AGENT-BASED ROUTING ALGORITHMS ON A LAN

Y. Yang, A. N. Zincir-Heywood, M. I. Heywood, S. Srinivas
Faculty of Computer Science, Dalhousie University
{yang@cs.dal.ca, zincir@cs.dal.ca, mheywood@cs.dal.ca, srini@cs.dal.ca}

Abstract

The AntNet algorithm for adaptive routing is implemented on a LAN using the TCP/ IP protocol. A study is made of the relative merits of different reinforcement parameters central to the stable operation of the algorithm. The case of a constant reinforcement leads to slow but dependable performance whereas adaptive reinforcement appears to be sensitive to the window over which statistics are estimated.

Keywords: *Computer networks, routing algorithms, reinforcement learning, ants routing algorithms.*

1. INTRODUCTION

Routing plays a critical role in communication networks in determining the overall network performance in terms of throughput and transmission delay. Research in social insect behaviour has provided computer scientists with powerful methods for designing distributed control and routing algorithms [1, 2]. The ant routing algorithm, inspired by the indirect model of communication observed in ant colonies, has received a lot of interest. In such a concept, ant-like agents explore the network, exchange collected information in an indirect way, and update routing tables of the network nodes, where all of these are inspired from how ants find the shortest path from their nests to their food.

In nature, ants lay down a thin layer of signaling chemicals called pheromones [1], wherever they travel to find food. When other ants detect these pheromones, they instinctively follow the path the chemicals mark. The thicker the pheromone trail, the more likely other ants will follow the path. Moreover, the ant itself is not a complex insect. For any ant considered individually, it has very simple and limited behaviors. In fact, all of its movements are based on immediate reactions to its surroundings or to its fellow ants. However, ants are social insects. By acting as a group, they represent a highly structured and complex social organization. Due to their social organization, complex tasks are performed that cannot be performed by individuals alone are solved in a distributed manner. Such general properties have resulted in an interest in applying the methodology to problems in computer networks and telecommunications [1, 2].

This paper presents the design and implementation of an ant routing algorithm in a real IP datagram based local area network (LAN) environment. To this end, Di Caro and Dorigo's AntNet routing algorithm [2] is used as a starting point and modified to be implemented in a real IP based LAN. Moreover, specific cases of ants with constant reinforcement and dynamic or adaptive reinforcement are investigated and tested under faulty network conditions. To the best of our knowledge, this will be the first time these algorithms are implemented.

In section 2, the AntNet routing algorithm of Di Caro and Dorigo is introduced. Section 3 presents the details of the test environment; whereas results of these tests are given in section 4 and conclusions are drawn in section 5.

2. DESCRIPTION OF AntNet ROUTING ALGORITHMS

There are two types of ants used in these algorithms, namely, *forward ants* and *backward ants* [2]. Forward ants have the same priority as normal data packets. They collect the traffic information on the network using a time cost with respect to a given destination and the routing decisions taken by the forward ant. When forward ants reach their destination, they become backward ants. Backward ants update routing information on all the nodes that the forward ants have visited. To do so higher priority queues are used than forward ants and data packets, so routing information along the path taken is updated as soon as possible. Forward ants and backward ants communicate the quality of paths in an indirect way; through the

information they currently read and write in two data structures stored in each network node, *k*, figure 1.

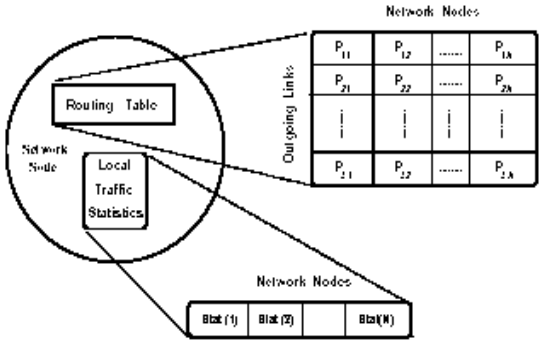

Fig. 1. The data structure of a given node [2]

The routing table T_k contains probabilistic entries. Each entry P_{dn} expresses the probability of choosing *n* as next node when the destination node is *d*. All routing table entries conform to the constraint,

$$\sum_{n \in N_k} P_{dn} = 1, d \in [1, N], N_k = \{neighbor\,(k)\}$$

A statistical model summarizes the traffic distribution over the network as seen by the node, *k*, using the tuple $M_k(\mu_d, \sigma_d{}^2, W_d)$. The model contains sample means μ_d and variances σ_d^2 as evaluated across a moving observation window length, W_d, with respect to destination node, *d*.

