AntNet: Routing Algorithm for Data Networks based on Mobile Agents

Benjamín Barán[†]

National Computer Center
National University of Asuncion
bbaran@sce.cnc.una.py
P.O.Box 1439
San Lorenzo-Paraguay

Rubén Sosa

Science and Technology School
Catholic University of Asuncion
rsosa@personal.com.py
Tuyutí 1343, Tel. 595-21-390378
Asunción-Paraguay

Abstract. AntNet is an innovative algorithm for packet routing in communication networks, originally proposed by M. Dorigo and G. Di Caro (1997). In AntNet, a group of mobile agents (or artificial ants) build paths between pair of nodes; exploring the network concurrently and exchanging obtained information to update the routing tables. This work analyzes AntNet and proposes improvements that were implemented, comparing their performance with respect to the original AntNet and other commercial routing algorithms like RIP and OSPF. The simulation results indicate a better throughput (amount of packages successfully routed per unit time) of the improved proposals. As for packet delay, the improved proposals equally overcame the original AntNet, although RIP and OSPF were unbeatable in this measure of performance. Due to the great increase in the number of users in networks like Internet, the network service administrators will prioritize the throughput (amount of service that could be offered in a given moment), in order to offer services to the growing number of users. So, AntNet and its variants are promising alternatives for routing of data in big networks.

Keywords: mobile agents, routing, performance, communication network.

1 Introduction

Routing in a data network is the action of addressing data traffic between pair of nodes source-destination, being this, fundamental in a communication network control. In conjunction with a flow control, congestion and admission, routing determines the total network performance, in terms of quality and amount of offered services (Dorigo 1998). The routing task is performed by routers, which update their routing tables by means of an algorithm specially designed for this purpose. The first routing algorithms addressed data in a network minimizing any costs function, like physical distance, link delay, etc. However, throughput optimization remained in a

[†] This work was partially supported by a DIPRI Research Grant of the National University of Asuncion.

second plane, possibly due to a relatively small amount of users. This is the case of the RIP algorithm (Routing Information Protocol), based on the distance-vector method and the OSPF (Open Shortest Path First), algorithm thoroughly used in Internet, based on the link-state method. Both methods choose the path with minimum cost (generally the shortest path) between pair of nodes (Dorigo 1997). This could produce "bottlenecks", because this path could congest, in spite of other paths, possibly expensive, but less congested (Shankar 1992).

Unfortunately, traditional routing methods, due to the limitations explained above, don't have enough flexibility to satisfy the new routing demands, like new network services, and mainly the impressive increase in the amount of users, that forces the network administrators to improve throughput in order to satisfy the immense amount of users that simultaneously request services. This situation has impelled the study and development of other routing methods, to satisfy these new demands. Such is the case, for example, of a routing method known as LBR (Load Balancing Routing) (Back 1999), based on a load-balancing scheme. This method addresses routing by equally distributing load over all possible paths. This diminishes the congestion probability in the minor cost links, improving the network performance.

Actually, other very studied routing alternatives are based on mobile agents (Dorigo 1997, 1998; Schoonderwoerd 1997). In fact, the present work analyzes an algorithm based in this method, known as AntNet, which was first proposed by M. Dorigo and G. Di Caro, of the Free University of Brussels-Belgium (Dorigo 1997, 1998). AntNet was inspired in previous successful works, based on ant colonies (ACS: Ant Colony Systems) (Dorigo 1996, 1997; Schoonderwoerd 1997; Barán 1999). ACS is an optimization method where a group of artificial ants moves around a graph, which represents the instances of the problem; so, they move building solutions and modifying the problem using the obtained information, until they find good solutions to the problem.

The ACS concept is used in AntNet. Here, each artificial ant builds a path from its source node until its destination. While an ant builds a path, it gets quantitative information about the path cost and qualitative information about the amount of traffic in the network. Then, this information will be carried by another ant travelling the same path but in the opposing direction modifying the visited nodes routing tables. The first simulations with AntNet (Dorigo 1997, 1998) showed a promising performance, overcoming traditional algorithms like RIP and OSPF, demonstrating that it is a valid alternative for data routing.

Present work analyzed Dorigo and Di Caro versions of AntNet (Dorigo 1997, 1998). After that, it proposes two improved versions, which were implemented in C language together with the two original AntNet versions, besides versions of RIP, OSPF and three versions of LBR, adding a dozen of algorithms to be compared. Finally, the simulations results demonstrate the improvements in throughput and packet delay obtained with the modified versions, here proposed.

