AntNet: A Mobile Agents Approach to Adaptive Routing

Gianni Di Caro and **Marco Dorigo**
IRIDIA, Université Libre de Bruxelles
50, av. F. Roosevelt, CP 194/6, 1050 - Brussels, Belgium

Technical Report IRIDIA 97-12

Abstract

This paper introduces *AntNet*, a new routing algorithm for communications networks. *AntNet* is an adaptive, distributed, mobile-agents-based algorithm which was inspired by recent work on the ant colony metaphor. We apply *AntNet* to 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 performance 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 is an important aspect of communications networks because it can greatly influence the overall network performance: good routing can cause greater throughput or lower average delays all other conditions being the same.

Routing algorithms strongly interact with congestion and admission control algorithms to determine the overall network performance. 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*.

The term stigmergy was first introduced by Grassé (Grassé, 1959) 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 as only information the pheromone trail deposited by other ants (Goss, Aron, Deneubourg, & Pasteels, 1989; Beckers, Deneubourg, & Goss, 1992).

Algorithms which take inspiration from ants' behavior in finding shortest paths have recently been successfully applied to both combinatorial optimization (Dorigo, Maniezzo, & Colorni, 1991; Dorigo, 1992; Dorigo, Maniezzo, & Colorni, 1996; Dorigo & Gambardella, 1997) and circuit switched communications network problems (Schoonderwoerd, Holland, & Bruten, 1997a; Schoonderwoerd, Holland, Bruten, & Rothkrantz, 1997b). 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 (Steenstrup, 1995).

AntNet shows the best performance and the more stable behavior for all the paradigmatical temporal and spatial traffic distributions considered. Its best competitor is still another new algorithm we introduce. Absolute performance is scored according to a scale defined by an ideal algorithm giving an empirical bound. Competitors algorithms performed poorly for heavy traffic situations and showed more sensitivity to internal parameters tuning. *AntNet*, on the other hand, has only a small set of tunable parameters and its behavior is very robust with respect to their 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.

The precise balance between throughput and average delay will be determined by the flow control algorithms operating concurrently with the routing algorithms. Citing (Bertsekas & Gallager, 1992) (pag. 367): *"the effect of good routing is to increase throughput for the same value of average delay per packet under high offered load conditions and to decrease average delay per packet under low and moderate offered load conditions"*.

Routing algorithms can be at first broadly classified into *centralized* or *distributed*, *static* or *adaptive*.

Centralized algorithms can be used only in particular cases. In general, the delays necessary to gather information about the network status and to broadcast the routing decisions make them unfeasible.

In static (or *oblivious*) routers, the path taken by a packet is determined only on the basis of its source and destination, without regard to the current network state. This path is usually chosen as the shortest one according to some cost criterion, and it can be changed to account for faulty links or nodes.

Adaptive routers are, in principle, more attractive, because they can adapt the routing policy to time and spatially varying traffic conditions. As a drawback, they can cause oscillations in selected paths. This fact can cause circular paths, as well as large fluctuations in measured performance, especially concerning average delays. In addition, adaptive routing can lead more easily to inconsistent situations, associated with node or link failures or

local topological changes. These stability and inconsistency problems are more evident for datagram networks than for virtual circuit networks (Bertsekas & Gallager, 1992). In fact, in datagram networks every packet selects the path hop by hop. This allows the system to react quickly and adapt to changes in routing tables but can also easily lead to an oscillatory behavior. Moreover, packets, to choose different paths, will likely arrive out-of-order and this can be a potential problem that has to be managed in an appropriate way by the underlying control protocol. Virtual circuit networks do not exhibit re-ordering problems and their (lower) sensitivity to routing tables updates strictly depends on the circuits creation rate and holding time.

Adaptive routers can be broken down in two broad categories: *minimal* and *non-minimal* (Bolding, Fulgham, & Snyder, 1994). Minimal routers allow packets to choose only minimal cost paths, while non-minimal algorithms allow choosing among all the available paths following some heuristic strategies. Examples of non-minimal routers are *deflection* routers, *hierarchical* routers, *cut-through* routers, *queuing* routers (see for example (Bolding et al., 1994) and (Bertsekas & Gallager, 1992) for descriptions and references).

