Improved AntNet routing

Benjamín Barán
National Computer Center
National University of Asuncion
P.O.Box 1439, San Lorenzo, Paraguay
Phone: (+595-21) 585550, email: bbaran@cnc.una.py

Abstract-AntNet is a new algorithm for packet routing in communication networks. In AntNet, a group of mobile agents (artificial ants) build paths between pair of nodes, exploring the network concurrently and exchanging data to update routing tables.

This work, based in a previous work of the author [3], analyzes AntNet algorithms and proposes improvements, comparing their performance with respect to the original AntNet and other commercial algorithms. Simulation results indicate a better throughput of the improved proposals. So, AntNet and its variant here proposed are promising options for routing in large public networks such as Internet.

Keywords: Routing, AntNet, Trhoughput, delay.

INTRODUCTION

Routing in a data network is the action of addressing data traffic between a 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 [9]. The routing task is performed by *routers*, which update their routing tables by means of an algorithm specially designed for this. The first routing algorithms addressed data in a network minimizing a cost function, like physical distance, link delay, etc [10,14]. However, throughput optimization remained in a 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 a pair of nodes [7]. This could produce *bottlenecks*, because this path could congest, in spite of other paths, possibly expensive, but not congested [13].

Unfortunately, traditional routing methods, due to the limitations explained above, do not have enough flexibility to satisfy new routing demands, like new network services, and 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 develop other routing methods as LBR (Load Balancing Routing) [2]. This method addresses routing by distributing load over all possible paths improving network throughput, because congestion probability decreases in the shorstest path links.

Nowadays, other very studied routing alternatives are based on mobile agents [7, 9, 12]. Inspired in those algorithms, this work analyzes an algorithm based on mobile agents, known as AntNet, which was first proposed by M. Dorigo and G. Di Caro, of the Free University of Brussels-Belgium [7-9]. AntNet was inspired in previous successful works, based on ant colonies (ACS: Ant Colony Systems) [1, 5, 6, 12]. ACS is an optimization method where artificial ants move 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 (or mobile agent) builds a path from its source node to 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 is carried by another ant travelling the same path but in the opposing direction modifying the visited nodes routing tables. The first simulations with AntNet (1997-98) showed promising results, overcoming classic algorithms like RIP and OSPF [7-9]. So, it seems a valid option for data routing.

The present work is based on two versions of Dorigo and Di Caro AntNet [7-9]. The version published in [9] (here denominated AntNet1.0) had a better performance than the one presented in [7]. Based on AntNet1.0, this paper proposes an improved version: AntNet1.1, which was implemented in C language together with AntNet1.0, besides versions of RIP, OSPF and LBR. Simulations results show a better throughput and packet delay for AntNet1.1 than for other Antnet versions.

ANTNET 1.0 ALGORITHM

Suppose a data network, with **N** nodes, being *s* a generic source node that generates an agent (or *ant*) toward a destination *d*. Two types of ants are defined:

Forward Ant, or $F_{s\rightarrow d}$, which will travel from a source *s* to a destination *d*.

Backward Ant, or $B_{s\rightarrow d}$, that will be generated by a forward ant $F_{s\rightarrow d}$ in the destination *d*. It will return to *s* through the path used by $F_{s\rightarrow d}$.

In its way to *s*, $B_{s\to d}$ updates routing tables of the visited nodes using the information already collected by $F_{s\to d}$.
Every ant carries a stack $S_{s\to d}(k)$ of data, where the index *k* refers to the *k-est* visited node in a journey, where $S_{s\to d}(0)= s$, $S_{s\to d}(m)= d$, being *m* the jumps done by $F_{s\to d}$ to reach *d*.

Let *k* be any network node; its routing table will have N entries, one for each possible destination.