2 AntNet Algorithms

The AntNet first version, presented in 1997 (Dorigo 1997), will be denominated AntNet1.0, and the second version published in 1998 (Dorigo 1998) will be called AntNet2.0. Following, both versions are briefly discussed.

2.1 Algorithm AntNet1.0

Suppose a data network, with **N** nodes, where *s* denotes a generic source node, when it generates an agent or ant toward a destination *d*. Two types of ants are defined:

1. Forward Ant, denoted $F_{s\to d}$, which will travel from the source node *s* to a destination *d*.
2. Backward Ant, denoted $B_{s\to d}$, that will be generated by a forward ant $F_{s\to d}$ in the destination *d*, and it will come back to *s* following the same path traversed by $F_{s\to d}$, with the purpose of using the information already picked up by $F_{s\to d}$ in order to update routing tables of the visited nodes.

Every ant transports a stack $S_{s\to d}(k)$ of data, where the *k* index refers to the *k*-est visited node, in a journey, where $S_{s\to d}(0)= s$ and $S_{s\to d}(m)= d$, being *m* the amount of jumps performed by $F_{s\to d}$ for arriving to *d*.

Let *k* be any network node; its routing table will have N entries, one for each possible destination.
Let *j* be one entry of *k* routing table (a possible destination).
Let N_k be set of neighboring nodes of node *k*.
Let P_{ji} be the probability with which an ant or data packet in *k*, jumps to a node *i*, $i \in N_k$, when the destination is *j* ($j \neq k$). Then, for each of the N entries in the node *k* routing table, it will be n_k values of P_{ji} subject to the condition:

$$\sum_{i \in N_k} P_{ji} = 1 \; ; \; j=1,...,\mathbf{N} \; . \qquad (\mathbf{1})$$

The following lines show AntNet1.0 pseudocode, using the symbols and nomenclature already presented:

BEGIN

{ Routing Tables Set-Up: For each node k the routing tables are initialized with a uniform distribution:

$$P_{ji} = \frac{1}{n_k} \quad , \quad \forall i \in N_k \, . \tag{2}$$

DO always (in parallel)

{ STEP 1: In regular time intervals, each node s launches an $F_{s\to d}$ ant to a randomly chosen destination d.

/*During its trip to d, when $F_{s\to d}$ reach a node k, ($k \neq d$), it does step 2*/

DO (in parallel, for each $F_{s\to d}$

{ STEP 2: $F_{s\to d}$ pushes in its *stack* $S_{s\to d}(k)$ the node k identifier and the time elapsed between its launching from s to its arriving to k.

$F_{s\to d}$ selects the next node to visit in two possible ways:

(a) It draws between i nodes, $i \in N_k$, where each node i has a P_{di} probability (in the k routing table) to be selected.

IF the node selected in (a) was already visited

(b) It draws again, but with the same probability for all neighbor nodes i, $i \in N_k$. $F_{s\to d}$ jumps to chosen node.

IF the selected node was already visited

STEP 3: A cycle is detected and $F_{s\to d}$ pops from its *stack* all data related to the cycle nodes, since the optimal path must not have any cycle. $F_{s\to d}$ comes back to step 2 (a).

END IF

END IF

}WHILE jumping node$\neq d$

STEP 4: $F_{s\to d}$ generates another ant, called backward ant $B_{s\to d}$. $F_{s\to d}$ transfers to $B_{s\to d}$ its *stack* $S_{s\to d}$ and then dies.

/*$B_{s\to d}$, will follow the same path used by $F_{s\to d}$, but in the opposing direction, that is, from d to s*/

DO (in parallel, for each $B_{s\to d}$ ant)

{ /*When $B_{s\to d}$ arrives from a node f, $f \in N_k$ to a k, it does step 5*/

STEP 5: $B_{s\to d}$ updates the k routing table and list of trips, for the entries regarding to nodes k' between k and d inclusive, according to the data carried in $S_{s\to d}$ (k').

IF $k \neq s$

$B_{s\to d}$ will jump from k to a node with identifier given by $S_{s\to d}$ (k-1)

END IF

}WHILE ($k \neq s$)

}

}

END

The routing table and list of trips updating methods for k are described as follows:

1. The *k* routing table is updated for the entries corresponding to the nodes *k*' between *k* and *d* inclusive. For example, the updating approach for the *d* node, when $B_{s\to d}$ arrives to *k*, coming from *f*, $f \in N_k$ is explained, next:
   - A P_{df} probability associated with the node *f* when it wants to update the data corresponding to the *d* node is increased, according to:

$$P_{df} \leftarrow P_{df} + (1 - r').(1 - P_{df}). \quad (3)$$

   where *r´* is an adimensional measure, indicating how good (small) is the elapsed trip time *T* with regard to what has been observed on average until that instant. Experimentally, *r*' is expressed as:

$$r' = \begin{cases} \dfrac{T}{c\mu} & c \geq 1 \text{ if } \dfrac{T}{c\mu} < 1 \\ 1 & \text{otherwise} \end{cases} \quad (4)$$

   where: μ is the arithmetic observed trip time *T* average.
   c is a scale factor experimentally chosen like 2 (Dorigo 1997).
   More details about r' and its significance can be found in (Dorigo 1997).
   - The other neighboring nodes ($j \neq f$) P_{dj} probabilities associated with node *k* are diminished, in order to satisfy equation (1), through the expression:

$$P_{dj} \leftarrow P_{dj} - (1 - r')P_{dj} . \quad \forall j \in N_k , \; j \neq f \quad (5)$$

2. A list $trip_k(\mu_i,\sigma_i^2)$ of estimate arithmetic mean values μ_i and associated variances σ_i^2 for trip times from node *k* to all nodes *i* ($i \neq k$) is also updated. This data structure represents a *memory* of the network state as seen by node *k*. The list trip is updated with information carried by $B_{s\to d}$ ants in their stack $S_{s\to d}$. For any node pair source-destination, μ after (n+1) samples (n>0) is calculated as follows:

$$\mu_{n+1} = \frac{n\mu_n + x_{n+1}}{n+1} \quad (6)$$

   where: x_{n+1} trip time *T* sample n+1,
   μ_n arithmetic mean after *n* trip time samples.

2.2. AntNet2.0 Algorithm

AntNet2.0 is an AntNet1.0 modified version (Dorigo 1998) with five main steps, which perform basically the same actions as AntNet1.0. The differences are how the routing tables and the lists trips (now known in as traffic local model M_k) are updated. Consequently, only these two differences will be explained.

Suppose that a $B_{s\to d}$ arrives to a node *k*, in its return trip to node *s*. The $B_{s\to d}$ ant will update the traffic local model M_k (list trip in AntNet1.0) and the neighbor nodes probabilities of *k* associated to node *d* in the routing table Γ_k . Also, as in AntNet1.0 step 5, the update is performed in the entries corresponding to every node $k' \in S_{s\to d}$, $k' \neq d$ in the subpaths followed by $F_{s\to d}$ after visiting *k*. If a subpath trip time *T* is statistically good (i.e: *T* is smaller than μ + I(μ,σ), where I is a μ interval confidence

estimator), then T is used to update the statistics related and the routing table. However, if T is bad, it is not used, because it doesn't give a true idea about the time required to arrive to the subpath nodes. The traffic local model M_k and the routing table Γ_k are updated for a generic destination $d' \in S_{s \to d}$ in the following way:

1. M_k is updated with the values carried in $S_{s \to d}$. The trip time $T_{k \to d'}$ employed by $F_{s \to d}$ to travel from k to d' is used to update $\mu_{d'}$, $\sigma_{d'}^2$ and the best observed value inside window $W_{d'}$ according to the expressions:

$$\mu_{d'} \leftarrow \mu_{d'} + \eta \left(T_{k \to d'} - \mu_{d'}\right). \quad (7)$$

$$\sigma_{d'}^2 \leftarrow \sigma_{d'}^2 + \eta \left(\left(T_{k \to d'} - \mu_{d'}\right)^2 - \sigma_{d'}^2\right) \quad (8)$$

where η is the weight of each trip time observed. The effective number of samples for will be approximately $5(1/\eta)$. Therefore, for 50 samples, η=0.1, and for 100 samples η=0.05. The role of $W_{d'}$ will be explained later.
The $T_{k \to d'}$ mean value and its dispersion could vary strongly, depending on traffic conditions: a poor (large) time with low data traffic could be very good with relation to another measure with more traffic. The statistical model should reflect this variability and continue the traffic fluctuations in a robust way. This model plays a critical role in routing table updating.

