To appear in *Proceedings of 31st Hawaii International Conference on Systems Sciences (HICSS-31) - The Software Technology Track, The Big Island of Hawaii, January 6-9, 1998.*

Mobile Agents for Adaptive Routing

Gianni Di Caro and Marco Dorigo
IRIDIA, Université Libre de Bruxelles
50, av. F. Roosevelt, CP 194/6
1050 - Brussels, Belgium
email: {gdicaro,mdorigo}@iridia.ulb.ac.be

Abstract

This paper introduces AntNet, a new routing algorithm for telecommunication networks. AntNet is an adaptive, distributed, mobile-agents-based algorithm which was inspired by recent work on the ant colony metaphor. We apply AntNet in a datagram network and compare it with both static and adaptive state-of-the-art routing algorithms. We ran experiments for various paradigmatic temporal and spatial traffic distributions. AntNet showed both very good performances and robustness under all the experimental conditions with respect to its competitors.

1 Introduction

We consider the problem of routing in communications networks. Routing refers to the activity of building forwarding tables, one for each node in the network, which tell incoming data which link to use to continue their travel towards the destination node.

Routing, together with congestion and admission control algorithms, plays a critical role in communication networks determining the overall network performance in terms of throughput and transmission delays.

In this work we focus on datagram-like networks with irregular topology, the most remarkable example of such networks being the Internet, and without congestion or admission control components.

The routing algorithm that we propose in this paper was inspired by previous works on ant colonies and, more generally, by the notion of *stigmergy*, introduced by Grassé [7] to describe the indirect communication taking place among individuals through modifications induced in their environment. Real ants have been shown to be able to find shortest paths using only the pheromone trail deposited by other ants [1].

Algorithms which take inspiration from ants behavior in finding shortest paths have recently been successfully applied to both combinatorial optimization [6, 5] and circuit switched communications network problems [13]. In ant colony based algorithms a set of artificial ants move on the graph which represents the instance of the problem: while moving they build solutions and modify the problem representation by adding collected information. In *AntNet*, the algorithm we propose in this paper, each artificial ant builds a path from its source to its destination node. While building the path, it collects explicit information about the time length of the path components and implicit information about the load status of the network. This information is then back-propagated by another ant moving in the opposite direction and is used to modify the routing tables of visited nodes.

We report on the behavior of *AntNet* as compared to some effective static and adaptive vector-distance and link-state shortest paths routing algorithms [14]. *AntNet* shows the best performances and the more stable behavior for all the paradigmatical temporal and spatial traffic distributions considered. Absolute performances are scored according to a scale defined by an ideal algorithm giving an empirical bound. Competing algorithms performed poorly for heavy traffic conditions and showed more sensitivity to internal parameters tuning.

2 Routing Algorithms: an Overview

The goal of every routing algorithm is to direct traffic from sources to destinations maximizing network performance while minimizing costs. In this way, the general problem of determining an optimal routing algorithm can be stated as a multiobjective optimization problem in a non-stationary stochastic environment. Additional constraints are posed by the underlying network switching and transmission technology.

The performance measures that usually are taken into account are *throughput* and *average packet delay.* The former quantify the quantity of service that the network has been able to offer in a certain amount of time, while the latter defines the quality of service produced at the same time.

Routing algorithms can be at first broadly classified as *static* or *adaptive*. In static (or *oblivious*) routers the path taken by a packet is determined only on the basis of the source and destination, without regard to the current network state. This path is usually chosen as the shortest one according to some cost criterion. Adaptive routers are, in principle, more attractive, because they try to adapt the routing policy to the varying traffic conditions. As a drawback, they can cause oscillations in selected paths. This can generate circular routes, as well as large fluctuations in performances, specially for what concerns average delays [2].

The most widely used routing algorithms (at least considering only wide-area networks) are *shortest paths* algorithms. Shortest path routing has a source-destination pair perspective: there is no a global cost function to optimize. Its objective is to determine the shortest path between two nodes, where the link costs are computed (statically or adaptively) following some statistical description of the link states.

The novel method we introduce in this paper in sect. 4, shares the same optimization perspectives as shortest path algorithms but not their usual implementation paradigms (depicted in Appendix).

3 The Communication Network Model

We focus our experiments on datagram networks with irregular topology without mechanisms for congestion and admission control. These mechanisms can influence greatly the network performances [4], but, as a first step, we wanted to check the behavior of our algorithm and of its competitors in conditions which minimize the number of critical interacting components.

In our model the instance of the communication network is mapped on a directed weighted graph with N nodes. All the links are viewed as bit pipes characterized by a bandwidth (bits/sec) and a transmission delay, and are accessed following a statistical multiplexing scheme. For this purpose, every node, of type store-and-forward (i.e., switch element), holds a buffer space where the incoming and the outcoming packets are stored. This buffer is a shared resource among all the queues attached to every incoming and outgoing link of the node. Traveling packets can be data or routing packets. Packets of the same type have the same priority, so they are queued and served only on the basis of a first-in-first-out policy, with routing packets having greater priority than data packets.

A packet reads from the routing table the information about which link to use to follow its path toward its target node. When link resources are available, they are reserved and the transfer is set up. The time it takes to a packet to move from one node to a neighboring one depends on its size and on the link transmission characteristics. If on packet's arrival there is not enough buffer space to hold it, the packet is discarded. No arrival acknowledgment or error notification packets are generated back to the source.

Situations causing a temporary or steady alteration of the network topology are not taken into account.

4 AntNet: an Adaptive Agent-based Routing Algorithm

As emphasized before, the routing problem is a stochastic distributed multiobjective problem. Information propagation delays and the difficulty to model the network dynamics under arbitrary traffic patterns, make the general routing problem intrinsically distributed. Routing decisions can only be made on the basis of local and approximate information about the current and the future network states.