New forward ants, F_{sd}, are created periodically, but independently of the other nodes, from source, *s*, to destination node, *d*, in proportion to the destination frequency of passing data packets. Forward ants travel through the network using the same priority structures as data packets, hence are subject to the same delay profiles.

The next link in the forward ant route is selected stochastically, *p'(j)*, in proportion to the routing table probabilities and length of the corresponding output queue.

$$p'(j) = \frac{p(j) + \alpha l_j}{1 + \alpha(|N_k| - 1)}$$

where $p(j)$ is the probability of selecting node *j* as the next hop; α weights the significance given to local queue length verses global routing information, $p(j)$; l_j is the queue length of destination '*j*' normalized to the unit interval; and N_k is the number of links from node *k*.

On visiting a node different from the destination, a forward ant checks for a buffer with the same identifier as itself. If such a buffer exists the ant must be entering a cycle and dies. If this is not the case, then the ant saves the previously visited node identifier and time stamp at which the ant was serviced by the current node in a buffer with the forward ant's identifier. The total number of buffers at a node is managed by attaching "an age" to buffer space and allowing backward ants to free the corresponding buffer space.

When the current node is the destination, $k = d$, then the forward ant is converted into a backward ant, B_{ds}. The information recorded at the forward ant buffer is then used to retrace the route followed by the forward ant.

At each node visited by the backward ant, routing table probabilities are updated using the following rule,

IF (node was in the path of the ant)
THEN $p(i) = p(i) + r\,\{1 - p(i)\}$
ELSE $p(i) = p(i) + r\,\mathrm{P}(i)$

where r ∈ [0, 1] is the reinforcement factor central to expressing path quality (length), congestion and underlying network dynamics.

The selection of reinforcement parameter *r* is of particular interest in this work. Two different methods are considered: 1) set the value of *r* to a *constant*; or 2) change the value of *r* dynamically.

Setting *r* to a *constant* implies that the significance of ant arrival rates remains unchanged irrespective of the network conditions. Ants traveling along better paths will arrive at a higher rate than other ants, thus their paths will have a higher probability than other paths. Naturally, every backward ant has the same effect on the routing table no matter how good the path that it finds is. To solve this problem, Di Caro and Dorigo recommended the dynamic reinforcement method [2].

According to this, the reinforcement factor should be a factor of trip time and the local statistical model of the node neighborhood. To this end, the following relationship is introduced [2];

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

In the above equation, c_1 and c_2 are constants that weigh the importance of each term and t_{ant} is the actual trip time taken by the ant. Moreover;

$$I_{inf} = W_{best};$$
$$I_{sup} = \mu_{kd} + W^{0.5}\{\sigma_{kd} / (1 - \gamma)\}.$$

where γ is a constant, determining the confidence interval; W_{best} is the best case trip time to destination d over a suitable temporal horizon, W.

The estimates for mean, μ_{kd}, and variant, σ_{kd}, of the trip time are also made iteratively, using the trip time information. Thus,

$$\mu_d = \mu_d + \eta(o_{kd} - \mu_d)$$
$$(\rho_d)^2 = (\rho_d)^2 + \eta\{(o_{kd} - \mu_d)^2 - (\rho_d)^2\}$$

The reinforcement value r obtained from the above equation is finally transformed by a monotonic function over the unit interval. This makes the system more sensitive to a high value of r, less sensitive to a low value of r, $s(r)/s(1)$:

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

The definition of function $s(x)$ is

$$s(x) = \left(1 + \exp\left(\frac{a}{x \mid N_k \mid}\right)\right)^{-1}, x \in (0,1], a \in R^+$$

Here, a, is a positive real number, and $|N_k|$ is the number of neighborhood nodes of the current node, k.