2. The routing table for k is updated in the following way:
   - The value $P_{fd'}$ (the probability for selecting the neighbor node f, when the node destination is d') is incremented by means of the expression:

$$P_{fd'} \leftarrow P_{fd'} + r(1-P_{fd'}). \quad (9)$$

   where r is a reinforcement factor indicating the goodness of the followed path.
   - The $P_{nd'}$ probabilities associated to the other nodes decreases respectively:

$$P_{nd'} \leftarrow P_{nd'} - r\,P_{nd'}, \quad n \in N_k,\ n \neq f \quad (10)$$

The factor of reinforcement r is calculated considering three fundamental aspects: (i) the paths should receive an increment in their probability of selection, proportional to their goodness, (ii) the goodness is a traffic condition dependent measure, that can be estimated by M_k, and (iii) they should not continue all the traffic fluctuations in order to avoid uncontrolled oscillations. It is very important to establish a commitment between stability and adaptability. Between several tested alternatives (Dorigo 1998), expression (11) was chosen to calculate r:

$$r = c_1 \left(\frac{W_{best}}{T}\right) + c_2 \left(\frac{I_{\sup} - I_{\inf}}{\left(I_{\sup} - I_{\inf}\right) + \left(T - I_{\inf}\right)}\right) \quad (11)$$

where: W_{best}. best trip of an ant to node d', in the last observation window $W_{d'}$,
$I_{\inf} = W_{best}$. lower limit of the confidence interval for μ,
$I_{\sup} = \mu + z^*\left(\sigma / \sqrt{|w|}\right)$. upper limit of the confidence interval for μ, with:
$z = 1/\sqrt{(1-\gamma)}$, γ = confidence level, $\gamma \in [0.75, 0.8]$.
c_1 and c_2 are weight constants, chosen experimentally as c_1=0.7 y c_2=0.3.
More details about r and its significance can be found in (Dorigo 1998).

3 Modifications Proposed to Improve Antnet

This section describes several modifications tested for AntNet1.0 and AntNet2.0, in order to improve their performances.

3.1. Intelligent Initialization of Routing Tables

AntNet1.0 and AntNet2.0 don't specify an initialization method for the routing tables (Dorigo 1997, 1998). For this reason, a uniform distribution of probabilities is assumed, according to the initialization given in the pseudocode. Due to this situation of null a-priori knowledge, it is proposed an initialization of each node routing table that reflects a previous knowledge about network topology. Furthermore, an initial greater probability value is assigned to the neighboring nodes that simultaneously could be destinations. For a node k this could be described as follows:

1. If a destination node d for a table entry is at the same time a neighbor node, that is $d \in N_k$, then the initial probability in the routing table of k is given by:

$$P_{dd} = \frac{1}{n_k} + \frac{3}{2} * \frac{(n_k - 1)}{{n_k}^2} \qquad \textbf{(12)}$$

For the rest of the neighboring nodes $i \in N_k$, it will be:

$$P_{di} = \begin{cases} \frac{1}{n_k} - \frac{3}{2} * \frac{1}{{n_k}^2} & \text{if } n_k > 1 \\ 0 & \text{if } n_k = 1 \end{cases} \qquad \textbf{(13)}$$

Of course, (12) and (13) satisfy (1).

2. If the destination d is not a neighbor node, that is $d \notin N_k$, then a uniform distribution is initially assumed:

$$P_{di} = \frac{1}{n_k}. \qquad \textbf{(14)}$$

Due to the advantage of network topology knowledge reflected by the initial probability values in the routing tables, this method showed a transient regime shorter than the observed during simulations in AntNet1.0 and AntNet2.0.

3.2 Intelligent Updating of Routing Tables after Network Resources Failures

Original AntNet algorithms (Dorigo 1997,1998) do not mention the following cases:

1. Routing tables updating in case of links or node failure, that is, immediately after a node k loses its link l_{kj} with its neighbor node j. In principle, it is supposed that if an ant is in k, the probability P_{dj}, for arrive to a destination d across a jumping node

j, that is, to use the link l_{kj}, is distributed uniformly between the remaining n_k -1 neighbor nodes for the entry *d* in the routing table of *k*. Mathematically, P_{dj} =0 during a link l_{kj} failure (it isn't possible to jump from *k* to *j* for arriving to *d*).

$$P_{di} = P_{di} + \frac{P_{dj}}{n_k - 1}. \quad \forall i \neq j \;\; i, j \in N_k \qquad (\mathbf{15})$$

Alternatively, the present work proposes the idea of new P_{di} values immediately after the l_{kj} link failure. These probabilities will be proportional to their relative values, before the failure, according to the acquaintance until that instant, instead of "forgetting" everything he learned until the moment of the failure, according to (15). So, in a node *k*, after the l_{kj} link failure, a factor Q is calculated like:

$$Q = \frac{P_{dj}}{1 - P_{dj}}. \qquad (\mathbf{16})$$

then, P_{di} is updated according to:

$$P_{di} = (1 + Q) * P_{di}. \quad \forall i \neq j; \quad i \in N_k, P_{dj}=0. \qquad (\mathbf{17})$$

This method reflects the node knowledge about the network traffic and topology before the failure, so during the event the algorithm should show a better performance according to the original algorithms.