Oblivious and minimal adaptive routing algorithms are the most widely used routing paradigms (at least taking in consideration wide-area communication networks). Considering different perspectives in minimal length paths computation, these algorithms can be further classified as *optimal routing* and *shortest path* algorithms (Bertsekas & Gallager, 1992), which are discussed in the following.

In section 5 we will introduce a novel method, *AntNet*, that shares the same optimization perspective as shortest path algorithms but not their usual implementation paradigms (depicted in sect. 2.2).

2.1 Optimal Routing

Optimal routing has a network-wide perspective and its objective is to optimize a function of all individual link flows (usually this function is a sum of link costs assigned on the basis of average packet delays).

These kind of routing models are called *flow models* because they try to optimize the total mean flow on the network. They can be characterized as multicommodity flow problems, where the commodities are the traffic flows between the sources and the destinations, the cost to be optimized is a function of the flows, subject to the constraints of flow conservation at each node and positive flow on every link. It is worth observing that the flow conservation constraint can be explicitly stated only if the traffic arrival rate is known.

The routing policy consists of splitting any source-target traffic pair at strategic points, then shifting traffic gradually among alternative routes. This often results in the use of multiple paths for a same traffic flow between an origin-destination pair.

Implicit in optimal routing is the assumption that the main statistical characteristics of the traffic are known and not time-varying. Therefore, optimal routing can be used for static and centralized/decentralized routing. It is evident that this kind of solution suffers all the problems of static routers. In addition, it has an important limitation: the optimized cost function hardly can capture all the performance measures we are interested in. Therefore, an optimal routing policy will probably determine an unbalanced situation among the relative measures of different desired performance levels.

2.2 Shortest Path Routing

Shortest path routing has a source-destination pair perspective. Its objective is to determine the shortest path between two nodes, where the link costs are computed following some statistical and/or physical description of the link states. Differently from optimal routing, here there is not a global cost function to be optimized, but the route between each node pair is considered by itself and no *a priori* knowledge about the traffic process is required (of course such knowledge could be fruitfully used).

If the link cost is static and reflects both the transmission capacity of the link and a measure of the expected traffic load over it, then computed shortest paths will be optimal on average for stationary traffic situations. Of course, serious loss of efficiency could arise in case of non-stationary conditions or when it is impossible to make assumptions about the traffic evolution.

On the other hand, if costs are assigned in a dynamic way, based on statistical measures of the link congestion state, a strong feedback effect is introduced between the routing policies and the traffic patterns. This can lead to undesirable oscillations, as has been theoretically predicted and observed in practice (Bertsekas & Gallager, 1992).

Considering the different content stored in each switch[1] routing table, shortest path algorithms can be further subdivided in two classes called *distance-vector* and *link-state* (Steenstrup, 1995).

The common behavior of most shortest path algorithms can be depicted as follows.

1. Each node assigns a cost to each of its outgoing links. This cost can be static or dynamic. In the latter case, it is updated in presence of a link failure or on the basis of some observed link-traffic statistics averaged over a defined time-window.

2. Periodically and without a required inter-node synchronization, each node sends to all of its neighbors a packet of information describing its current estimates about some quantities (link costs, distance from all the other nodes, etc.).

3. Each node upon receiving the information packet updates its local routing table and executes some class-specific actions.

Routing decisions can be made in a deterministic way, choosing the best path indicated by the information stored in the routing table, or, adopting a more flexible strategy, using all the information stored in the table to choose some randomized or alternative path.

We now describe the features specific of each class; classes differ mainly because of the different information stored in the routing tables.

Distance-vector algorithms make use of routing tables consisting of a set of triples of the form *(Destination, Estimated Distance, Next Hop)*, defined for all the destinations in the network and for all the neighbor nodes of the considered switch. In this case, the required topological information is represented by the list of the reachable nodes identifiers. The average per node memory occupation is of order $O(Nn)$, where N is

1. The term switch in this context is intended as an alias for network node. It is an appropriate term to indicate a virtual device reading the information stored in the local routing table and forwarding the packet on the selected outgoing link.

the number of nodes in the network and n is the average connectivity degree (i.e., the average number of neighbors nodes considered over all the nodes).