From the above algorithm, it is, therefore, apparent that by dynamic reinforcement, ants are able to make decisions under more uncertainty than was previously the case.

3. IMPLEMENTATION OF AntNet

In this study, both versions – the constant reinforcement learning and the dynamic reinforcement learning – of AntNet are implemented (in C on an IP-based LAN, figure 2). As stated above, the objective is to study the applicability/implement-ability of the algorithms on a real network, and compare their performances under different experimental conditions.

In order to study/explore, how ant routing algorithms work on a real network environment, five multi-homed computers are connected to each other to form a local area network. Each node runs Windows and standard TCP/IP and is an AMD K6-266MHz processor, with a 32MB RAM and up to 3 10M/100M NICs.

As it can be seen in figure 2, each computer on the LAN has one or more network cards, where the ones, which are directly connected by cable, are grouped into the same sub-network. In this topology, links or cables are all in different sub-networks, and each one works in full duplex mode.

Furthermore, the time synchronization software *AboutTime* [3] is installed on each node to enable ants to collect the trip time information in order to calculate dynamic reinforcement values, hence real environmental feedback. *AboutTime* uses Simple Network Time Protocol (SNTP) to synchronize time on the Internet or in a local network. The Simple Network Time Protocol (SNTP) is described in RFC 1769 [4]. It's an adaptation version of the Network Time Protocol (NTP), which is used to synchronize computer clocks on the Internet. With *AboutTime*, one can achieve synchronization accuracies of $\pm$50 milliseconds typically [3].

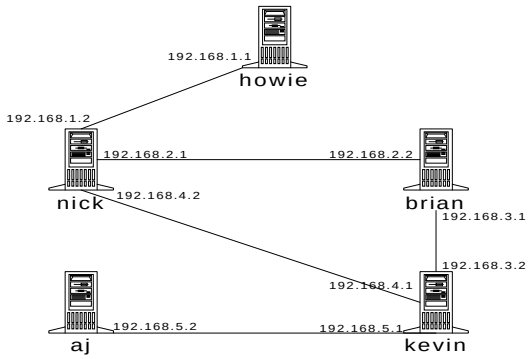

Fig. 2. The network topology of the testbed

4. TEST RESULTS

Since the aim of our experiments is to explore how the ants' routing algorithms (constant reinforcement and dynamic reinforcement) behave and compare their performances, we therefore performed tests under three different traffic load conditions (different generating rates of data packets) for both algorithms. Moreover, under each condition, four different generating rates of forward ants (F-Ants) are used. Table 1 gives the details for these test scenarios.

Table 1. Different test scenarios

No Data Packets	Data Packets 10/sec	Data Packets 20/sec
F-Ants 1/min	F-Ants 1/min	F-Ants 1/min
F-Ants 2/min	F-Ants 2/min	F-Ants 2/min
F-Ants 1/sec	F-Ants 1/sec	F-Ants 1/sec
F-Ants 2/sec	F-Ants 2/sec	F-Ants 2/sec

In the results presented below, 100% stacked area charts are used to display the changes/trends of choosing a link in a routing table for a particular destination. The changes are represented as probabilities over time. Different colors (white, gray and black) represent different percentage values in the same column of a routing table on a network node. This indicates the probability of selecting that link as the next hop for a given destination. Parameters configuring the AntNet algorithm follow the recommendations in [2], and are summarized in Tables 2 and 3.

Table 2. Parameters for Constant Reinforcement Learning

Constant Reinforcement Ants' Algorithm	
Parameters	**Value**
Reinforcement value r	0.1
Next hop selection	Uniform distribution [0..1]

Table 3. Parameters for Dynamic Reinforcement Learning

Dynamic Reinforcement Ants' Algorithm	
Parameters	**Value**
C_1	0.7
C_2	0.3
Z	1.7
A	2.5
η	0.05
Sliding window size ($\|W\|$)	100
Next hop selection	Uniform distribution [0..1]

Figures 3 to 6, demonstrate that ants can actually adapt to the environment much better as the load on the network increases. On the other hand, if the load is kept the same but the generation rate of forward ant is decreased then the ants working with the dynamic reinforcement algorithm perform much better, figures 7 and 8.