2. Routing table updating for the *k*-*j* node pair when the link l_{kj} is up at time t_2 (t_2>0), since this link was down at time t_1, 0<t_1<t_2. AntNet1.0 and AntNet2.0 use a routing table reinitialization for *k* and *j* according to (2), losing all information learned right before the link failure. As an alternative, this work proposes a reinitialization subject to a commitment between learned information until instant t_1, before the link failure, and total ignorance of the node as in *t*= 0. So, the probabilities in the routing table for a node *k*, whose link failed in t_1, but recovered in t_2 will be:

$$P_{di}(t_2) = (1-\lambda)P_{di}(0) + \lambda P_{di}(t_1). \qquad 0 \leq \lambda < 1 \qquad (\mathbf{18})$$

The factor λ is a constant, known as *coefficient of memory*, since its value indicates how much it remembers what it has learned until time t_1. An empirical value of 0,6 was adopted. This criterion makes more robust the algorithm allowing a faster recovery time after link or node failure.

3.3 Introduction of a noise factor and limitation of probability values

With the routing tables updating methods in AntNet1.0 and AntNet2.0, the distribution of probabilities eventually "would freeze" with any probability value, near to one, with the rest of them remaining with insignificant values. Thus, in any node, ants and data packets would mostly choose the output line with the highest probability. In order to prevent this, it is defined a noise factor *f*, so, every time an ant should jump to a following node, it chooses that node with a probability *f*, according to an uniform distribution probability, and with a probability (1-*f*), according to the

values stored in the routing tables (Schoonderwoerd 1997). With this, the ants by "accident" could discover new and better paths. So, potentially both the delay and throughput could improve.

Also, any probability in the routing tables was limited to a maximum value of: P= 1-(n_k-1)*ε, (experimentally, ε=0.05), being ε, the minimum probability admitted.

3.4 Dual Method for selection of jumping node

In the AntNet1.0 and AntNet2.0 algorithms, being in a node k, a data packet, whose destination is a node $d \neq k$, will select a jumping node j randomly, according to P_{dj}, $\forall j \in N_k$. Also, this work considers the possibility of selecting directly the output line corresponding to node j with the highest probability value in the routing table of k, between the n_k probabilities associated to node d. This last method is called hierarchical method (Schoonderwoerd 1997).

As mentioned before, in each node, packets will decide randomly whether to use the usual method (random) or the hierarchical method, in order to choose the jumping node. Particularly, a better behavior for the case of $P = 0.5$ was observed, where P is the probability for the use of the random method, normally used in AntNet. So, for a data packet, there will exist a probability $P = 0.5$ of using the random approach, and a probability $\overline{P} = 0.5$ of directly using a node, whose probability is the highest in the k routing table, for the destination d. For AntNet1.0 and AntNet2.0 P=1 is considered.

3.5 Control of the number of ants inside the network

AntNet1.0 and AntNet2.0 don't mention any method to maintain control of the total numbers of ants moving inside the network, which, under certain circumstances, could contribute to congestion. In order to control the number of ants, it was first attempted to limit the maximum number of ants in an amount equal to the square of the number network nodes. This approach was computationally very heavy, in addition to requiring very large data structures. For this reason, the number of ants was limited to an amount four times the number of network nodes. With this alternative, the simulation results were improved and the computing load diminished. So, this approach was adopted for the implemented algorithms.

3.6 Self-destruction of a Backward Ant

Self-destruction of a backward ant $B_{s \to d}$ refers to a $B_{s \to d}$ ant that can't return to its source node, because its return trip was cut, due to a link or node failure. Under this situation the ant is self-destroyed, because the information stored in its stack already does not reflect the real state of the network. This point was very important, so it was added to all the AntNet algorithms, including AntNet1.0 and AntNet2.0.

4 Experimental Results

With the proposed modifications 2 alternative algorithms were implemented:

1. AntNet1.1: It is a modified version of AntNet1.0.
2. AntNet2.1: It is a modified version of AntNet2.0.

The main characteristics for the implemented algorithms are:

- All these algorithms have been implemented in C language.
- Parallel behavior simulated with serial code.
- A data traffic simulation analysis is performed after each time slot.

For the simulations, three networks have been used as models:

1. A simple network, with 8 nodes, links with unitary cost, Fig. 1 (Dorigo 1998).
2. The NSFNET (National Science Foundation network) from the United States, with 14 nodes and 1.5 Mbps links. It is observed the links delay in [ms], Fig. 2.
3. The NTT network (Nippon Telephone Telegraph) of Japan, composed by 55 nodes and links of 6 Mbps, Fig. 3.