Let *j* be a entry of *k* routing table (possible destination).
Let N_k be the 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 node *k* routing table, it will be n_k values of P_{ji} with:

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

In what follows, AntNet1.0 pseudocode is presented.

```
BEGIN
{
  Routing Tables Set-Up: For each node k the routing tables are
  initialized with a uniform distribution of probability:
  $P_{ji} = \frac{1}{n_k}, \ \forall i \in N_k$
  (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.
    /*when $F_{s\to d}$ reaches a node k, ($k\neq d$), it performs step 2*/
    DO (in parallel, for each $F_{s\to d}$)
    {
      STEP 2: $F_{s\to d}$ pushes in its stack $S_{s\to d}(k)$ node k identifier and the
      time 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 the jumping node, but now with the
                   same probability for all neighbors $i$, $i \in N_k$.
              IF the selected node was already visited
                STEP 3: A cycle is found. $F_{s\to d}$ pops from its stack all
                data of the cycle nodes, because the optimal path must
                not have any cycle. $F_{s\to d}$ returns to 2 (a) if the time
                spent in the cycle is less than a threshold; else it dies in
                order to avoid infinite loops.
              END IF
            END IF
    }WHILE jumping node$\neq$ d
    STEP 4:  $F_{s\to d}$ generates in d another ant, called backward ant $B_{s\to d}$.
    $F_{s\to d}$ transfers to $B_{s\to d}$ its stack $S_{s\to d}$.
    /*$B_{s\to d}$, will return to s,following the same path used by $F_{s\to d}$*/
    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 node k, it performs
      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'), increasing probabilities associated
      to path used and decreases other paths probabilities, by mean of a
      criteria explained in [7].
      IF $k\neq s$
          $B_{s\to d}$ will leave k and jump to a node given by $S_{s\to d}$ (k-1).
      END IF
    }WHILE ($k\neq s$)
  }
}END
```

The main differences between the two already published versions of AntNet algorithms [7-9] are the following:

- In [7], the destination node *d* for a mobile agent is selected randomly. However, in [9], the destination node is selected according to the data traffic patterns generated by the local workload.
- The first version of AntNet given in [7] only considers routing table information when a *Forward Ant* ($F_{s\to d}$) selects a next node during a travel towards destination. However, AntNet 1.0 considers also buffer use to calculate a better estimation of buffer delay.
- Each node *k* has a data structure of size 2**N** known as *List Trip*$_k(\mu_i, \sigma_i)$, where μ_i and σ_i are the mean an variance for trip times $T_{k\to i}$ performed by ants traveling from node *k* to all other nodes *i* in the network. This data structure plays a role of data traffic local estimation. The *List Trip* in [7] is updated using all measured trip times (from the first trip time to the last). In turn, the *List Trip* updating is performed in [9] using windowed strategies. For this, a factor η is defined to indicate how many of the last trip time samples will have a moving window *W* and consequently, how many samples will really influence the calculation of μ and σ.
- For routing tables updating, each version uses a different heuristic calculation method (see formulae in [7-9]). From these two alternatives, a better performance was reported with the method proposed in [9].

ANTNET1.1: AN IMPROVED VERSION OF ANTNET1.0

AntNet1.1 basically uses the same pseudocode as AntNet1.0. However, several modifications were implemmented in order to improve the performance of AntNet1.0. These modifications are briefly explained here.

Intelligent Initialization of Routing Tables

AntNet versions do not specify an initialization method for the routing tables [7-9]. For this reason, a uniform distribution of probabilities is assumed, according to the initialization given in the presented pseudocode. Due to this situation of no a-priori knowledge, here we propose an initialization of each 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. This saves network resources, because it is possible to reach the destination using just a link. For a node *k* this could be as follows:

a) 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:

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

The other neighbors nodes ($i\neq d$), $i \in N_k$, will have:

$$P_{di} = \begin{cases} \frac{1}{n_k} - \frac{3}{2} * \frac{1}{n_k^2} & \text{if } n_k > 1 \\ 0 & \text{if } n_k = 1 \end{cases} \qquad (4)$$

Of course, (3) and (4) satisfy (1).

b) If the destination d is not a neighbor node, then a uniform distribution is initially assumed:

$$P_{di} = \frac{1}{n_k} \qquad (5)$$

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

Intelligent Update after Network Resources Failures

Original AntNet algorithms [7-9] do not mention the following cases:

1. Updating of routing tables in case of links or node failure, that is, immediately after a node k loses its link l_{kj} with its neighbor node j. First, it was supposed that if an ant is in k, the probability P_{dj}, to a destination d through node j, (i.e. to use the link l_{kj}), is distributed uniformly between the remaining n_k -1 neighbors for the entry d in the routing table of k. Mathematically:

   P_{dj} =0, during a link l_{kj} failure (it is not possible to travel from k to j for arriving to d).

   $$P_{di} = P_{di} + \frac{P_{dj}}{n_k - 1} \qquad \forall i \neq j, \quad i, j \in N_k \qquad (6)$$

   Alternatively, this 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, instead of "forgetting" what it has learned until the moment of the failure, according to (6). So, in k, after the failure of l_{kj} link, a factor Q is calculated as:

   $$Q = \frac{P_{dj}}{1 - P_{dj}} \qquad (7)$$

   then, P_{di} is updated according to:

   $$P_{di} = (1 + Q) * P_{di} \qquad \forall i \neq j, \; i \in N_k \qquad (8)$$

   logically, during the l_{kj} link failure P_{dj}=0.

   This method reflects node knowledge about the network traffic and topology before the failure, so a better performance is expected.

2. Updating of routing table for the k-j node pair when the link l_{kj} is up at time t_2, since this link was down at time t_1, $0<t_1<t_2$. AntNet1.0 uses a routing table reinitialization for k and j according to (2), losing the learned information right before the link failure.

   As alternative, this work proposes a reinitialization subject to a commitment between learned information until instant t_1, before the failure, and total ignorance of the node as in t= 0. The probabilities in the routing table of k, whose link failed in t_1, but recovered in t_2 will be:

   $$P_{di}(t_2) = (1 - \lambda) * P_{di}(0) + \lambda * P_{di}(t_1) \qquad 0 \leq \lambda < 1 \qquad (9)$$

   The factor λ is a constant, known as *coefficient of memory*. Its value indicates how much it remembers what it had learned until time t_1. After several tests, an empiric value of 0,6 was adopted. This makes more robust the algorithm allowing a faster recovery time.

Noise factor

With the routing tables updating methods in original versions of AntNet, the distribution of probabilities eventually "would freeze" with a probability value, close to one, and the rest of them could remain with insignificant values. With this, in any node, the ants and data packets would mostly choose the output line with the highest probability (not using other possible paths). To prevent this, we define a f noise factor, so, every time that an ant should jump to a following node, it chooses a node with a probability f, according to an uniform distribution of probabilities, and with a probability (1-f), according to the probabilities stored in the routing tables [12]. With this, the ants by "accident" can discover new and better paths. So, potentially both the delay and throughput could improve.

Dual Method Randomic and Deterministic:

In the original AntNet of Dorigo and Di Caro, being in a node k, a data packet, whose destination is a node d≠k, will select a jumping node j randomly, according to P_{dj}, $\forall j \in N_k$. The present work considers a deterministic method of selecting a jumping node [11]. Whenever a node k have in its queue M packets, it calculates the number of packets to be routed via each of their neighbor nodes according to their probabilities associated for each destination. Therefore, $\forall j \in N_k$ a number of $M_j \cong M*P_{dj}$ packets will be routed through j [11].

In each node, packets will decide randomly whether to use the usual method (random) or the deterministic method, in order to choose the jumping node. Particularly, the best behavior was observed for P=0.5, where P is the probability of using the random method, normally used in AntNet1.0. 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 using the deterministic method, when it travels to the destination d. For AntNet1.0, P=1.

Control of the number of ants inside the network

Original versions of AntNet do not 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, the total number of ants was limited to an amount four times the number of network nodes, because this is an average number of links for each node in the networks used (Figs. 1,2). With this method, simulation results were improved.

Seft-destruction of Ants

In order to avoid infinite loops, self-destruction of a forward ant $F_{s\to d}$ occurs if the amount of jumps in a cycle is higher than half of the already accumulated number of jumps.

When a backward ant $B_{s\to d}$ can not return to its source node because its return trip was interrupted, due to either a link or node failure, it is self-destroyed, because the information stored in its stack does not reflect anymore the real state of the network. Regarding the implementations, these situations were important, so they were added to AntNet1.0 and AntNet1.1.

EXPERIMENTAL RESULTS

All the algorithms mentioned before, were implemented with a parallel behavior simulated with serial code. A data traffic simulation analysis was done for each time slot.