These features make the problem well adapted to be solved following a multiagent approach like our *AntNet* system, composed by two sets of *homogeneous mobile agents* (see [15] for an agents taxonomy), called in the following respectively *forward* and *backward* ants.

Agents[1] in each set possess the same structure, but they are differently situated in the environment; that is, they can sense different inputs and they can produce different, independent outputs. Agents behave *reactively* retrieving a pre-compiled set of simple behaviors to select the route and to modify the routing tables, but at the same time they maintain a complete internal state description.

The *AntNet* algorithm can be informally described as follows.

1. At regular intervals, from every network node s, a mobile agent (that we will call *forward ant*) $F_{s\rightarrow d}$, is launched, with a randomly selected destination node d. The identifier of every visited node k and the time elapsed since its launching time to arrive at this $k-$th node, are pushed onto a memory stack $S_{s\rightarrow d}(k)$ and inserted in a dictionary structure $D_{s\rightarrow d}$, carried by the agent.
2. Each traveling agent selects the next hop node using the information stored in the routing table. The route is selected, following a random scheme, proportionally to the goodness (probability) of each neighbors node, or, with a tiny probability (exploration probability), assigning the same

[1] In the following will use interchangeably the terms Ant and Agent

selection probability to each of the neighbor nodes. If, in the proportional case, the chosen node has already been visited, a uniformly random selection among the neighbors is applied.

3. If a cycle is detected, that is, if an ant is forced to return in an already visited node, the cycle's nodes are popped from the ant's stack and all the memory about them destroyed.
4. When the destination node d is reached, the agent $F_{s \to d}$ generates another agent (*backward ant*) $B_{d \to s}$, transferring to it all its memory.
5. The backward ant makes the same path as that of its corresponding forward ant, but in the opposite direction. At each node k along the path it pops its stack $S_{s \to d}(k)$ to know the next hop node.
6. Arriving in a node k coming from a neighbor node f, the backward ant updates the following two data structures maintained by every node:
   i) a routing table, organized as in vector-distance algorithms; in the table, a probability value P_{in} which expresses the goodness of choosing n as next node when the destination node is i, is stored for each pair (i, n) with the constraint
   $$\sum_{n \in \mathcal{N}_k} P_{in} = 1, i \in [1, N], \ \mathcal{N}_k = \{neighbors(k)\};$$
   ii) a list $Trip_k(\mu_i, {\sigma_i}^2)$ of estimates of arithmetic mean values μ_i and associated variances ${\sigma_i}^2$ for trip times from itself to all the nodes i in the network (for agent-sized packets). This data structure represent a memory of the network state as seen by node k.

   These two data structures are updated as follows:
   i) the list $Trip_k$ is updated with the values stored in the stack memory $S_{s \to d}(k)$; all the times elapsed to arrive in every node $k' \in S_{k \to d}$ starting from the current node k are used to update the corresponding sample means and variances $Trip_k(\mu_{k'}, {\sigma_{k'}}^2)$;
   ii) the routing table is changed incrementing the probability P_{df} associated with node f when the destination is node d and decrementing the probability P_{dn} associated with the other nodes n in the neighborhood for the same destination.

The update of the routing table happens using the only available feedback signal, that is, the trip time experienced by the forward ant. This time gives a clear indication about the goodness of the followed route because it is proportional to its physical length (hops, bandwidth and delay of the used links, processing speed of the crossed nodes) and to the traffic congestion. This last aspect is extremely important: forward agents share the same queues as data packets (backward agents do not, they have priority over data to faster propagate the accumulated information), so if they cross a congested area, they will be delayed for a long time. This has a double effect: (i) the trip time will grow and then back-propagated probability increments will be small and (ii) at the same time these increments will be assigned with a bigger delay.

We used the time measure as a reinforcement signal to provide structural and temporal credit assignment. The credit assignment problem we have to face with is the typical one arising in reinforcement learning field [3]: we cannot associate to the realized performance (trip time) an exact error measure. "Optimal" times depend on traffic and/or components failure states, and they have to be considered from a network wide point of view. We can only give an "advice" about the goodness of the observed trip time on the basis of the estimated mean values for the agent's trip times, stored in the list $Trip_k$.

In light of these considerations we can detail the procedure followed to update the routing tables (we will omit indices when they are not necessary).

If T is the observed trip time and μ is its mean value, as stored in the list $Trip$, we compute a raw quantity r' measuring the goodness of T, with small values of r' corresponding to satisfactory trip times,

$$r' = \begin{cases} \dfrac{T}{c\mu}, \quad c \geq 1 \quad if \quad \dfrac{T}{c\mu} < 1 \\ \\ 1 \qquad\qquad\qquad otherwise. \end{cases}$$

r' is an adimensional measure, problem independent, scoring how good is the elapsed trip time with respect to what has been on average observed until now. μ plays the role of a unit of measure and c is a scale factor (we found that setting c to 2 is a reasonable choice). "Out-of-scale" values are saturated to 1.

A correction strategy is applied to the goodness measure r' taking into account how reliable is the currently observed trip time with respect to the variance in the so far sampled values, that is, considering how stable is the trip time mean value. We say that the observations in the mean are stable if $\sigma/\mu < \epsilon, \ \epsilon \ll 1$

In this case, a good trip time (i.e., r' less than a threshold value t that we set to 0.5) is decreased by subtracting a value

$$S(\sigma, \mu; a) = e^{-a \frac{\sigma}{\mu}}$$

to the value of r', while a poor trip time is increased adding the same quantity.