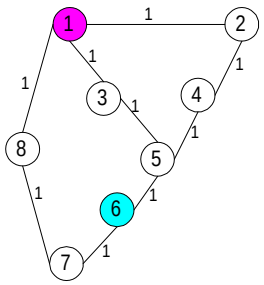

Fig. 1. Simple Network

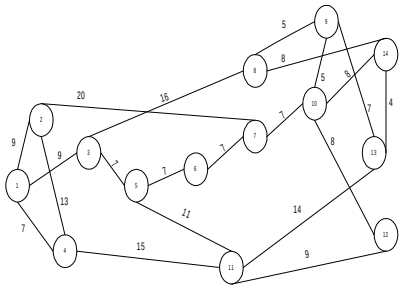

Fig. 2. NSFNET

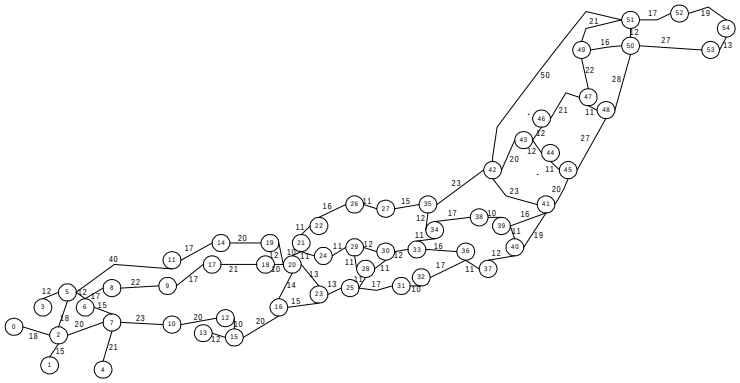

Fig. 3. NTTnet

A benchmark was established for the simulations. 12 simulation scenarios compose our benchmark, as shown in Table 1. Each scenario was tested for each one of the three model networks presented above.

	Lost Packet threshold	Transient Regime	Link Failure	Node Failure	Hot Spot
Low Traffic	5%	✓	✓	✓	✓
Medium Traffic	10%	✓	✓	✓	✓
High Traffic	20%	✓	✓	✓	✓

Table 1. Benchmark used for each model network

For each simulation cycle, the traffic simulator will stop generating packets when a certain fraction (expressed in %) of the generated packet has not arrived to destination (Lost Packet threshold). The link transmission delay is used as metric for link costs, expressed in milliseconds. The performance parameters for the algorithms are:

- *Instantaneous Packet Delay*. It is the average delay of all data packets routed successfully for a given instant *t* in the algorithm simulation.
- *Average Packet Delay*. It is the average delay of all data packets routed successfully during the whole simulation period.
- *Instantaneous Throughput*. It is the amount of packets routed successfully for a given instant *t* in the algorithm simulation.
- Average Throughput: It is the amount of packets routed successfully during the whole simulation period.

Figures 4 to 7 and Tables 2 to 4 show the simulation results for some of the experiments performed for the three mentioned networks and only for medium traffic. It is important to mention that the results obtained for the LBR algorithms were not as good as first expected, so they will not be presented nor discussed for any of the experiments to be considered. In the tables, THR stands for average throughput and for AVP for average packet delay. The following abbreviations will be used for AntNet algorithms: AntNet1.0=A1.0, AntNet1.1=A1.1, and so on.

In what follows, it will be presented the simulations results for the networks mentioned:

1. Simple Network (Fig. 1): It is wanted to compare the routing methods used by the traditional algorithms RIP and OSPF with A1.0. For this, it was simulated data traffic between nodes 1 and 6, with the following results.
- Throughput: A1.0 have a performance about three times better than RIP (Table 2). This is so because there are three possible paths between nodes 1 and 6; two of which have the same cost. In spite of this, RIP and OSPF send the packets only through one path (the chosen best path). On the contrary, A1.0 distributes the load between the three paths, proportionally to their goodness (cost).
- Packet Delay: In table 2 it is observed a packet delay greater in A1.0, because approximately the third part of the packets is sent through the longest path.

Algorithms	THR [packets]	AVP [ms]
RIP	2367	3.01
OSPF	2589	3
A1.0	7245	3.03

Table 2: Average parameters results

Clearly, as expected, AntNet has a better throughput at a price of a larger packet delay, a situation that is founded in almost every experiment.

