PACKET ROUTING WITH GENETICALLY PROGRAMMED MOBILE AGENTS

Jon Schuringa
Günter Remsak
Institute of Communication Networks
Technical University Vienna, Austria
{jon.schuringa,guenter.remsak}@tuwien.ac.at

Abstract This paper presents novel mobile agent routing techniques. We use genetic programming to build mobile agents that monitor the network status and set the routing tables in the network nodes in such a way that it maximizes network throughput and minimizes the overall packet delay. Performance is measured by simulation using a realistic network and traffic model that is capable of generating fractal traffic. The result is a high performance, self-configuring routing method, irrespective of the network topology and traffic.

Keywords: Routing, Mobile Agents, Genetic Programming

1. INTRODUCTION

The increasing complexity and diversity of communication networks make them hard to manage, managing them efficiently when traffic is strongly fluctuating is even harder. It is widely known that the mobile agents concept can be used to provide the basis for a self-configuring network (e.g. [2, 3]).

We describe in this paper a self-configuring routing system, which is based on "AntNet" [1]. The idea used has its origins in the behavior of real ants. Real ants are capable of finding shortest paths by using information (pheromones) deposited by other ants [4, 5].

Apart from significant improvements in the AntNet algorithm, we use genetic programming techniques to build mobile agents that monitor the network status and set the routing tables in the network nodes in such a way that it maximizes network throughput and minimizes the overall packet delay.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35522-1_37

H. R. van As (ed.), *Telecommunication Network Intelligence*

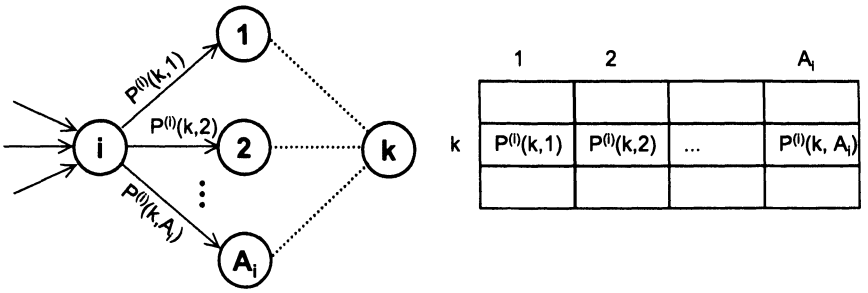

Figure 1 Routing table $P^{(i)}$ at node i

The routing table (Fig. 1) indicates how the packets arriving at that node must be routed on the outgoing links, depending on the final destination of the packets. The routing table for node i is a matrix $P^{(i)}$ with dimension $N \times A_i$, where N is the number of nodes in the network and A_i is the number of neighbors of node i. $P^{(i)}(k,j)$ is the fraction of traffic with destination k, that is routed through neighbor j at node i. It is the task of the routing agents, described in the following section, to set the routing table.

2. AGENT ROUTING

This section will first give a summary of the agent routing algorithm and then discusses the differences with the AntNet system.

Agents are generated concurrently with normal packets, but far less frequently. The main task of the agent on the path to its destination is to monitor and collect the network condition on the route between source and destination. The agent takes the same route as normal packets, according to the probability in the routing table inside the visited node, however for agents the probability is adjusted: we increase the probability for outputs with small queue sizes. This mechanism helps the agent in finding new, better routes. When the agent arrives at its destination, it takes the same route backwards and updates, according to the collected information and locally available information in the node, the routing tables at every visited node. The agent dies when it returns back to its source node.

Below are the main differences between our system and the AntNet system:

- High priority agents

  In AntNet the backward agents have higher priority than normal data packets, to quickly propagate the information the forward

agent collected. Forward agents have the same priority as data packets, so they also experience the same network conditions. The fact that agents experience these delays is used by the algorithm (it is one of the main principles).

The disadvantage is however that it could a long time before an agent can react to specific situations like congestion. Forward routing systems [7, 8] do not inhabit the slow round trip delay.

In our system we use backward routing, however our agents always have a higher priority than normal data packets. They no longer experience the normal packet delay; they calculate it by using the current queue size and the link capacity. On the return path the calculated delays are used as if they were the real experienced delays. The advantage is that the same information is propagated much faster in the network. The effect on the network performance will be discussed later.