On the other side, if the mean is not stable, the raw values r' cannot be completely considered reliable and, in this case, the quantity

$$U(\sigma,\mu;a') = \frac{e^{-a'\dfrac{\sigma}{\mu}}}{e^{a'}}, \qquad \frac{\sigma}{\mu} \in [\epsilon, 1],$$

with $a' \leq a$, is added to a good r' value and subtracted from a poor one. In this case we try to avoid following the traffic fluctuations, with the risk of amplifying them: adding and subtracting the value U helps to stabilize them.

The above correction strategy, for both cases of σ/μ values, can be summarized as:

$$r' \leftarrow r' + sign(t - r')sign(\frac{\sigma}{\mu} - \epsilon)f(\sigma,\mu) \ ,$$

with f being S or U according to the case. The f functions have been chosen as decreasing/increasing exponential because both the function and its first derivative are monotonically decreasing/increasing with increasing values of the σ/μ ratio. The obtained value of r' is finally reported on a more compressed scale through a power law, $r' \leftarrow (r')^h$ (see below for an explanation), and bounded in the interval $[0, 1]$.

These transformations from the raw value T to the more refined value r' play the role of a local estimation of a traffic model. More sophisticated and computationally-demanding models could be learnt to compute a more effective traffic-dependent correction.

The obtained value r' is used by the current node k to define a positive reinforcement, r_+, for the node f the backward ant comes from, and a negative one, r_-, for the other neighboring nodes n:

$$\begin{aligned} r_+ &= (1 - r')(1 - P_{df}) \\ r_- &= -(1 - r')P_{dn}, \quad n \in \mathcal{N}_k,\ n \neq f, \end{aligned}$$

where P_{df} and P_{dn} are the last probability values assigned to neighbors of node k for destination d. In this way, the reinforcements are proportional to the obtained goodness measure r' and to the previous value of node probabilities.

These probabilities are then increased/decreased by the reinforcement values as follows (their sum will still be 1, being $r' \in [0, 1]$):

$$P_{df} \leftarrow P_{df} + r_+, \quad P_{dn} \leftarrow P_{dn} + r_- \ .$$

It is now clear that the power law rescaling of the r' value is equivalent to the definition of a learning rate: the scale compression factor and its degree of non linearity determine the finale size of the allowed jumps in the probability values.

The constants (c, a, a', ϵ, h, t) used in this section are not problem-dependent and they simply define an appropriate scaling system for the computed values. They have been set to the following values: $c = 2, a = 10, a' = 9, \epsilon = 0.25, h = 0.04, t = 0.5$.

As a last consideration, notice the critical role played by the used paradigm for agents communication. In fact, each agent is enough complex to solve a single sub-problem but the global routing optimization problem cannot be solved efficiently by a single agent. It is the interaction between the agents that determines the emergence of a global effective behavior from the network performances point of view. The key concept in the cooperative aspect lies in the way communication among agents happens. The intrinsically distributed nature of the problem makes natural and convenient using a *blackboard* type of inter-agents communication, that is, an indirect communication from one agent to all the others mediated by the environment. The information locally stored and updated at each network node defines the agent input state. Each agent uses it to realize the next node transition and, at the same time, it will modify it, modifying in this way the local state of the node as seen by future agents. This specific form of indirect communication through the environment with no explicit level of agents coordination is called *stigmergy* [7, 13, 15]. *Active stigmergy* occurs when an agent alters the environment so as to affect the input of another agent, *passive stigmergy* occurs when an agent alters the environment in such a way that the effect of the actions of the other agents is no more the same. In our case we used active stigmergy as a way of transmitting the information associated with every "experiment" made by each agent (we could see our system as a particular instance of an iterated Monte Carlo simulation).

5 Routing Algorithms Used for Comparison

To evaluate the performances of *AntNet*, we selected a set of competitors algorithms from the shortest path class (see Appendix) reflecting Internet standards and state-of-the-art for routing algorithms.

OSPF: is our implementation of the official Internet routing algorithm [11]. The Internet *OSPF* has a lot of features for the full network management. Here we are only interested in data packet routing in simplified conditions, therefore, the original algorithm is reduced to the definition of routing tables by static shortest paths calculation.

BF: is an implementation of the asynchronous distributed Bellman-Ford algorithm with dynamic link metrics [2]. Vector-distance Bellman-Ford-like algorithms are today in use mainly for intra-domain routing, being used in the Routing Information Protocol (RIP) [9] supplied with the BSD version of Unix.

SPF: is the prototype of link-state algorithms with dynamic metric for link costs evaluations. A similar algorithm was implemented in the second version of ARPANET [10] and in its successive revisions [8]. We implemented it with state-of-the-art link costs evaluation and flooding algorithms.

SPF_1F: is the same as *SPF* but with flooding limited to the first neighbors. As far as we know, this is the first time that a similar algorithm is presented in literature. It has the nice features that the shortest paths are computed on the basis of locally updated information and the costs of far links are all set to the same value.

Daemon: is an ideal algorithm. It defines an empirical bound on the achievable performances in the absence of any *a priori* assumption on traffic statistics. The algorithm posses a "daemon" able to read in every instant the state of all the queues in the network and then it can calculate instantaneous "real" costs for all the links. With this information paths are assigned on the basis of a network wide shortest paths re-calculation for every packet hop. Links costs are assigned in the following way:

$$C_{l_i} = d_{l_i} + \frac{S_p}{b_{l_i}} + (1-\alpha)\frac{S_{Q(l_i)}}{b_{l_i}} + \alpha\,\frac{\bar{S}_{Q(l_i)}}{b_{l_i}}, \quad \forall i \in [1, N]$$

where d_{l_i} is the delay for link l_i, b_{l_i} is its bandwidth, S_p is the size of the data packet doing the hop, $S_{Q(l_i)}$ is the length of the queue for link l_i, $\bar{S}_{Q(l_i)}$ is the exponential mean of the length of links queue (it is a correction to the current length of the link queue on the basis of what observed until that moment), the weight α is set to 0.4.