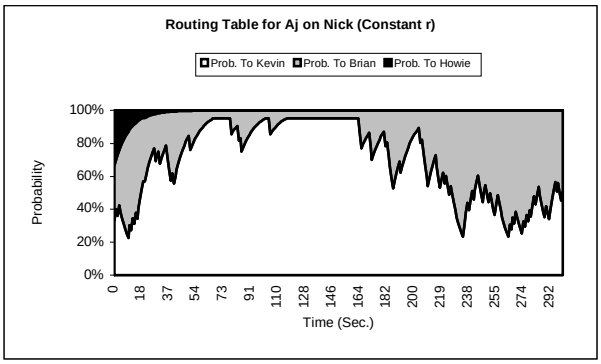

Fig. 3. Routing table for Aj on Nick (constant *r*, no data packet, F-Ant rate = 1/sec.)

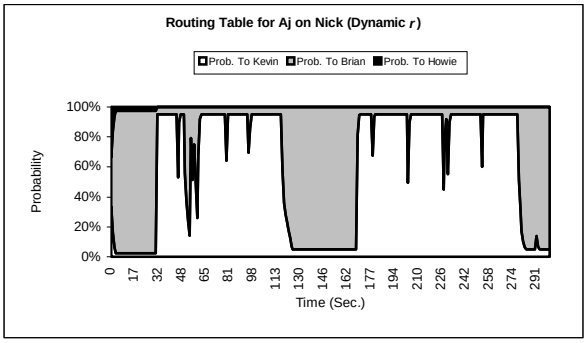

Fig. 4. Routing table for Aj on Nick (dynamic *r*, no data packet, F-Ant rate = 1/sec.)

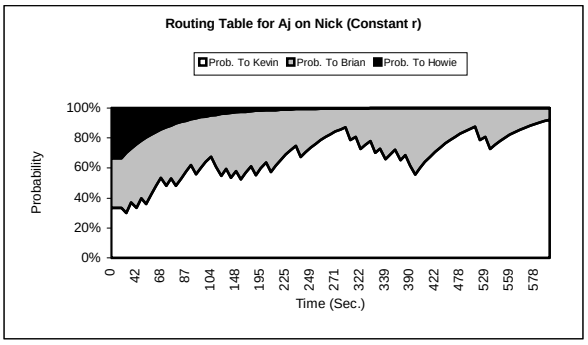

Fig. 5. Routing table for Aj on Nick (constant *r*, data packet rate = 20/sec, F-Ant rate = 1/sec.)

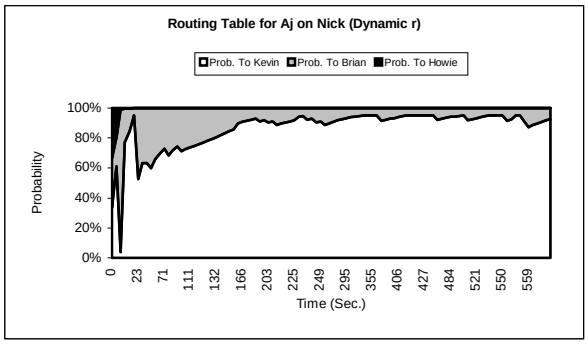

Fig. 6. Routing table for Aj on Nick (dynamic *r*, data packet rate = 20/sec, F-Ant rate = 1/sec.)

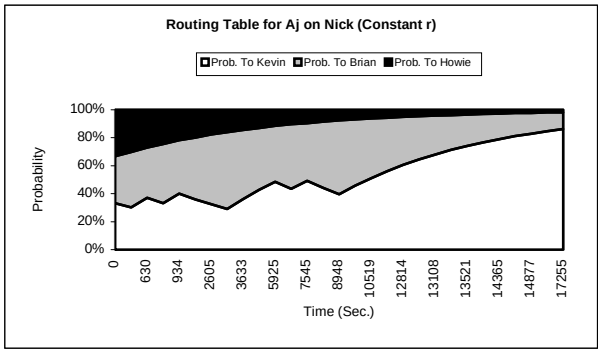

Fig. 7. Routing table for Aj on Nick (constant *r*, data packet rate = 20/sec, F-Ant rate = 1/min.)