- Collecting neighboring data

  Directly related to the previous point, we also know the transmission queue sizes of all neighboring nodes along the visited path. This information is collected by our agent also, but only if the delay is small enough. To be precise, we only use the information if the calculated delay to the neighbor is smaller than the mean delay to that neighbor. Calculating the extra delays to neighboring nodes complicates the algorithm, because it may happen that more than one possible path between two nodes occur. Section 2.1 gives a detailed explanation of the problem and its solution.

- Inforcement rule

  For each node that the backward agent visits, we update the probability routing table. The amount should be proportional to the goodness of the trip. We use genetic programming to find a way to measure the goodness of the trip, it should find the best trade-off between adaptivity and stability.

  Different than in the AntNet system, we allow negative inforcements also. Very bad scoring trip times could be negatively awarded. The inforcement rule gives a measure of how good the agent's trip was. This measure, which is a number between -1 (bad) and 1 (good), is used to update the routing table. An inforcement of 0 will leave the probability unchanged. Let r denote the inforcement, then the following formula defines how to update the probability: let P be the short form of $P^{(i)}(k, o)$ (i.e., the current probability

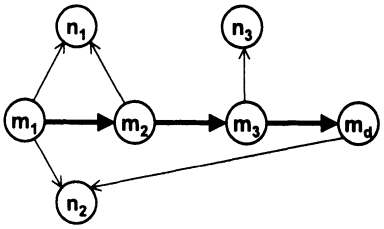

Figure 2 Example of the delay graph; The nodes on the main path m_i are connected by bold arrows, the other nodes are neighboring nodes n_j of which we have the delay info as well.

to use output o for packets with destination k at node i). The new probability P' is defined by

$$R_{plus} = r\left[\frac{1}{2} + \left(\frac{1}{2} - P\right) sign(r)\right] \tag{1}$$

$$P' = P + R_{plus} \tag{2}$$

By normalization, the other probabilities P_j (all outputs not used by the agent) should be adjusted:

$$P'_j = P_j\left(1 - \frac{R_{plus}}{1 - P}\right) \tag{3}$$

The main difference in the algorithm between our system and AntNet is that our agents inspect the transmission queues, allowing lower agent delays and more network information per agent. We call it the "Queue Inspecting Agents" (QIA) system.

2.1. EFFICIENT ALL PAIRS SHORTEST PATH ALGORITHM

The method we applied for collecting delay information along the visited path, together with information about neighboring nodes could lead to the existence of multiple paths between two node pairs. We are only interested in the shortest path, so we have to apply some sort of shortest path algorithm. We could use Floyd's or Dijkstra's algorithm, however, due to their computational complexity ($O(N^3)$ and $O(NE \log N)$ respectively), not well suited for our purposes. If we make a graph of the delay information (Fig. 2), we can see that we have a graph with special properties for which we developed a simple and efficient algorithm based on theorem (4). To compute the shortest path between all pairs

has computational complexity $O(N^2)$. Note however that the algorithm is processed in different nodes and each node on the return path only needs to know the delay from itself to the other nodes, which can be computed in $O(N)$ time and therefore the algorithm is scalable.

The main idea is based on two properties:

- The vertices in the graph that represent neighboring nodes, have no outgoing edges.
- Let $m_1, m_2, ..., m_k$ represent the nodes the agent visited on the forward path. There exists one and only one path from m_i to m_j for $1 \leqslant i < j \leqslant k$, while in the other direction (from m_j to m_i) no paths exists.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ a weighted, directed graph. $\mathcal{V}$ consists of the sum of the visited vertices $\mathcal{V}_m$ (the main agent track) and $\mathcal{V}_n$ (the neighboring vertices), respectively. Note that the outgoing degree of the vertices in $\mathcal{V}_n$ equals 0. Now let $\mathcal{V}_m = \{m_1, m_2, ..., m_k\}$, i.e., the agent started in vertex (node) m_1 and visited all m_i until m_k. The edges between two vertices in $\mathcal{V}_m$ are directed only from $m_i \rightarrow m_{i+1}$. We claim that the shortest path $\mathcal{D}$ between vertices from $\mathcal{V}_m$ and all other vertices in $\mathcal{V}$ equals

$$\mathcal{D}(m_i, y) = \min[\mathcal{D}(m_{i+1}, y) + w(m_i, m_{i+1}), w(m_i, y)] \tag{4}$$

Proof by induction on $\mathcal{V}_t$:

Basis: Let $i = k$, from the vertex m_k only edges to $\mathcal{V}_n$ exists, therefore the minimum distance to all other vertices equals $w(m_k, y)$ which is satisfied in (4): $\mathcal{D}(m_k, y) = \min[\infty, w(m_k, y)]$.