Algorithms *BF*, *SPF* and *SPF_1F* use a dynamic metric for link costs. We tried the following different metrics documented in literature [12].

1. The link cost is measured by the fraction of time of non-empty queue with respect to the empty-queue period, measured over the last time-window.
2. The link cost is set to the sum of the mean packet delay in the link queue over the last time-window plus the transmission delay.
3. The mean of the transmission time over the link, $\bar{T}_l$, and the mean delay in the link's queue, $\bar{D}_{q(l)}$, are computed over the last time-window. The link cost is then assigned to [8]: $1 - \bar{T}_l/(\bar{D}_{q(l)} + \bar{T}_l)$.
4. The link cost is a weighted combination of a pair of the above metrics. We experimentally observed that the best combination was given by a weighted average between metrics 2 and 3.

6 Experimental Settings

We defined a limited set of tunable components and for each of them our choices are explained:

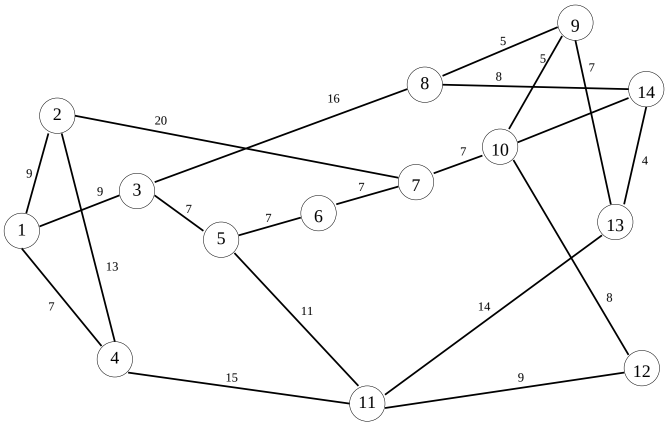

Figure 1: NSFNET. Numbers in circles are node identifiers. Each edge in the graph represents a pair of directed link and the numbers are propagation delays in msec.

Topology and physical properties of the net. In our experiments we used a real net instance, NSFNET, the USA backbone. It is composed of 14 nodes and 21 bidirectional links. The topology and the propagation delays are showed in fig. 1. The bandwidth is 1.5 Mbit/s for every link, the assigned link or node fault probability is null, local buffers have a gigabyte of capacity and links are accessed through statistical multiplexing.

Traffic patterns. Two models, static and dynamic, of temporal traffic patterns have been used, modeling respectively the case of *Constant Bit Rate* (CBR) and of *Variable Bit Rate* (VBR) [12].

- In the *static* model all the sessions start at the beginning of the simulation and they last until the end. In this way we simulate a situation of stationarity. The packets inter-arrival time is of 150 ms and their size distribution is negative exponential with mean 512 bytes.
- In the *dynamic* model, sessions are activated following a negative exponential distribution for the inter-arrival times. The distribution mean value is fixed to 15 sec. The total number of packets per session, their sizes and

their inter-arrival time, are negative exponentially distributed, with respectively mean values of 50, 512 bytes and 10 μsec. In this case sessions are "bursty" and hence data flows are highly irregular.

For the geographical distribution of traffic patterns we considered four significant situations.

- *Uniform-deterministic* distribution (UD): there are $n \geq 1$ open sessions between any pair of nodes.
- *Uniform-random* distribution (UR): there are $n \geq N^2$ ($N=$ total number of nodes) open sessions. Start and end-points are selected uniformly randomly.
- *Uniform-deterministic-hot-spots* distribution (UDHS): two different types of load are concurrently present. One is the same as in the UD case, the other is represented by a set H of end-points nodes, $|H| < N$, which act like hot-spots. Each node has $n \geq 1$ open sessions with an end-point node $h, h \in H$.
- *Uniform-random-hot-spots* distribution (URHS): as for UDHS, two different types of load are concurrently present. One is the same as in the UR case, while the other is the same hot-spots component of UDHS.

Metrics for performances evaluation. We considered only sessions having equal costs, benefits and priorities, In this perspective the measures we are interested in are: *throughput* (delivered bits/sec) and *average packet delay* (sec).

Routing algorithms parameters. The generation interval for *AntNet* is set to 1 (sec), start and end-points are sampled uniformly over the network, the exploration probability is set to 0.002, the agent size is 24 bytes for the forward and $24 + N_h$ bytes, N_h =number of hops, for the backward agent. The ant processing time is 2 ms.
For shortest paths algorithms (see Appendix), the time interval for information broadcasting and the time window to average link costs are the same, and they are set to 8 seconds. For the *BF* algorithm the routing packet has elaboration time of 4 ms and size of $(24 + 12N)$ bytes. For the other algorithms the elaboration time is set to 6 ms and the size to $(18 + 8|\mathcal{N}_k|)$ bytes, where $\mathcal{N}_k$ is the set of neighbors of the broadcasting node k.

7 Experimental Results

We report results relative to the four cases of spatial traffic distribution for each of the two temporal traffic patterns, CBR and VBR. For very low and uniform traffics loads, the six algorithms behave almost in the same way and their performances are near to optimal. Increasing traffic load and/or considering strongly asymmetric spatial distributions, performances become appreciably different from a statistical point of view. We report results only for some representative cases.

The time length of the simulation has been set to 120 seconds. We observed that after this time interval the behavior of the algorithms is already well characterized. A 100 seconds of "learning" time has been assigned to the algorithms to learn routing tables in absence of traffic. Reported values are averaged over 10 trials.