2. NSFNET (Fig. 2): Here are showed results for transient regime experiment for AntNet algorithms (Table 3), and for a transitory link failure experiment (link 5-6, see Fig. 3) (Fig 4-5, Table 3), concluding the following:

- Transient Regime: Table 3 indicates how the modified algorithms "learn" quicker (better throughput and packet delay) than A1.0 and A2.0, due to using of the routing tables intelligent initialization (section 3.1) and the dual method for jumping node selection by data packets (section 3.4). Also, the superiority of A2.1 is observed between all the algorithms.
- Link 5-6 Failure: Throughput. RIP and OSPF decay completely at the instant of the failure (Fig. 4); however, the AntNet algorithms are not severely affected, demonstrating their robustness for this type of failures. In particular, it is observed in Table 3 that A2.1 has the best average throughput. Also, A1.1 overcome to A1.0 (Table 3), due to routing tables intelligent reinitialization method (section 3.2).
- Link 5-6 Failure: Packet Delay. All algorithms are proportionally affected (Fig. 5), while the inherent advantage of RIP and OSPF in this figure of merit remains. Here, the modified AntNet algorithms also overcome A1.0 and A2.0.

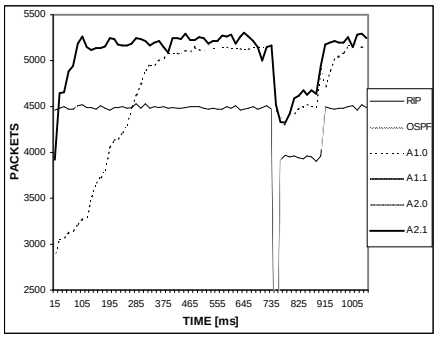

Fig. 4. Link 5-6 failure. Instant. throughput

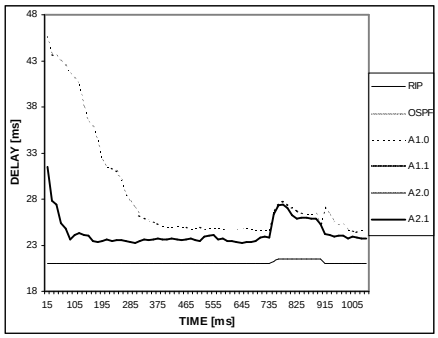

Fig. 5. Link 5-6 failure: Instant. Packet Delay

		RIP	OSPF	A1.0	A1.1	A2.0	A2.1
Transient	THR [packets]			3572.75	4509.77	4716.79	5008.4
Regime	AVP [ms]			37.79	28.72	27.17	24.27
Link 5-6	THR [packets]	4347.61	4450.33	4605.02	4869.94	4844.56	5081.3
Failure	AVP [ms]	21.06	20.1	28.53	26.81	25.58	24.25

Table 3. Experimental Results for Average throughput and Average packet delay

3. NTTnet (Fig. 3): Experimental results are showed in for: link failure and hotspot.

- Link 31-32 failure Throughput. AntNet algorithms are more robust than RIP and OSPF. A2.1 has the best average THR (Table 4) and A1.1 is better than A2.0.
- Link 31-32 Failure: Packet Delay. Again, our AntNet are the best (Table 4).
- Transient Hotspot: Throughput. Node 41 was chosen as a hotspot (Fig. 3). Here, A2.1 is the best (Table 4), although finally, all AntNet converge to a similar behavior (Fig. 6). AntNet algorithms show small oscillations during the hotspot.
- Transient Hotspot: Packet Delay. A2.1 has the best behavior in presence of a hotspot (Fig. 7), possibly due to change in the data traffic spatial distribution, caused by the hotspot. Table 4 shows the best average values for our AntNet.