Induction step: Assume $\mathcal{D}(m_{i+1}, y)$ is the minimum distance between m_{i+1} and y. Note that there are two possible ways to go from m_i to y. From all the outgoing edges in m_i, there is one going to m_{i+1}, all others go to a vertex in $\mathcal{V}_n$. Since the outgoing degree of the edges of the vertices in $\mathcal{V}_n$ equals 0, there are only two edges from m_i that may lead to y. That is either directly with cost $w(m_i, y)$, or via m_{i+1} with cost $\mathcal{D}(m_{i+1}, y) + w(m_i, m_{i+1})$, from these two possibilities we have to take the minimum, which is exactly equation 4. □

We must take care that the properties of the graph $\mathcal{G}$ remain valid during the forward trip of the agent. There are cases that need special attention:

- Do not go to an already visited node. This is fulfilled by how the agent moves in the network. If the agent has no other choice,

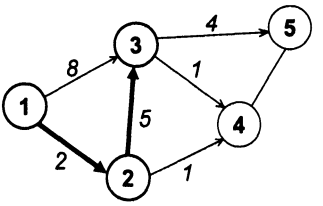

Figure 3 Forward agent arrives at node 3.

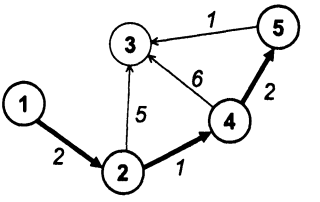

Figure 4 Forward agent arrives at destination node 5.

i.e., all other nodes already have been visited, the agent does not continue on its forward path. It will start with its backward trip to update the routing tables.

- Do not include delay info to an already visited node. This is a small restriction that could loose some information. Let m_a denote the node that we visited in the past and is a neighbor of our current node m_b. Including the delay info $m_b \rightarrow m_a$ would bring no information to nodes we visited: (1) before m_a and, (2) nodes we will visit after m_b, but it could for nodes between m_a and m_b.

- If the agent enters a node that was previously a neighbor node, we have to distinguish between two cases:

  1 Imagine the case depicted in Figure 3; the agent arrives at node 3, of which we have delay information from node 1. In this case, the total delay of going from 1 to 3 via 2 is less than going directly from 1 to 3, and therefore we *delete* the direct delay info from 1 to 3.

  2 When the direct link is shorter, we *change* the visited path by including the shorter link in the visited path, and deleting the delays that are no longer part of the visited path.

  Let's assume that after the agent arrived in node 3 in Fig. 3, the agent continues to node 4. Node 4 is an again a neighbor of node 2, but this time the direct link is shorter. Node 3 is deleted from the visited list. If the agent's next hop is to node 5, then we will have the graph as shown in Fig. 4.

As an example, the minimum distance from node 4 to 3 in Fig. 4 is the direct link $w(4,3) = 6$ or via node 5. The cost of going via node 5 equals $2 + \mathcal{D}(5,4) = 2 + 1 = 3$. So, the minimum distance from node 4 to 3 equals 3.

The algorithm to compute the shortest path in a node consist of a single loop over all destination nodes (thus, complexity $O(N)$) and selecting the minimum distance according to equation 4. Note that the shortest path information calculated in one node is used by other nodes on the return path.

3. THE SIMULATION MODEL

In this section we will discuss the simulation model of the nodes in our network (Section 3.1) and the model of the traffic generator (Section 3.2). We implemented these models in a (C++) simulation tool we wrote ourselves.

3.1. NODE MODEL

All incoming packets are directed to the router (Fig. 5), which has a deterministic service time per packet. We choose this to be sufficient small compared to the speed of the incoming packets. The function of the router is to inspect the destination address in the packet header and perform a routing table lookup. Whenever an agent arrives and it has to perform certain functions in the node, the router forwards the agent to the processor. It may send the agent to another node by setting the destination and forward the agent to the node's router. We found out that the processor queue-server system has a very small influence on the total network performance, because: a) the number of concurrent agents in one node is very small and, b) the algorithm is fast (Section 2.1). Compared to the propagation delays on the links, the processor time is negligible. Nevertheless, we used a deterministic service time of 1 msec. per agent.