In tables 1-8, observed mean values (and their standard deviations, in brackets) for throughput and mean packet delays are reported. The content of tables 1-4 is relative to the case of VBR temporal distribution considered together with the four spatial distribution cases of sect. 6 (UD, UR, UDHS, URHS). Tables 5-8 follow the same scheme but are relative to the CBR case as temporal traffic distribution. Experimental conditions are explained in detail in tables captions with the following conventions: N_{UD} =active sessions between all the node pairs; N_{UR} =randomly selected sessions in the network, except for hot-spot sessions; N_{HSN} = hot-spot nodes; N_{HSS} =hot-spot sessions. Graphs 2-11 refer to some of the most meaningful cases considered in tables. For each of the considered situations, temporal evolution of the average packet delay is showed for every algorithm. Throughput plots are not showed because of space reasons and because throughput performances present much less striking differences than those for packet delays. From tables it is evident that *BF*, the algorithm based on distributed Bellman-Ford, scores very poorly with respect to the others, therefore, results about it are not plotted.

Reported results show clearly that *AntNet* is the best performing algorithm among the considered ones (except for the ideal algorithm *Daemon*). In some cases its superiority is evident, in others it performs like the best ones within statistical fluctuations. In the CBR case *AntNet* shows very low delays compared to the others, while in the VBR case, the other new algorithm, *SPF_1F*, presents comparable or slightly better performances. Of course, the *Daemon* algorithm has always the best performances, as expected, and comparing its performances with those of *AntNet* we can see that in the half of the cases *AntNet* performances are almost the same within statistical uncertainties, confirming in this way the excellent behavior of our algorithm according to an absolute scale of values.
"Classical" algorithms (*OSPF, SPF* and *BF*) performed poorly with respect to *AntNet* and *SPF_1F* (limited to the VBR case) and their behavior showed

Table 1: Results for *VBR* temporal traffic distribution and *Uniform-deterministic* spatial traffic distribution. $N_{UD} = 5$.

	AntNet	OSPF	SPF	SPF_1F	Daemon	BF
Mean Delay (sec)	0.10 (0.01)	0.78 (0.10)	1.41 (0.12)	0.075 (0.03)	0.036 (0.003)	7.85 (2.5)
Throughput (10^7 bits/sec)	1.348 (0.005)	1.345 (0.007)	1.330 (0.007)	1.347 (0.004)	1.347 (0.005)	0.590 (0.150)

Table 2: Results for *VBR* temporal traffic distribution and *Uniform-random* spatial traffic distribution. $N_{UR} = 840$.

	AntNet	OSPF	SPF	SPF_1F	Daemon	BF
Mean Delay (sec)	0.09 (0.01)	0.17 (0.12)	1.00 (0.52)	0.05 (0.04)	0.033 (0.002)	7.00 (2.51)
Throughput (10^7 bits/sec)	1.244 (0.003)	1.245 (0.003)	1.201 (0.002)	1.244 (0.002)	1.244 (0.003)	0.434 (0.580)

Table 3: Results for *VBR* temporal traffic distribution and *Uniform-deterministic-hot-spots* spatial traffic distribution. $N_{UD} = 5, N_{HSN} = 5$ and $N_{HSS} = 5$.

	AntNet	OSPF	SPF	SPF_1F	Daemon	BF
Mean Delay (sec)	0.86 (0.35)	4.80 (2.50)	2.13 (0.35)	1.05 (0.25)	1.05 (0.07)	4.80 (2.03)
Throughput (10^7 bits/sec)	1.822 (0.008)	1.677 (0.008)	1.784 (0.016)	1.823 (0.002)	1.824 (0.003)	0.870 (0.300)

Table 4: Results for *VBR* temporal traffic distribution and *Uniform-random-hot-spots* spatial traffic distribution. $N_{UR} = 840, N_{HSN} = 5$ and $N_{HSS} = 5$.

	AntNet	OSPF	SPF	SPF_1F	Daemon	BF
Mean Delay (sec)	0.32 (0.12)	3.01 (1.50)	1.94 (0.54)	0.53 (0.18)	0.11 (0.11)	7.48 (2.22)
Throughput (10^7 bits/sec)	1.723 (0.001)	1.652 (0.008)	1.680 (0.002)	1.723 (0.002)	1.723 (0.001)	0.671 (0.117)

Table 5: Results for *CBR* temporal traffic distribution and *Uniform-deterministic* spatial traffic distribution. $N_{UD} = 5$.

	AntNet	OSPF	SPF	SPF_1F	Daemon	BF
Mean Delay (sec)	0.93 (0.20)	5.85(1.43)	3.58 (0.83)	4.96 (1.25)	0.10 (0.03)	4.27 (1.22)
Throughput (10^7 bits/sec)	2.392 (0.011)	2.100 (0.002)	2.284 (0.033)	2.201 (0.004)	2.403 (0.010)	1.410 (0.047)

Table 6: Results for *CBR* temporal traffic distribution and *Uniform-random* spatial traffic distribution. $N_{UR} = 840$.

	AntNet	OSPF	SPF	SPF_1F	Daemon	BF
Mean Delay (sec)	0.79 (0.18)	4.63 (1.03)	2.01 (0.50)	2.36 (0.67)	0.06 (0.01)	3.90 (1.05)
Throughput (10^7 bits/sec)	2.219 (0.011)	2.013 (0.011)	2.171 (0.023)	2.141 (0.008)	2.205 (0.007)	1.280 (0.065)

Table 7: Results for *CBR* temporal traffic distribution and *Uniform-deterministic-hot-spots* spatial traffic distribution. $N_{UD} = 5, N_{HSN} = 5$ and $N_{HSS} = 5$.