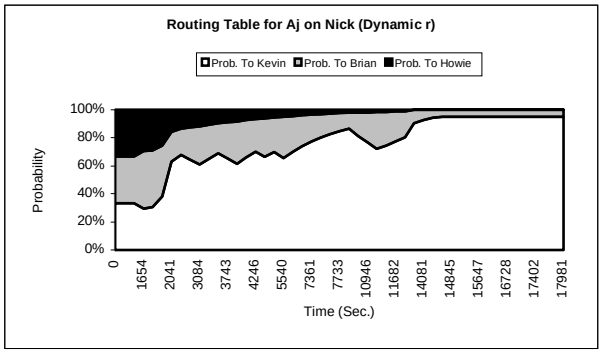

Fig. 8. Routing table for Aj on Nick (dynamic *r*, data packet rate = 20/sec, F-Ant rate = 1/min.)

From these results it is apparent that the constant reinforcement case provides for a continuous gradual change in routing strategy, typically utilizing multiple routes at a time. The dynamic reinforcement, under the same conditions, appears to provide a bang-bang profile with hysteresis.

In the next scenario, the performances of the two algorithms are tested to study how they adapt in terms of switching to an alternative path/link, when the link with the highest probability becomes blocked or down. In this test, at the very beginning, the scenario starts with all the links up and running with a 2000-ms time delay on the link from node *Nick* to node *Kevin*. Moreover, about 90 seconds later, we unplug the cable from *Nick* to *Brian* to see how the ants will behave. The changes on the routing tables are given below, figures 9 and 10.

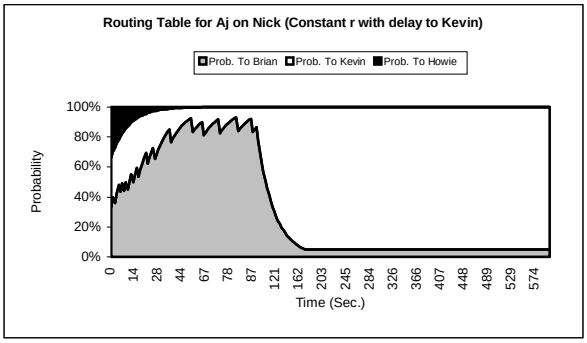

Fig. 9. Routing table for Aj on Nick (constant *r*, no data packets, F-Ant rate = 1/sec., link to Brian is down)

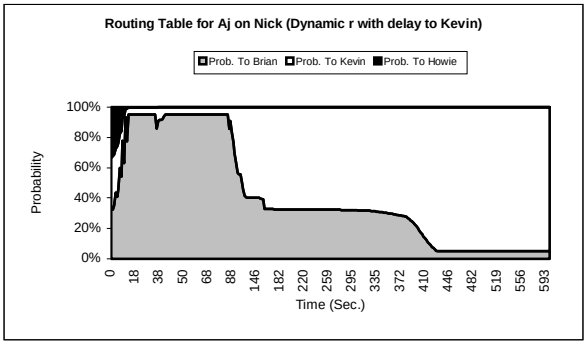

Fig. 10. Routing table for Aj on Nick (dynamic *r*, no data packets, F-Ant rate = 1/sec., link to Brian is down)

When the same scenario is tested under more loads, it is observed that the adaptive method switches to 'Kevin' in a series of discrete steps, figures 11 and 12. Thus, constant reinforcement algorithm shows a better performance to find a new path than the dynamic reinforcement algorithm, when the previously found good path/link is down. Constant reinforcement learning appears to let ants following longer paths to have equal weighting (per backward ant) as the ants following shorter paths. Hence, the results are more

fluctuant, thus it is easier to find a new path, when the old path is blocked. On the other hand, during the dynamic reinforcement learning, ants need to compare the time cost with the previous short time cost, as well as the mean time cost. In other words, since the new path is longer than the previous path, it will take more time for dynamic reinforcement learning algorithm to find an alternative path, i.e., dynamic reinforcement is sensitive to the window width, W_d.