The parameters used in order to evaluate each algorithm performance are:

- *Instantaneous Packet Delay*. It is the average delay of all data packets routed successfully for a given time slot *t* during an algorithm simulation.
- *Average Packet Delay*. It is the average delay of all data packets well routed during the whole simulation period.
- *Instantaneous Throughput*. It is the amount of packets routed successfully for a given time slot *t* during an algorithm simulation
- *Average Throughput*. It is the average amount of packets routed successfully during the whole simulation period.
- A benchmark was established for the simulations. Twelve simulation scenarios, as shown in Table I, composed this benchmarkc.

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

TABLE I: Benchmark used to analyze each of the above paradigm network.

For each simulation cycle, a traffic simulator stops generating packets when a certain fraction (expressed in %) of the generated packet does not arrive to destination (Lost Packet threshold). The link transmission delay is used as metric for link costs, expressed in milliseconds.

For simulations, three networks were used as models:

- A fictitious simple network of 8 nodes and 9 links for extensive simulations [4].
- The NSFNET network, of the National Science Foundation (United States), with 14 nodes and links of 1.5 Mbps (Fig.1 shows the net with links delay in [ms]).

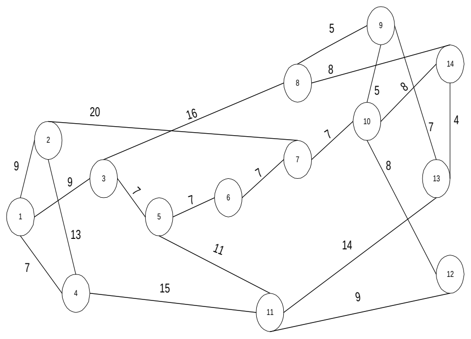

Fig.1. NSFNET

- The NTTnet network, of the Nippon Telephone Telegraph (Japan), with 57 nodes and links of 6 Mbps (Fig. 2).

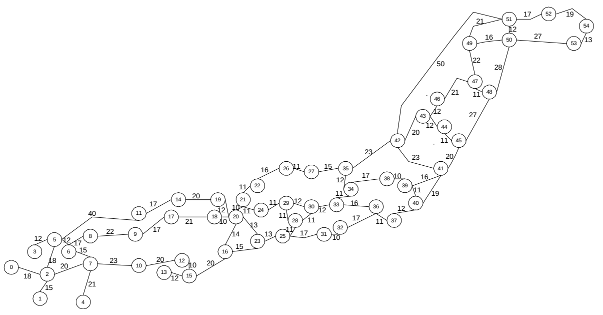

Fig. 2. NTTnet

Figures 3 to 8 and Tables II and III show the simulation results for some of the experiments performed for the last two networks and only for medium traffic. In the tables that follows, THR means average throughput and AVP: average packet delay. Other abbreviations are AntNet1.0 = A1.0 and AntNet1.1 = A1.1.

Experimental results with the NSFNET

Table II shows results of average parameters for a transient regime experiment for AntNet algorithms and for a transitory link failure (link 5-6, Fig. 1). Figures 3-4 show the instantaneous average delay and throughput for a typical experiment, concluding the following:

- Transient Regime. It can be observed in Table II how A1.1 "learns" quicker (better throughput and packet delay) than A1.0. This is due mainly to the routing tables intelligent initialization and the use of dual method for hop node selection.
- Link 5-6 Failure: Throughput. RIP and OSPF throughput decreases completely at the instant of the failure (Fig. 3); however, AntNet algorithms are not severely affected, demonstrating their robustness. A1.1 has the best instantaneous and average throughput (Fig. 3 and Table II). This is due mainly to the routing tables intelligent reinitialization method. LBR had the worst performance.

