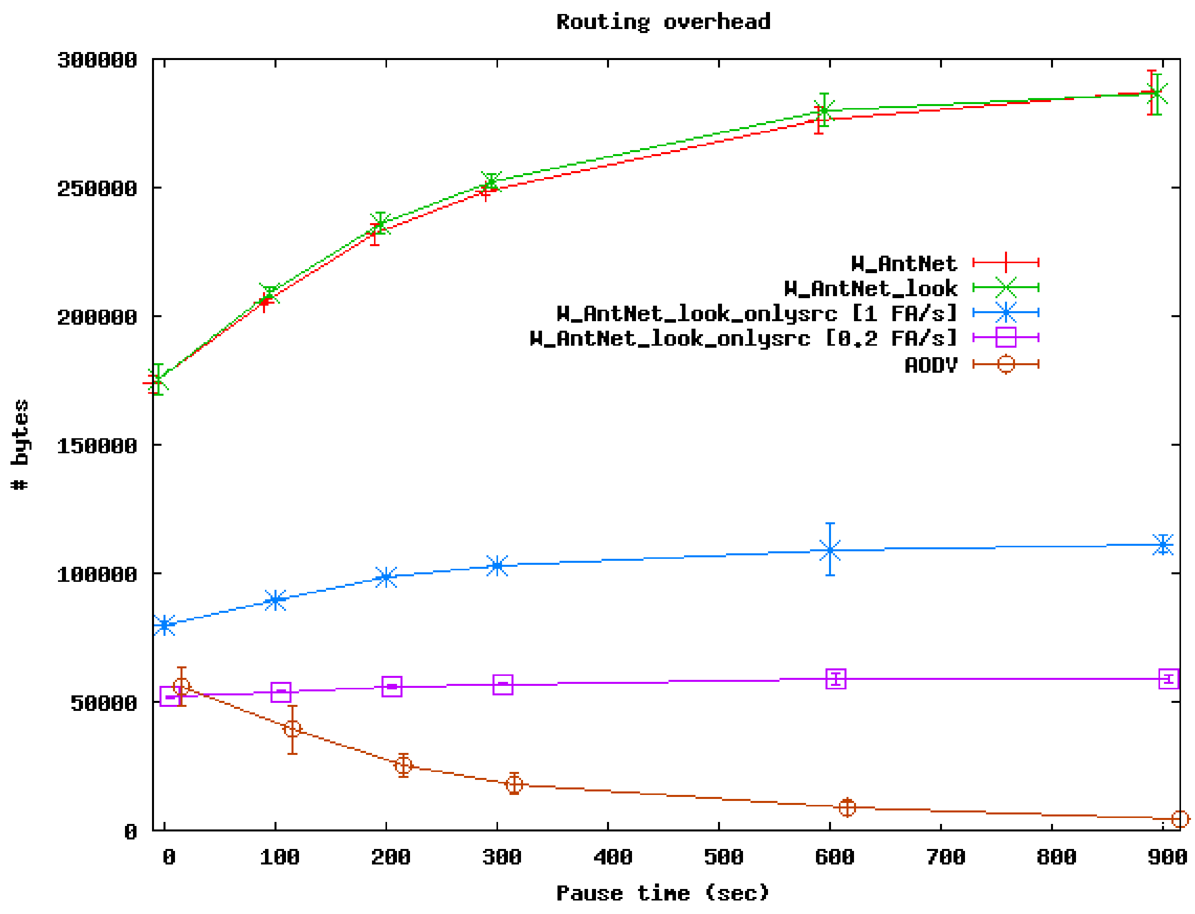
In a large scenario we can see that the protocol W_AntNet still be more optimal tan AODV. If you only forward ants sent to neighboring routers known we achieve a substantial improvement.If more than at first the ants do not send in the form of broadcast, we obtained better results by AODV.