	AntNet	OSPF	SPF	SPF_1F	Daemon	BF
Mean Delay (sec)	3.37 (0.60)	12.00 (2.25)	9.60 (1.44)	11.48 (1.52)	3.28 (0.54)	4.19 (1.97)
Throughput (10^7 bits/sec)	3.134 (0.060)	2.128 (0.044)	2.815 (0.047)	2.480 (0.054)	3.140 (0.058)	1.250 (0.131)

Table 8: Results for *CBR* temporal traffic distribution and *Uniform-random-hot-spots* spatial traffic distribution. $N_{UR} = 840, N_{HSN} = 5$ and $N_{HSS} = 5$.

	AntNet	OSPF	SPF	SPF_1F	Daemon	BF
Mean Delay (sec)	3.18 (0.75)	10.30 (1.94)	9.42 (0.85)	9.25 (1.00)	2.83 (0.78)	5.09 (1.28)
Throughput (10^7 bits/sec)	2.986 (0.014)	2.235 (0.009)	2.619 (0.008)	2.350 (0.021)	3.012 (0.008)	1.321 (0.140)

significant fluctuations, both in terms of absolute performances and of stability. *AntNet* resulted in the more stable performances and behavior, that is, always moving rapidly toward a good stable delay value after an initial transitory phase.

8 Discussion and Conclusions

In this paper, we introduced *AntNet*, a new algorithm for adaptive routing. It is a mobile-agents-based distributed algorithm using stigmergy as primitive form of communication among agents. Its behavior with respect to throughput and mean packet delay has been compared to the behavior of five shortest paths routing algorithms. As a testbed we considered heavy traffic conditions for some representative temporal and spatial traffic patterns for a real network instance.

AntNet performed always as the best among its competitors or at the same level within the statistical fluctuations. Differently from the other algorithms,

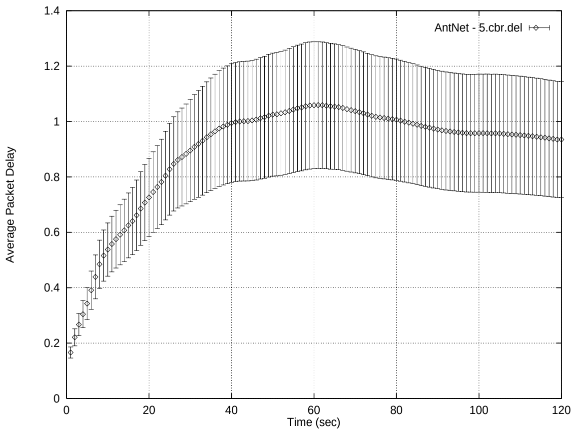

Figure 2: *AntNet*: Average packet delay for *CBR* temporal traffic distribution and *Uniform-deterministic* spatial traffic distribution. $N_{UD} = 5$.

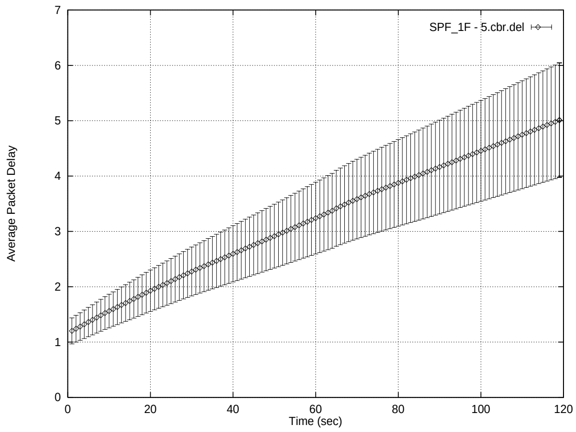

Figure 5: *SPF_1F*: Average packet delay for *CBR* temporal traffic distribution and *Uniform-deterministic* spatial traffic distribution. $N_{UD} = 5$.

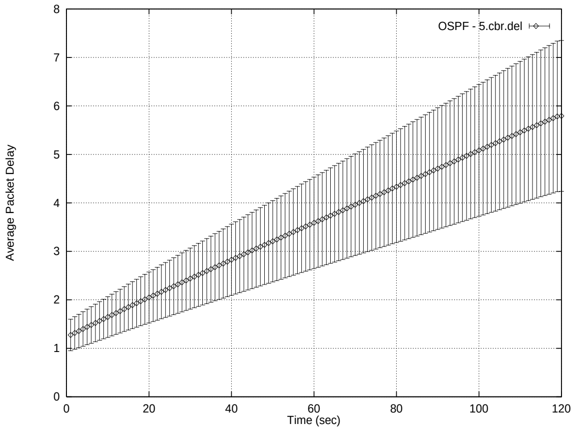

Figure 3: *OSPF*: Average packet delay for *CBR* temporal traffic distribution and *Uniform-deterministic* spatial traffic distribution. $N_{UD} = 5$.

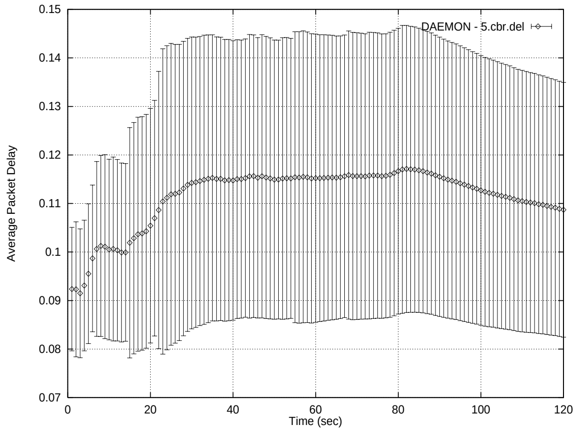

Figure 6: *Daemon*: Average packet delay for *CBR* temporal traffic distribution and *Uniform-deterministic* spatial traffic distribution. $N_{UD} = 5$.