		RIP	OSPF	LBR	A1.0	A1.1
Transient Regime	THR [packets]				4716.79	5079.46
	AVP [ms]				27.17	24.02
Link 5-6 Failure	THR [packets]	4347.61	4450.33	4090.09	4844.56	5174.47
	AVP [ms]	21.06	20.1	28.7	25.58	23.89

Table II: Experimental Results for average throughput and packet delay

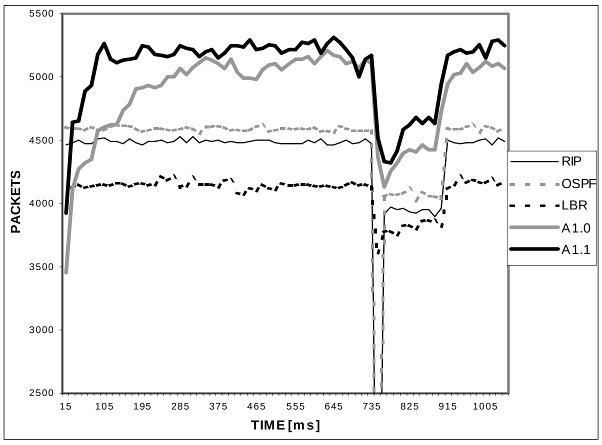

Fig. 3. NSFNET Link 5-6 failure. Instantaneous throughput

- Link 5-6 Failure: Packet Delay. All algorithms are proportionally affected (see Fig. 4) during the failure. RIP and OSPF maintain an inherent advantage in this figure of merit (see Fig. 4 and Table II). Here, A1.1 overcome A1.0 again, in both instantaneous and average packet delay. Again LBR had a poor performance.

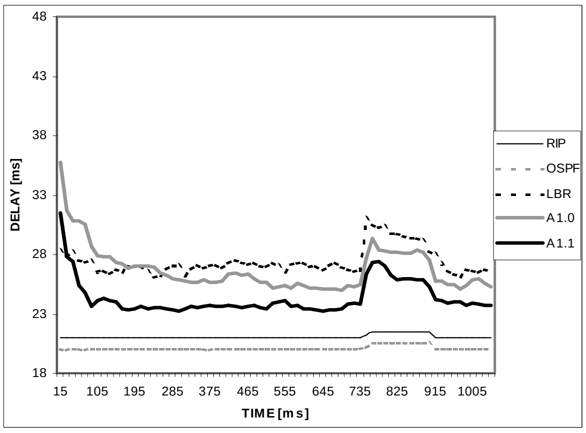

Fig. 4. NSFNET Link 5-6 failure. Instantaneous packet delay

Experimental Results with the NTTnet

In what follows, simulation results using NTTnet (Fig. 2) for node failure and hotspot experiments are discussed.

- Node 37 failure: Throughput. The robustness of AntNet algorithms can be observed, with relationship to RIP and OSPF, at the instant of the failure (see Fig. 5). However, A1.0 has the slowest recovering after the node failure. Particularly, A1.1 has the best average throughput (see Table III).

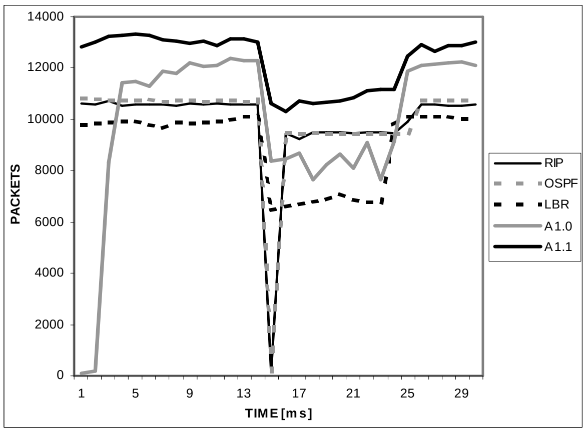

Fig. 5. NTTnet Node failure. Instantaneous throughput

- Node 37 Failure: Packet Delay. It is observed that all the algorithms are affected proportionally (see Fig. 6). A1.1 show a smaller average and instantaneous packet delay than A1.0 (see Table III and Fig. 6). A1.0 just was better than LBR in this experiment.