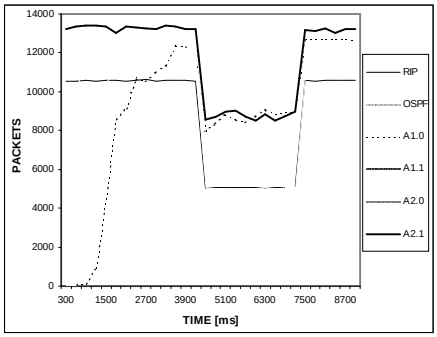

Fig. 13. Transient Hotspot. Instant. throughput

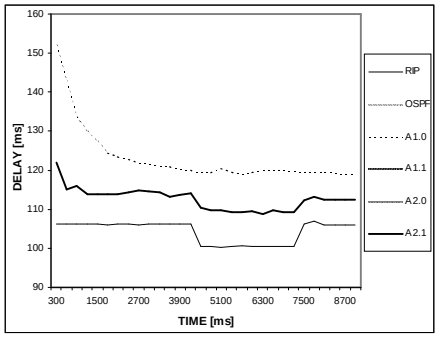

Fig. 14. Transient Hotspot. Inst. packet delay

		RIP	OSPF	A1.0	A1.1	A2.0	A2.1
Link 31-32	THR [packets]	10201.77	10803.32	9717.06	11061.8	10774.9	12879
Failure	AVP [ms]	107.49	104.61	124.16	120.72	118.02	114.53
Transient	THR [packets]	8736.23	8848.26	8891.3	11065.6	9423.13	11759
Hotspot	AVP [ms]	104.26	102.63	123.19	119.41	116.25	112.58

Table 4. Experimental Results for Average throughput and Average packet delay

After the analysis of simulation results, these general conclusions can be done:

- Our AntNet have shorter transient regime than A1.0 and A2.0 (Table 3).
- AntNet algorithms are more robust than RIP and OSPF algorithms for link failure, because their instantaneous throughput does not decay completely at instant of the failure (Tables 3-4).
- For hotspots, the results suggest that AntNet algorithms are sensitive to changes in the data traffic geographical distribution, because oscillations in the instantaneous packet delay were observed, during the presence of the hotspot (Figs. 6, 7).
- In most of the experiments A2.1 showed the best performance.
- Among all the AntNet algorithms, A1.0 showed the worst performance, and it never performed better than A2.0, due to the superior method used for the routing tables and trip lists updating (Dorigo 1998). However, A2.0 proved worse than A1.1 in some circumstances (Figs. 11,13), in spite of the fact this last algorithm is based on A1.0, demonstrating the effectiveness of the modifications proposed.

5 Conclusions

In this work two versions of AntNet algorithms were studied, a novel adaptive routing technique for data networks, based on mobile agents, oriented towards packet switching, such as Internet. After their study, two modified versions were presented.

AntNet algorithms, in addition to RIP, OSPF and LBR (Back 1999) were implemented and simulated. A better performance of our versions of AntNet was observed in most of the experiments. The modifications implemented in our versions that contributed more for a better behavior of them were: a) routing tables intelligent initialization and, b) dual method for selecting jumping node for data packets.

In general, the results of the experiments remained proportional (Sosa 2000). Results obtained in a different simulation scope suggest that AntNet algorithms could have better throughput as well as packet delay than RIP and OSPF (Dorigo 1997, 1998). If this is the case, it is expected that the modified algorithms proposed here will have better performance than the original AntNet versions.

It is also expected an efficient AntNet behavior with: flow control, congestion and admission schemes. Therefore, it can be inferred that the commercial implementation of this algorithm may be feasible and it can even be considered its use in large networks, such as Internet, as a future option.

References

Almirón, M., Barán, B., Chaparro, E.: Ant Distributed System for Solving the Traveling Salesman Problem. 15th Informatic Latinoamerican Conference-CLEI, Vol. 2. Paraguay (1999) 779-789

Bak, S., Cobb, J., Leiss, E.: Load Balancing Routing via Randomization. 15th Informatic Latinoamerican Conference-CLEI, Vol. 2. Asuncion-Paraguay (1999) 999-1010

Dorigo, M., Di Caro, G.: AntNet. A Mobile Agents Approach to Adaptive Routing. Technical Report, IRIDIA- Free Brussels University, Belgium (1997)

Dorigo, M., Di Caro, G.: AntNet. Distributed Stigmergetic Control for Communications Networks. Journal of Artificial Intelligence Research, Number 9 (1998) 317-365

Dorigo, M., Maniezzo, V., Colorni, A.: The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B, Vol. 26. Number 1 (1996) 1-13

Feit, S.: TCP/IP. Architecture, Protocols and Implementation. McGraw & Hill (1996)

Rutkowski, T.: Dimensioning the INTERNET. IEEE Internet Computing, Vol. 2. Number 2 (1998) 8-10

Schoonderwoerd, R., Holland, O., Bruten, J.: Ant-like agents for load balancing in telecommunications networks. Hewlett-Packard Laboratories, Bristol-England (1997)

Shankar, A., Alaettinoglu, C., Matta, I.: Performance Comparison of Routing Protocols using MaRS. Distance Vector versus Link-State. Technical Report, Maryland-USA (1992)

Sosa, R.: Improved AntNet. Algorithm for data routing based on mobile agents. Science and Technology Faculty, Asuncion Catholic University, (2000) 47-79

Tanenbaum, A.: Computer Network. Prentice & Hall, Third Edition (1996)