If the router has to forward the packet (or agent), packet transmission is started immediately on the output, or is queued if the output is not idle. All output queue space is dynamically shared among all outputs.

3.2. TRAFFIC MODEL

It is important that the generated traffic in our simulation represents real traffic closely, because the agents that will be produced are best suited to deal with the traffic that was used in the simulations. What we use are $N(N-1)$ independent traffic sources (one for each source - destination pair) that can generate Poisson as well as fractal traffic. Ignoring the long-range dependence typically results in overly optimistic performance predictions and inadequately network resource allocation [9, 10]. The method used is called "Random Midpoint Displacement"

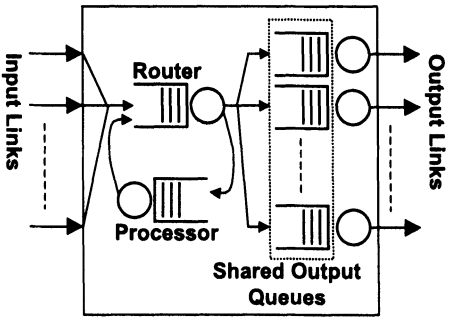

Figure 5 Network node model

(RMD), which is fast, simple, efficient and adequate for qualitative studies.

4. THE EVALUATION METHOD

The main problem of our system is how to update the routing table with the information the agents collect. We want to find a formula for this non-trivial problem by using genetic programming techniques.

The process starts with completely random expressions, for each we run a simulation and we evaluate it using a fitness formula:

$$score = c * \frac{B_{success}}{B_{offered}} + (1 - c) * \frac{max(d_{lim} - d, 0)}{d_{lim}} \tag{5}$$

Which yields a number between 0 (bad) and 1 (good). $B_{success}$ denotes the total number of bytes that were successfully delivered, $B_{offered}$ is the total number of bytes that were offered to the system. With parameter c we can control the importance of throughput versus packet delay, we used $c = 0.9$. A delay d greater than d_{lim} is set to d_{lim}, we used 10 seconds for this limit. We now use Darwinian principles [6] like natural selection, mutation and crossover to build new expressions which we also evaluate, whenever this gives us a better expression, the better one replaces a less good one.

The process continues until we reach a converged state (i.e., no progress during the last x tries). The result is a set of expressions, which are best suited in solving the problem from all expressions we tried. However, we only looked at one specific topology and traffic characteristic so far. To see if the expression we found is more general, we evaluate the expression in about 200 different network configurations. For this purpose we use a

set of powerful computers, including a supercomputer. The results are stored in a database.

The expressions may contain constants, variables and operators. The simulation tool represents expressions by a tree structure, which allows easy manipulation (e.g. mutation and crossover) of the expressions while preserving a correct syntax. The variables that may be used by the genetic programming process are listed below. Assume the agent is on its backward path and enters a node, then the following information is now available:

- d_{no} - The new delay info delivered by the agent to go from the current node to node n using output o.
- $\overline{d_n}$ - The mean delay to go from the current node to node n.
- σ_n - The standard deviation of the collected delays from the current node to node n.
- $P(n, o)$ - The current routing table probability to use output o for packets with destination node n.
- h_{no} - The number of hops the last agent visited on the way to node n using output o.
- tq - A number between 0 and 1 indicating the ratio between the total used queue space and the total available queue space in the current node.
- oq_o - A number between 0 and 1 indicating the ratio between the used queue space by output o and the total used queue space in the current node.
- B_n - The best delay seen by the last w agents to go from the current node to node n.

4.1. OTHER ROUTING PROTOCOLS

Similar like in the evaluation of the AntNet system, we compare the agent based routing system with other routing methods. We implemented in our simulation tool the following routing protocols:[1]

- OSPF: At the simulation start the routing tables in the node are filled according to a shortest path algorithm with a cost function based on the link capacity.

[1]Note that we only implemented the functionality of the routing protocols that is relevant to this study.

- OSPF with load balancing: If multiple paths with equal costs to a destination exist, OSPF may use load balancing to spread the load over multiple paths.

- Daemon: This is the same routing protocol as used in AntNet. It is a routing protocol that knows at any time and without a delay the complete state of the network. Based on this information, a shortest path for each packet is calculated at each hop. The main purpose of this routing protocol is to obtain an empirical upper bound of the achievable performance.