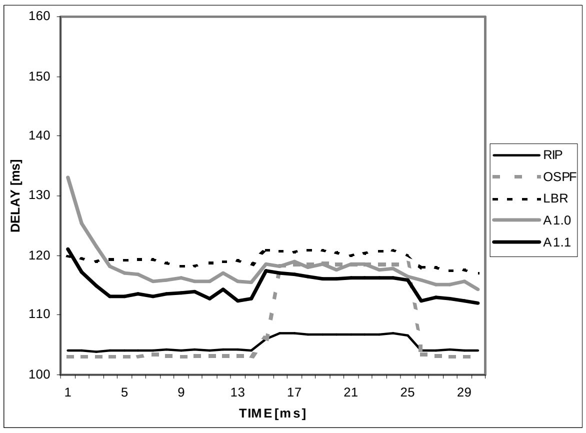

Fig. 6. NTTnet Node failure. Instantaneous packet delay

- Transient Hotspot: Throughput. Node 41 was chosen as a hotspot. Again, in instantaneous an average throughput, A1.1 has the best performance (Fig. 7,Table III).

		RIP	OSPF	LBR	A1.0	A1.1
Node 37 Failure	THR [packets]	9999.03	9977.27	7976.7	9886.11	12268.34
	AVP [ms]	105.08	109.08	122.34	117.75	114.75
Transient Hotspot	THR [packets]	8736.23	8848.26	8717.98	9423.13	11759
	AVP [ms]	104.26	102.63	116.76	116.25	112.58

Table III: Experimental results for average throughput and packet delay

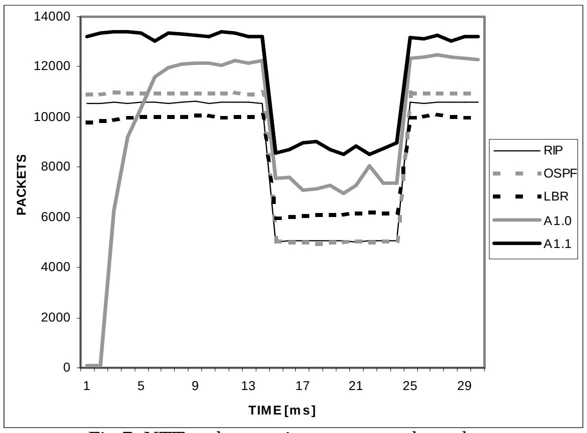

Fig.7. NTTnet hotspot instantaneous throughput

- Transient Hotspot: Packet Delay. During the hotspot the delay of the algorithms is smaller due to the geographical position of the hotspot (Fig. 3), which is approximately equidistant to all nodes. According to Figure 8 and Table 3, again A1.1 has a better behavior than A1.0.

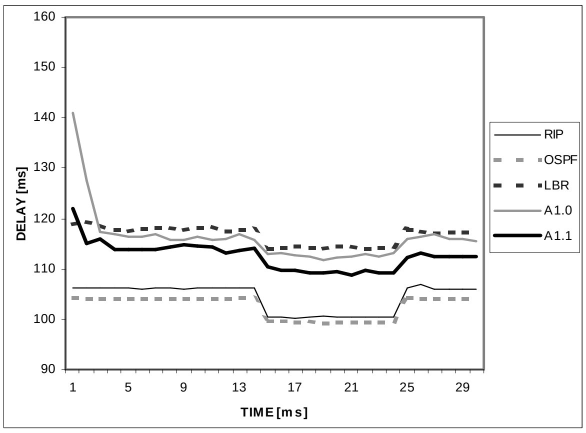

Fig. 8. NTTnet hotspot instantaneous packet delay

After the analysis of simulation results for each of the 12 scenarios for the three tested networks (for a total of 36 experiments), the following general conclusions can be inferred:

- In all our experiments, AntNet1.1 had a shorter transient regime, a better throughput and a shorter packet delay than A1.0, demonstrating the improvements of the modifications here proposed.
- AntNet algorithms are more robust than RIP, OSPF and LBR algorithms, in the case of link and node failure, because their instantaneous throughput does not decay completely at the instant of a failure (see Figs. 5 and 7). However, they have a slower recovery than RIP and OSPF, during these failures.
- RIP and OSPF had always less throughput than AntNet1.1; however, they always performed better in packet delay, because RIP and OSPF mainly optimize delay, relegating throughput to a second plane, as it was previously discussed. However, this characteristic becomes a disadvantage, because the current simultaneous demands of network services are growing fast, consequently, throughput becomes a new priority.