The algorithm works in an iterative, asynchronous and distributed way. The information that every node sends to its neighbors is the list of its last estimates of the distances from itself to all the other nodes in the network. After receiving this information from a neighbor node j, the receiving node i updates its table of distance estimates in the entry corresponding to node j as next hop node.

Routing decisions at node i are made choosing as next hop node the one satisfying the relationship:

$$arg \min_j \{d_{ij} + D_j\}$$

where d_{ij} is the assigned cost to the link connecting node i with its neighbor j and D_j is the estimated shortest distance from node j to the destination.

It can be shown that this process converges in finite time to the shortest paths with respect to the used metric if no link cost changes after a given time (Bertsekas & Gallager, 1992).

The above briefly described algorithm is known in literature as distributed *Bellman-Ford* (Bellman, 1958; Ford & Fulkerson, 1962; Bertsekas & Gallager, 1992) and it is based on the principles of dynamic programming (Bellman, 1957; Bertsekas, 1995). It is the prototype and the ancestor of a more wide class of distance-vector algorithms (Malkin & Steenstrup, 1995) developed with the aim of reducing the risk of circular loops and to accelerate the convergence in case of rapid changes in link costs.

Link-state algorithms make use of routing tables containing much more information than those used in vector-distance algorithms. In fact, at the core of link-state algorithms there is a distributed and replicated database. This database is essentially a dynamic map of the complete network, describing the details of its components and their current interconnections. Using this database as input each node calculates its best paths using an appropriate algorithm (usually Dijkstra's (Dijkstra, 1959) algorithm is used, but, as described in (Cherkassky, Goldberg, & Radzik, 1994), a wide variety of alternative efficient algorithms for shortest paths computations are available). The memory space occupation required for each node is in this case of order $O(N^2)$.

In the most common form of link-state algorithm each node acts autonomously, broadcasting information about its link costs and states and computing shortest paths from itself to all the destinations on the basis of its local link costs estimates and of the estimates received from other nodes. Each routing information packet is broadcast to all the neighbor nodes that in turn send the packet to their neighbors and so on. A distributed flooding (Bertsekas & Gallager, 1992) mechanism supervises this information transmission trying to minimize the number of re-transmissions.

As in the case of vector-distance, the described algorithm is a general template and a variety of different versions has been implemented to make the algorithm behavior more robust and efficient (Moy, 1995).

2.2.1 Links Costs Updates

We conclude our discussion of shortest paths algorithms stressing the critical role played by the choice of an appropriate link costs updating policy in case of non-static costs and time-varying traffic patterns.

It is not an easy task to find the best balance between adaptiveness and stability. Highly adaptive metrics can lead to large oscillations with consequent decrease in performance. More stable metrics can show a poor responsivity to traffic fluctuations and congestion conditions. Link costs are usually assigned in the following way attempting to reduce big variations and taking into account both long-term and short-term statistics of link congestion states (Pinelli, 1996):

1. The first step is to define a metric for the links cost evaluation. The cost is updated at regular instants $t_0, t_1, t_2, \ldots, t_i \ldots$. During every time interval $I_i = (t_{i-1}, t_i]$ the cost is not varied, but "instantaneous" raw cost samples, o_t, are calculated (e.g., if the metric is a function of the number of bits in the link queue, a new o_t is computed at every instant t of a packet arrival or departure). Statistics about the values $o_t, t \in I_i$, are recorded and at the end of the time-window I_i a mean value of this raw short-term cost $\bar{s}_{t_i}$ is computed:

$$\bar{s}_{t_i} = \sum_{t \in I_i} \frac{o_t}{N_{I_i}},$$

   where N_{I_i} is the number of sampled o_t values in the time-window I_i.

2. At the same time, all the sampled o_t are used to update an exponential mean $\bar{l}_t$ of the link cost (long-term cost evaluation)

$$\bar{l}_t \leftarrow \bar{l}_t + \alpha(o_t - \bar{l}_t)$$

3. At the end of each time window the short and the long-term averages are averaged and the obtained value is saturated by a linear function with c_{max} and c_{min} as extreme values:

$$c_{t_i} = \frac{\bar{l}_{t_i} + \bar{s}_{t_i}}{2}$$

$$c_{t_i} \leftarrow max(\ min(ac_{t_i} + b,\ c_{max}),\ c_{min}\ ),$$

   where a and b define respectively the slope and the offset of the linear saturation function.