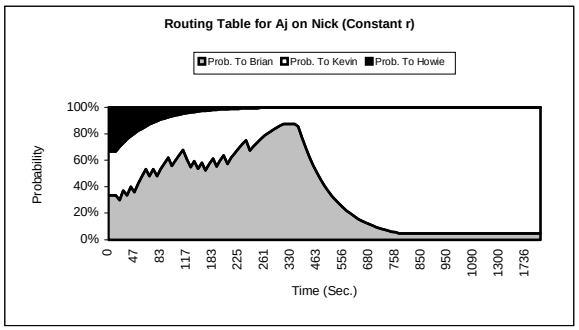

Fig. 11. Routing table for Aj on Nick (constant *r*, data packet = 20/sec, F-Ant =1/sec., link to Brian is down)

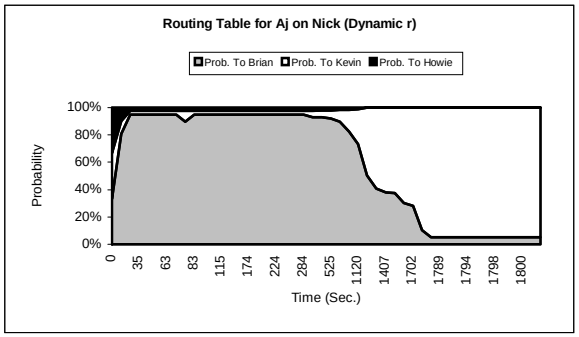

Fig. 12. Routing table for Aj on Nick (dynamic *r*, data packet = 20/sec, F-Ant =1/sec., link to Brian is down)

5. CONCLUSION

In this work, ant routing algorithms with constant and dynamic reinforcement learning are implemented. As indicated before, both of these routing algorithms make use of collective behavior arising from the interactions between ants to find the shortest path to the destination. These two ant routing algorithms are tested on an IP based LAN environment under different network traffic conditions. These tests demonstrate that both algorithms are able to find paths autonomously. However, it is observed that a dynamic reinforcement algorithm has better performance under heavy network traffic than the constant case. It takes approximately 30 seconds for the dynamic algorithm to adapt to the load on the network, and to find a good path with approximately 98% accuracy. Thus, a dynamic reinforcement algorithm can find the correct path more quickly. Moreover, once a good path is found, the dynamic reinforcement algorithm provides a more stable platform than the constant reinforcement algorithm. However, because of this very characteristic, it is quicker to switch to an alternative path for the constant reinforcement learning than for the dynamic reinforcement learning under faulty network conditions.

Furthermore, although, to the best of our knowledge, this implementation is the first of its kind, and shows that the adaptive ants routing algorithm can work on real computer networks, there is more work to be done in order to test the system for its reliability and robustness. Tests on different network topologies and bigger networks as well as using different synchronization methods need to be performed. In effect both algorithms for defining reinforcement are actually sensitive to specific parameters: r in the case of a constant reinforcement, W_d in the case of dynamic reinforcement.

Acknowledgements

The authors gratefully acknowledge the support of NSERC for funding provided by the research grants of Drs. Zincir-Heywood, Heywood, and Srinivas.

References

[1] R. Schoonderwoerd, O. Holland, J. Bruten, L. Rothkrantz, “Ant-Based Load Balancing in Telecommunications Networks”, *Adaptive Behavior*, 5, 169-207, 1997.

[2] G. Di Caro, M. Dorigo, “AntNet: Distributed Stigergetic Control for Communication Networks”. *Artificial Intelligence*. 9, 317-365, 1998.

[3] The web site of the *AboutTime* software, http://www.arachnoid.com/abouttime/

[4] D. Mills, “Simple Network Time Protocol (SNTP)”, *RFC 1769*, University of Delaware, March 1995