- Daemon Source Routing: Under certain circumstances, OSPF and the agent routing achieve a better performance than the daemon discussed above. Packet delays can be higher, which was caused by oscillating packets. To resolve this we implemented another daemon, which uses source routing. At the generation of the packet a shortest path is calculated to the destination, and this path remains fixed for this packet.

5. RESULTS

This section presents the best agent we found with the genetic programming method (Section 5.1), and discusses the performance of that agent for two scenarios (Section 5.2).

5.1. THE BEST AGENT

Recall that all agents we evaluated are the same, except for their inforcement rule. In this section we present the agent that has the best overall characteristics. We evaluated the agent with all kinds of topologies, network sizes (up to 500 nodes), link capacities, traffic load, etc.

$$r = \begin{cases} \max\left[0.0068, P(n,o)^{\left(\frac{0.01 * {d_{do}}^2}{{B_d}^2}\right)^{0.98}}\right] & \text{for } tq \leq 0.9 \\ \frac{0.76}{2.02h\exp(h*oq_k)-0.95} & \text{for } tq > 0.9 \end{cases} \quad (6)$$

The agent consists of two parts; the one that is taken depends on the node's buffer fill ratio. Let us first consider the first part, i.e., buffer fill ratio is less than 90%. The term $\frac{{d_{do}}^2}{{B_d}^2}$ is a relative measure of how good the trip was. The fact that it is relative is important because otherwise the agent would not perform well in different scenarios, i.e., it will most likely not work equally well in small and large networks.

Furthermore, the formula uses the current routing table probability. For probabilities close to 1, r will be close to 1, so the new probability will remain close to 1, even if the new trip delay time d_{do} is relatively bad. If the probability is close to 0, r will strongly depend on the term $\frac{{d_{do}}^2}{{B_d}^2}$. Assume that $P(k,j) = 1$ for a certain output j and destination k, so $P(k,x) = 0$ for $x \neq j$. Without the "max" function in (6) the probabilities $P(k,\cdot)$ would never change again, the small constant 0.0068 is used to prevent this. This is a way to prevent oscillations in the network, because it takes at least one other agent (who takes the same low probability route) to make a significant routing table probability change.

The first part of (6) balances the network load, so if the traffic increases until congestion arises, it is likely that the queues of multiple nodes will quickly be more than 90% filled, and for these nodes, the second part of (6) will be used by the agents. The inforcement r will be higher for routes that have a low hop count. This has a positive impact on the throughput, because the result is that the agent now tries to use routes with a minimal number of hops and therefore also a minimum chance of being discarded by a full buffer. The second variable is oq_k (which has a smaller influence on r than the hop count) it encourages to use a small local output queue. Recall that the output buffer space is dynamically shared by all outputs.

5.2. SIMULATION RESULTS

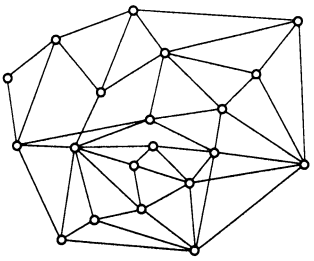

Figure 6 Random topology network

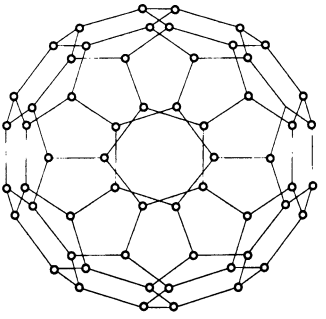

Figure 7 Bucky ball topology

We show the performance of the agent for two different networks: a randomly generated network (Fig. 6) and a network with a regular structure, the so-called Bucky ball (Fig. 7). We chose the Bucky ball because we expected this topology to be relative hard for our agent

system. There might be a high number of more or less equal cost paths, which has a negative influence on the convergence time.

Table 1 Network properties

	Bucky Ball	*Random*
Topology	Regular (Carbon 60 molecule)	Irregular (random)
Nodes	60	20
Bidirectional Links	90	44
Link capacity	5 and 10 Mbit/sec uniform distributed	0.25, 0.5, 0.75 and 1 Mbit/sec uniform distributed
Propagation delay	25 msec.	U(5,10) msec.
Source rate	U(0,Max) packets/sec	U(0.1Max,Max) packets/sec
Source type	Fractal (H=0.7)	Poisson
Packet Size	U(100,1500) bytes	U(100,1500) bytes

The properties of the two simulated networks are shown in table 1.[2][3]
Both simulations use an output queue space of 1 Mbyte per node and the size of the observation window w (used for B_n) equals 200. All simulation runs simulated 1000 seconds, with the routing tables initially set according to the shortest paths.