4. The obtained cost value c_{t_i} is normalized with respect to a maximum admitted variation Δ_{max}, and a final link cost evaluation is obtained. This cost will be held for all the duration of the next time interval I_{i+1}:

$$c_{t_i} := \begin{cases} c_{t_i} + sign(c_{t_i} - c_{t_{i-1}})\Delta_{max} & if\ |c_{t_i} - c_{t_{i-1}}| > \Delta_{max} \\ \\ c_{t_i} & otherwise. \end{cases}$$

Several different metrics can be used, we will discuss some of them in section 7.

3. Routing in the Internet

Several versions of shortest path algorithms have been used during the years for the routing management in Internet and in its predecessor, ARPANET.

The first routing algorithm on ARPANET, implemented in 1969, was a distributed adaptive asynchronous Bellman-Ford algorithm. It used a dynamic link cost metric based on measures of traffic congestion on the links and in particular of the number of packets waiting in the link queue. Because this metric led to considerable oscillations, a large positive constant was added to stabilize them. Unfortunately, this solution dramatically reduced the sensitivity to traffic congestion situations.

These problems led in 1980 to a second version of the routing algorithm in ARPANET, known as *Shortest Path First* (SPF)(McQuillan, Richer, & Rosen, 1980), a link-state routing algorithm. A dynamic link cost metric was still used, based on the statistics of the delays experienced by data packets. Delays for each single link were computed summing the queuing and the transmission and propagation delays.

This new class of algorithms exhibited more robust behaviors than distance-vector algorithms, but in this case too, in case of heavy load conditions, the observed oscillations were too large. So, after some revisions concerning the reduction of the allowed variability of the links costs (Khanna & Zinky, 1989; Zinky, Vichniac, & Khanna, 1989), the current routing algorithm of Internet, *Open Shortest Path First* (OSPF) (Moy, 1994) is a link-state algorithm with a static metric assigned by network administrators. Only temporary or permanent topological alterations are flooded in the network.

This short historical retrospective is intended to highlight the difficulties encountered in the implementation of adaptive routing algorithms for the Internet.

The current, essentially static, Internet routing system is not clearly the "best solution", it is just the "best compromise" among efficiency, stability and robustness that has been found so far. Much of the complexity has been moved from the routing system to the congestion control system, to avoid high congestion states due to the lack of a mechanism for an adaptive flows subdivision among multiple paths.

But the need for an efficient adaptive routing system is still there, and this need is made more evident by forthcoming high-speed networks, which will allow the formulation of requests of Quality of Service (QoS) (Crawley, Nair, Rajagopalan, & Sandick, 1996; Pinelli, 1996). This means that a user-session can specify the quality of service it needs to receive from the network. For example it could specify the maximum and minimum levels of throughput, or of end-to-end delay, or the maximum acceptable rate of packets loss, etc. In this scenario, it will be of critical importance an efficient adaptive routing policy able to support the flow admission control algorithm both in the creation of QoS sessions with very strict requirements, and in the global optimization of the network resources usage (Crawley et al., 1996).

It is in this perspective that we developed a robust, adaptive, and distributed routing system, *AntNet*, implemented following alternative schemes with respect to those of classical shortest path implementations here briefly reported and that showed more than one limitation[2].

2. Of course, taking in consideration the huge number of interests involved with the Internet (and with QoS), a large number of different interesting solutions are proposed (see for example the Request For

4. The Communication Network Model

We focus our experiments on datagram networks with irregular topology and without mechanisms for congestion and admission control. This choice has some well-defined motivations. First, we want to check the behavior of our algorithm and of its competitors in conditions which minimize the number of interacting components. In fact, any congestion or admission control algorithm has a considerable impact on the network performance and it is difficult to evaluate the impact of the control algorithm by itself and of its interactions with the routing algorithm. As an example, some authors reported an improvement ranging from 2 to 30% in various performance measures for real Internet traffic (Danzig, Liu, & Yan, 1994), changing from the Reno version to the Vegas version of the TCP (Peterson & Davie, 1996) (other authors even claimed an improvement ranging from 40 to 70% (Brakmo, O'Malley, & Peterson, 1994)). In presence of variations of this entity, it becomes difficult to choose the "right" congestion avoidance mechanism and to understand its relationships with the underlying routing algorithm.