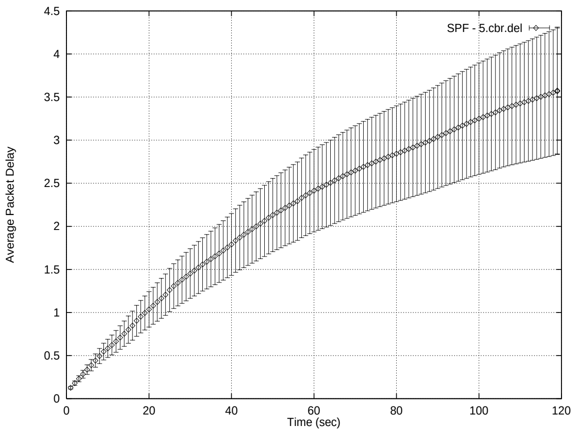

Figure 4: *SPF*: Average packet delay for *CBR* temporal traffic distribution and *Uniform-deterministic* spatial traffic distribution. $N_{UD} = 5$.

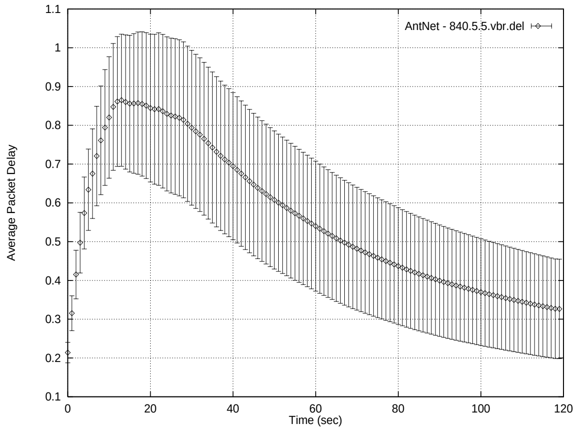

Figure 7: *AntNet*: Average packet delay for *VBR* temporal traffic distribution and *Uniform-random-hot-spots* spatial traffic distribution. N_{UR}=840, N_{HSN}=5, N_{HSS}=5.

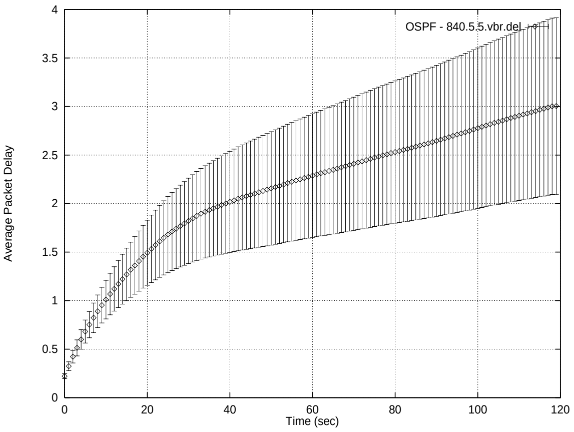

Figure 8: *OSPF*: Average packet delay for *VBR* temporal traffic distribution and *Uniform-random-hot-spots* spatial traffic distribution. N_{UR}=840, N_{HSN}=5, N_{HSS}=5.

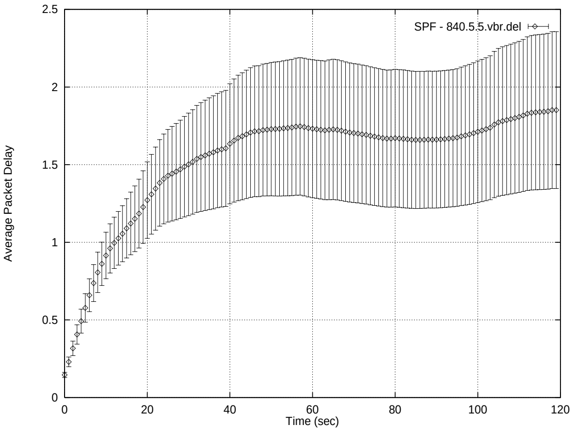

Figure 9: *SPF*: Average packet delay for *VBR* temporal traffic distribution and *Uniform-random-hot-spots* spatial traffic distribution. N_{UR}=840, N_{HSN}=5, N_{HSS}=5.

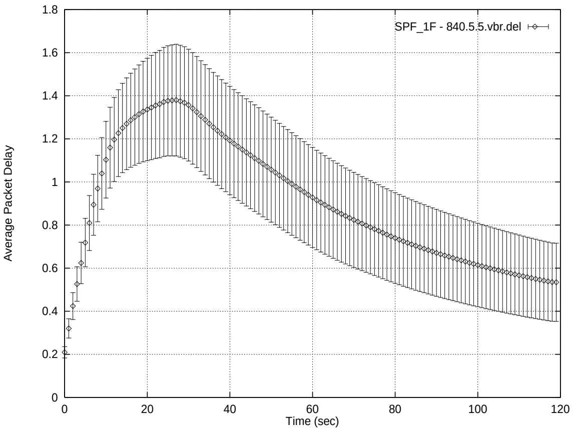

Figure 10: *SPF_1F*: Average packet delay for *VBR* temporal traffic distribution and *Uniform-random-hot-spots* spatial traffic distribution. N_{UR}=840, N_{HSN}=5, N_{HSS}=5.

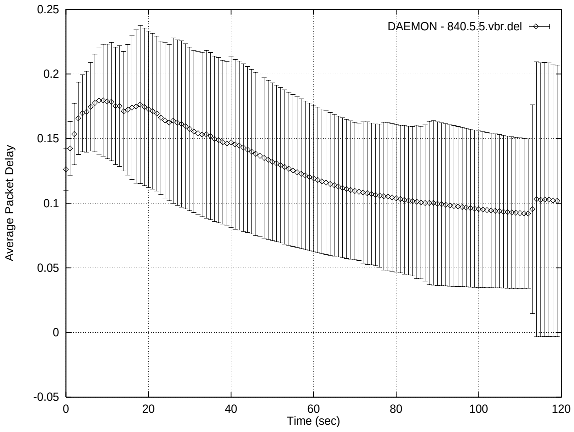

Figure 11: *Daemon*: Average packet delay for *VBR* temporal traffic distribution and *Uniform-random-hot-spots* spatial traffic distribution. N_{UR}=840, N_{HSN}=5, N_{HSS}=5.