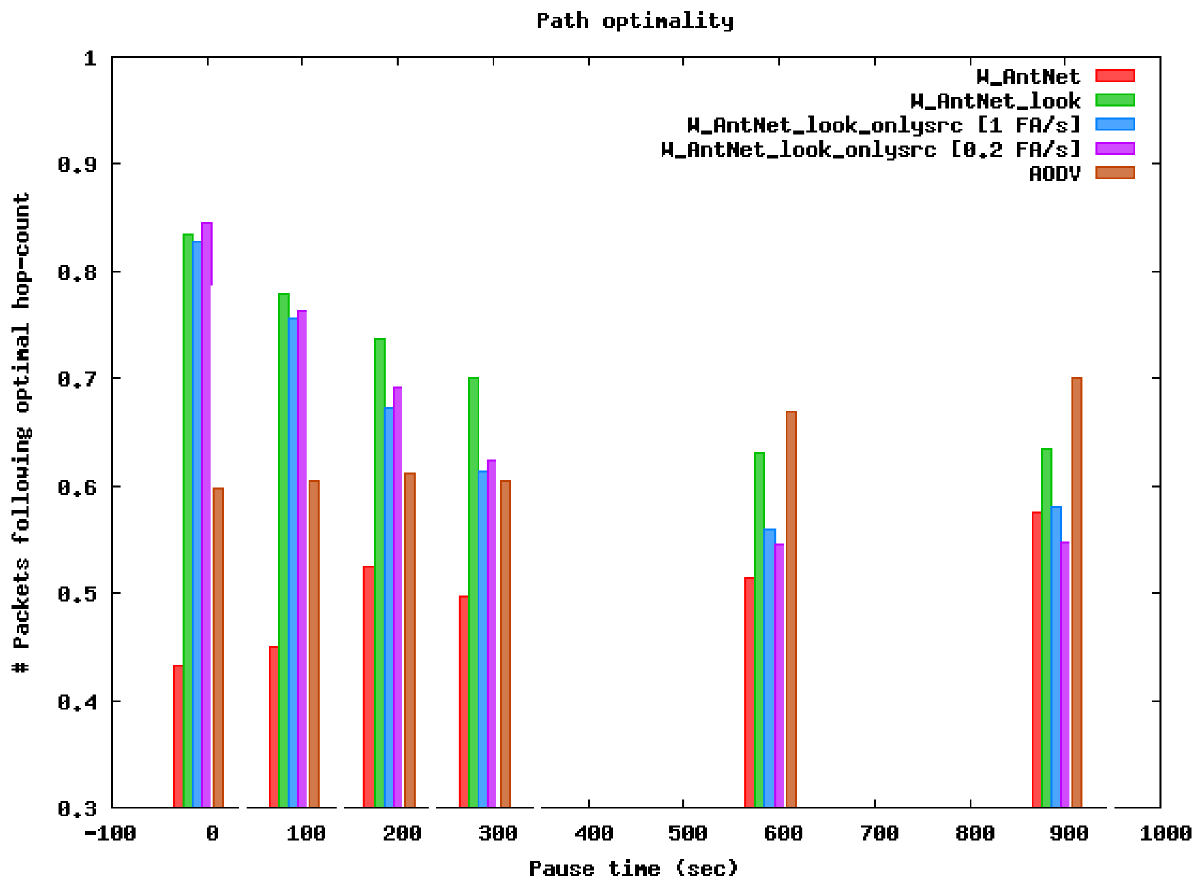
Looking at the ratio between packets sent packets i received in a large scenario we can see that the differences with a small stage are minimal in the case of AODV, but the protocol variants W_AntNet prove more effective in large than in small scenarios.



Looking at the graph of the delay between extremes we can see that W_AntNet_lool_onlysrc variant [0,2 FA / s] where only forward ants are sent to the nodes known every second, it is very competitive. After 900 seconds, this delay is comparable to AODV protocol. Therefore in larger scenarios, the delay generated by the protocols in W_AntNet vasados, is lower than in smaller stages.



This graph routing moderation overa .. We can see that the versions W_AntNet_look W_AntNet and in the early stages of implementing the network saturated with packets broadcasting. In deployments W_AntNet_look_onlysrc [1 FA / s] and W_AntNet_look_onlysrc [0,2 FA / s] routers do not flood the network with improved broadcast packets sent bit rate.



The charts show that if only sent to and from destinations known ForwardAnts performance increases significantly WAntNet

6-Conclusion:

After examining carefully the nature-based protocols, I have come to the conclusion that the only protocol that could be implemented in Ad-Hoc networks is the protocol that is based in Ant Colony Optimization. Other protocols inspired by nature can be applied to our scenario. The bees based protocol could be applied to three-dimensional scenarios, but that would be the matter of another project.

After the simulations made with NS-2 between WAntNet and AODV, WAntNet turns to be not as good as expected.

The main drawback is that when he discovered WAntNet route proactively sends a large number of control packets (BackwardAnts), which often saturates the buffers of the nodes. This leads to lost packets of all types if the buffer is not large.

In a highly dynamic environment, it can produce loops. Performance is decreased with WAntNet compared with AODV, which does not have this problem.

Another flaw we could find is that another host needed to forward its traffic. If this host declined this possibility, we could not use this protocol.

To conclude, we can state that WAntNet can be used as protocol in Ad-Hoc networks but is not as effective as AODV. The approach of Ant Colony Optimization is good, but maybe this is not its environment or maybe I have not managed to implement it correctly.