OSPF using load balancing did not show a significant improvement on OSPF without load balancing in our networks, the same holds for both daemons, the daemon with source routing performed equally well, or slightly better than the other daemon. That is why OSPF with load balancing and the normal daemon are not shown in the graphs in this section.

Bucky ball: Figure 8 shows the bucky ball network throughput for different network loads. All routing methods achieve the same throughput for low and moderate load. OSPF is the first routing method that starts dropping packets, followed by AntNet. Our agent system (QIA) that uses: high priority agents, information from neighboring nodes and an inforcement rule obtained with genetic programming shows a significant improvement on AntNet.

[2]The parameter *Max* in the table is varied in the simulation to get the different traffic load scenarios.
[3]With the notation $U(a, b)$ we mean a number uniformly distributed on the interval $[a, b]$.

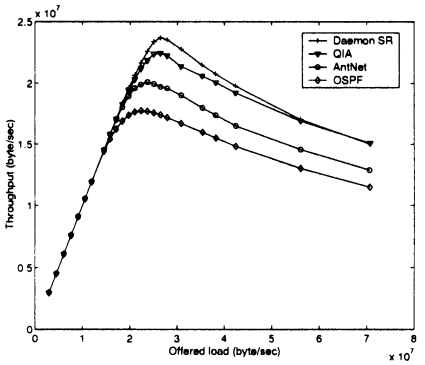

Figure 8 Bucky ball throughput

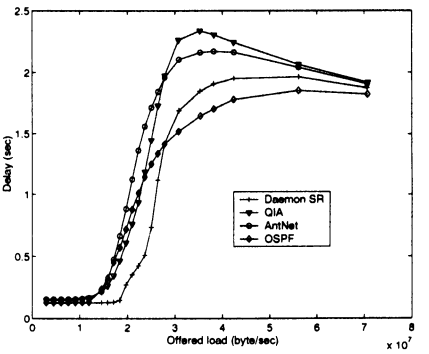

Figure 9 Bucky ball packet delay

From Figure 9 we see, as expected, that the daemon has the lowest delay until congestion arises. For higher loads, OSPF has the lowest delay, but we saw in Figure 8 that OSPF has at the same time the lowest throughput for high loads. The QIA system has a lower mean delay than AntNet until the network becomes congested.

Random network: The throughput graph for the random network (Fig. 10) is similar to the throughput graph of the bucky ball. Except for very high loads, QIA lies between AntNet and the daemon. The delay graph (Fig. 11) shows the excellent performance of QIA: for network loads below 2Mbyte/sec the curve is very close to the daemon, at a load of 1.62 Mbyte/sec AntNet has a mean delay three times higher than QIA. For loads above 2Mbyte/sec there is no significant difference between AntNet and QIA. OSPF is not shown for better readability, but performed bad due to the fact that the network is not well balanced and as a result some links were extremely overloaded.

To give an example of the differences in convergence time between AntNet and QIA, we ran the following experiment. We used the same random network as before, but initialized the routing tables in such a way that each link had the same probability of being used (i.e., random routing). We started the simulation for each of the following routing methods, with a relative low offered load (1.37 Mbyte/sec):

1 AntNet

2 AntNet Modified: We modified AntNet at two points: the agents now also use high priority on their forward path, and inspect the

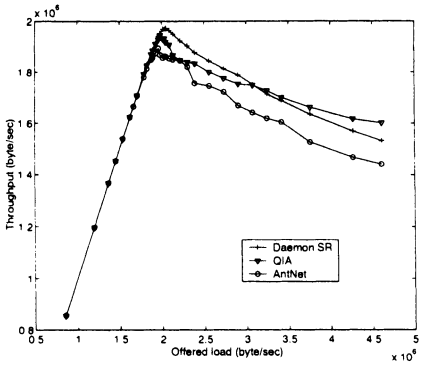

Figure 10 Random network throughput

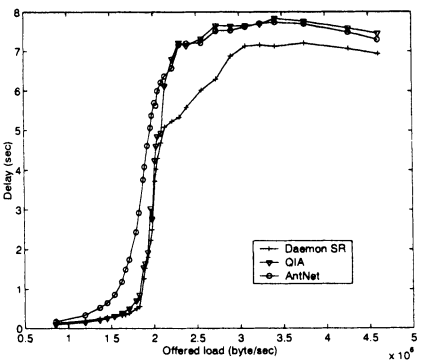

Figure 11 Random network packet delay

neighboring queues as well. Except for the inforcement rule, the modified AntNet system equals QIA.