CONCLUSIONS

This work introduced AntNet, a novel adaptive routing technique for data networks, based on mobile agents, whose use is currently oriented towards packet switching networks, such as Internet. After presenting the original versions, the best original AntNet algorithm (here called AntNet1.0) was briefly described. Several modifications of AntNet1.0 were proposed, and a final version was called AntNet1.1.

AntNet algorithms, in addition to RIP, OSPF and LBR (Load Balancing Routing, in development phase [2]) were implemented and simulated. A better performance of AntNet1.1 with respect to throughput was observed throughout all our experiments for three types of traffic called: low, medium and high and for each of the three tested networks. The modifications implemented in AntNet1.1 that contributed the most for a better behavior were: routing tables intelligent initialization and the dual method of selecting jumping nodes.

In general, results of different experiments remained with the same patterns. RIP and OSPF showed a smaller instantaneous and average packet delay, in all our experiments and for the three types of traffic. Results obtained in a different simulation scope suggest that AntNet algorithms could have better throughput as well as packet delay than the other traditional algorithms [7-9]. If this is the case, it is equally expected that AntNet1.1, proposed in this paper, will have a better performance than AntNet1.0, given that our modified version outperform the original AntNet algorithm in all the experiments.

Based on the performed experiments, it is also expected an efficient AntNet1.1 behavior with: flow control, congestion and admission schemes. Therefore, it can be inferred that a commercial implementation of this algorithm may be feasible and its use can be considered for large networks, such as Internet, as a future option when throughput is the main concern.

REFERENCES

[1] Almirón M., Barán B. & Chaparro E., “Ant Distributed System for Solving the Traveling Salesman Problem,” *XXV Informatic Latinoamerican Conf.-CLEI*, Paraguay, pp.779-789, 1999.

[2] Bak S., Cobb J. & Leiss E., “Load Balancing Routing via Randomization,” *XXV Informatic Latinoamerican Conf.-CLEI*, Asuncion-Paraguay, pp.999-1010, 1999.

[3] Barán B. & Sosa R., “A New Approach for AntNet Routing”, International Conference on Computer Communication and Networks IEEE ICCCN-2000, Las Vegas - Estados Unidos. 2000.

[4] Barán B. & Sosa R., “AntNet: Routing Algorithm for Data Networks based on Mobile Agents”, Argentine Symposium on Artificial Intelligence ASAI’ 2000, Buenos Aires, Argentina, 2000.

[5] Dorigo M., Maniezzo V. & Colorni A., “The Ant System: Optimization by a colony of cooperating agents,” *IEEE Trans. Systems, Man, and Cybernetics*-Vol.26,N 1, pp.1-13, 1996.

[6] Dorigo M. & Gambardella L., “Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem,” *IEEE Trans. on Evol. Computation*, Vol.1, N 1, pp.53-66, 1997.

[7] Dorigo M. & Di Caro G., “AntNet: A Mobile Agents Approach to Adaptive Routing,” Tech. Report, IRIDIA- Free Brussels University, Belgium, 1997. http://iridia.ulb.ac.be/dorigo/ACO.

[8] Dorigo M. & Di Caro G., “Ant Colonies for Adaptive Routing in Packet-switched Comm. Networks,” Technical Report, IRIDIA-Free Brussels University, Belgium, 1998.

[9] Feit S., *TCP/IP: Architecture, Protocols and* Dorigo M. & Di Caro G., “AntNet: Distributed Stigmergetic Control for Communications Networks,” Journal of Artificial Intelligence Research, Number 9, pp. 317-365, 1999. *Implementation*, McGraw Hill, 2nd Edition, 1996.

[10] Goldberg D., *Genetic Algorithms in Search, Optimization, a Machine Learning*, Addison-Wesley, 1989.

[11] Schoonderwoerd R., Holland O. & Bruten J., “Ant-like agents for load balancing in telecommunications networks,” Technical Report, Hewlett-Packard Laboratories, Bristol-England, 1997. http://sigart.acm.org:80/proceedings/agents97/.

[12] Shankar A., Alaettinoglu C. & Matta I., “Performance Comparison of Routing Protocols using MaRS: Distance Vector versus Link-State,” Technical Report, Maryland-USA, 1992.

[13] Tanenbaum A., *Computer Network*,, Prentice & Hall, Third Edition, 1996.