AntNet showed always a robust behavior, being able to rapidly reach a good stable level in performances.
Its performances are affected mainly by the frequency with which the agents are launched and by their spatial distribution. We ran several experiments to check the sensitivity of the algorithm to these parameters. With no traffic, we observed that the system converges more quickly to the shortest paths if the launching rate is increased till a threshold value (of order of a few millisec for the considered network). Beyond that, the routing tables oscillate too much or have the tendency to converge too fast, and the overall performance tends to degradate (in case of no traffic the ant system can solve shortest paths instances, although other algorithms, like Bellman-Ford or Dijkstra, can solve them more efficiently). A similar behavior has been observed in presence of a traffic load. If the launching rate is too high, the interaction among the ants creates oscillations and degrades the performances. We observed that for the considered network, varying the launching rate approximatively in the range between 0.5 and 2.0 sec does not affect considerably the performances. Concerning the spatial distribution of the ants, we note that selecting the (source, destination) pairs uniformly over the network makes possible to update uniformly in time all the routing tables and therefore to balance at the best the load over all the network. If we create a bottleneck somewhere in the network the overall performance will suffer because of it, even if other paths are selected in an apparently very optimized way. An important point concerns the "reaction time" of the algorithm: each forward ant makes a single "experiment", and successively the backward ant updates the probabilistic tables (this is equivalent to an iterated parallel Monte Carlo system, with the addition of temporal constraints). This means that, from the moment

of the "implicit observation" of a traffic situation by the forward ant, to the moment the backward ant will use this observation, there is a delay. In the considered cases this delay is of the order of approximately 1 second: this is perfectly compatible with a "high" rate of varying traffic. Anyway, also if the update is no longer consistent with an evolved traffic situation, the update will affect only the route for a single destination. This feature makes the system robust to "wrong estimates", and it is very different from what happens for example in SPF, where a wrong link cost estimate alters all the routes crossing the node. As a last note, it is important to note that the impact of the system on the network resources is neglectable, both in terms of bandwidth and computation, and this is also true for the other considered algorithms.

Acknowledgments

This work was supported by a Madame Curie Fellowship awarded to Gianni Di Caro (CEC-TMR Contract N. ERBFMBICT 961153). Marco Dorigo is a Research Associate with the FNRS.

Appendix: Shortest Path Routing

In relation to the different content stored in each node routing table, shortest path algorithms can be classified as *distance-vector* or *link-state* [14]. They show the following common behavior: each node assigns a static or dynamic cost (assigned on the basis of failure states and of some link-traffic statistics averaged over a time-window) to each of its outgoing links and broadcasts them periodically to its neighbor nodes. These latter use the received information to update their local routing tables.

Distance-vector algorithms maintain a set of triples of the form *(Destination, Estimated Distance, Next Hop)*, defined for all the destinations in the network and for all its neighbor nodes. The information sent to neighbor is the list of its last estimates of the distances from itself to all the other nodes in the network. On receiving this information from a neighbor node j, the receiving node i updates its table of distance estimates in the entry corresponding to the case of node j as next hop node. Iterating this procedure, known as distributed *Bellman-Ford* [2] and based on the principles of dynamic programming, the routing tables will converge to shortest paths tables.

Link-state algorithms maintain in each node a replicated database describing all the topological details of the network components. Using this database as input each node calculates its best paths using an appropriate algorithm for shortest paths calculations. Each node acts autonomously, assigning costs to its connected links, periodically flooding the information about them, and computing the shortest paths from itself to all the destinations on the basis of these local costs and of the received link costs.

References

[1] 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*, 159:397–415, 1992.

[2] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, 1992.

[3] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[4] P. B. Danzig, Z. Liu, and L. Yan. An evaluation of TCP vegas by live emulation. Technical Report 94-588, University of Southern California, 1994.

[5] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[6] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: An autocatalytic optimizing process. Technical Report 91-016, Politecnico di Milano, Dipartimento di Elettronica ed Informatica, 1991.

[7] P. P. Grassé. La reconstruction du nid et les coordinations interindividuelles chez *bellicositermes natalensis et cubitermes sp.* La théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.

[8] A. Khanna and J. Zinky. The revised ARPANET routing metric. *SIGCOMM 89*, 19(4):45–56, August, 19-22 1989.

[9] G. S. Malkin and M. E. Steenstrup. Distance-vector routing. In M. E. Steenstrup, editor, *Routing in Communications Networks*, chapter 3, pages 83–98. Prentice-Hall, 1995.

[10] J. M. McQuillan, I. Richer, and E. C. Rosen. The new routing algorithm for the ARPANET. *IEEE Trans. on Comm.*, 28:711–719, 1980.

[11] J. Moy. OSPF version 2. RFC 1583, Network Working Group, 1994.

[12] D. Pinelli. Evaluation of dynamic metrics for IP routing supporting integrated services (in italian). Master's thesis, CEFRIEL, Milano (IT), 1996.

[13] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz. Load balancing in telecommunications networks. *Adaptive Behavior*, 5(2), 1997.

[14] M. E. Steenstrup, editor. *Routing in Communications Networks*. Prentice-Hall, 1995.

[15] P. Stone and M. Veloso. Multiagent systems: a survey from a machine learning perspective. Submitted, 1996.