3 QIA: Our system with high priority agents, using the neighbor queue information and the inforcement rule we found using genetic programming.

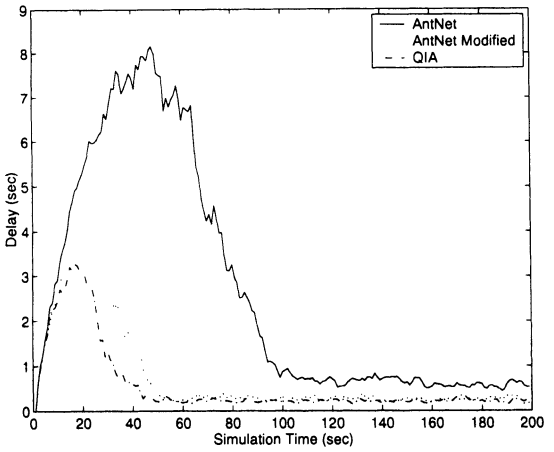

Figure 12 Effect on the convergence time.

Figure 12 shows the results of the experiment; AntNet takes about 100 seconds to reach a stable state, applying the modifications to AntNet reduces this to only 50 seconds. As a direct result, the maximum mean

packet delay is about 50% smaller. The modifications make AntNet adapt faster, without destabilizing the system. The QIA system displays an even better performance.

6. CONCLUSIONS

This paper introduced new techniques for agent routing systems, and made a comparison with other routing methods. We have shown that genetic programming can be successfully applied to the non-trivial problem of agent based routing. Furthermore, we conclude that our other improvements on the AntNet system, which were: a) high agent priority, and b) using neighbor queue information, make it much more reactive on changes in the traffic load without destabilizing the system.

References

[1] G. Di Caro and M. Dorigo, "AntNet: Distributed Stigmergetic Control for Communications Networks", *Journal of Artificial Intelligence Research (JAIR)*, Vol. 9, pp. 317-365, 1998.

[2] T. Magedanz, K. Rothermel and S. Krause, "Intelligent Agents: An emerging technology for next generation telecommunications?", *INFOCOM 1996*, Vol. 2, pp. 464-472, 1996.

[3] S. Appleby and S. Steward, "Mobile software agents for control in telecommunications networks", *British Telecom Technol. Journal 12*, pp. 104-113, 1994.

[4] S. Goss, S. Aron, J.L. Deneubourg and J.M. Pasteels, "Self-organized shortcuts in the Argentine ant", *Naturwissenschaften*, 76, pp. 579-581, 1989.

[5] R. Beckers, J.L. Deneubourg and S. Goss, "Trails and U-turns in the selection of the shortest path by the ant Lasius Niger", *Journal of Theoretical Biology*, 159, pp. 397-415, 1992.

[6] J.R. Koza, F.H. Bennett III, D. Andre and M.A. Keane, "Genetic Programming: Biologically Inspired Computation that Creatively Solves Non-Trivial Problems", *Proceedings of DIMACS Workshop on Evolution as Computation*, 1999.

[7] R. Schoonderwoerd, O. Holland, J.Bruten and L.Rothkrantz, "Ant-based load balancing in telecommunication networks", *Adaptive Behaviour*, Vol. 5:2, pp. 169-207, 1996.

[8] M. Heusse, D. Snyers, S. Gurin and P. Kuntz, "Adaptive Agent-Driven Routing and Load Balancing in Communication Networks", *Ants'98*, Brussels, Belgium, October 1998.

[9] W. Lau, A. Erramilli, J.L. Wang and W. Willinger, "Self-Similar traffic Generation: The Random Midpoint Displacement Algorithm and its Properties", *ICC 1995*, Vol. 1, pp. 466-472, 1995.

[10] V.Paxson and S. Floyd, "Wide-Area Traffic: The failure of Poisson Modeling", *Proc. ACM Sigcomm*, London, UK, pp. 257-268, 1994.