Second, we chose to work with datagram networks because using virtual circuits in (high-speed) communications networks is tightly bounded to considerations of QoS applications (see sec. 3). Therefore, complex admission control algorithms should be introduced. But, again, as a first step we think that it is more reasonable trying to check the validity of a routing algorithm reducing the number of components heavily influencing the network behavior. Anyway, our algorithm presents no evident limitations to prevent its use in virtual circuit networks.

In the following, a brief description of the features of the considered network abstraction is given.

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. All the traveling packets are subdivided in two classes: data and routing packets. All the packets in the same class have the same priority, so they are queued and served only on the basis of a first-in-first-out policy, but routing packets have a greater priority than data packets.

These are fed into the network by applications whose arrival rate is dictated by a selected probabilistic model. By application, we mean a process sending data packets from an origin node to a destination node. The number of packets to send, their sizes and the intervals between them are assigned according to some defined stochastic process.

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

Comments (RFC) of the Network Working Group at ftp://ds.internic.net), besides those we presented in this paper.

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.

After transmission, a stochastic process generates service times for the newly arrived packet, that is the delay between its arrival time and the time when it will be ready to be put in the buffer queue of the selected outgoing link.

Situations causing a temporary or steady alteration of the network topology or of its physical characteristics are not taken into account (link or node failure, adding or deleting of network components, etc.).

We developed a complete network simulator in C++. It is a discrete event simulator using as main data structure an event list which holds the next future events. The simulation time is a continuous variable and is set by the currently scheduled event. The aim of the simulator is to closely mirror the essential features of the concurrent and distributed behavior of a generic communication network without sacrificing efficiency and flexibility in code development.

5. AntNet: an Adaptive Agent-based Routing Algorithm

As shown before, the routing problem is a stochastic distributed multiobjective problem. Information propagation delays, and the difficulty to completely characterize 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 suited for a multiagent approach like our *AntNet* system, composed by two sets of *homogeneous mobile agents* (see (Stone & Veloso, 1996) for an agents taxonomy), called in the following *forward* and *backward* ants.

Agents[3] 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. We can broadly classify them as *deliberative* agents, because they behave *reactively* retrieving a pre-compiled set of behaviors, and 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*) $F_{s \to d}$, 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 $S_{s \to d}(k)$ and inserted in a dictionary structure $D_{s \to d}$, 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 neighbor node, or, with a tiny probability (exploration probability), assigning the same 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. In the following, we will use interchangeably the terms Ant and Agent.

3. If a cycle is detected, that is, if an ant is forced to return to an already visited node, the cycle's nodes are popped from the ant's stack and all the memory about them is 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 takes 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

   $$\sum_{n \in \mathcal{N}_k} P_{in} = 1, i \in [1, N], \quad \mathcal{N}_k = \{neighbors(k)\};$$

   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 node k 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 by 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 (see fig. 1 for an example).

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 length from a physical point of view (number of hops, transmission capacity of the used links, processing speed of the crossed nodes) and from a traffic congestion point of view. This last aspect is extremely important: forward ants share the same queues as data packets (backward ants 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. In other words, links on congested paths will be given only a little, very delayed reward.

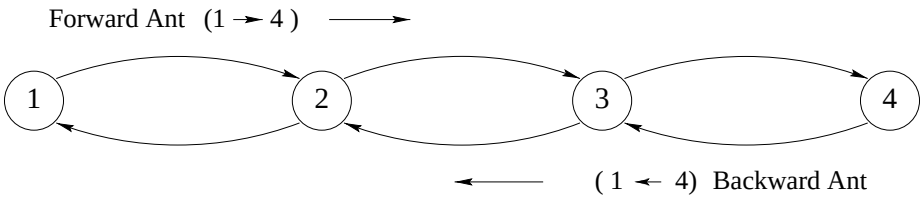


Figure 1: Example of the *AntNet* behavior. The forward ant, $F_{1\to 4}$, moves along the path $1 \to 2 \to 3 \to 4$ and, arrived at node 4, launches the backward ant that will travel in the opposite direction. In each node k, $k = 2, \ldots, 4$, the backward ant will use the stack contents $S_{1\to 4}(k)$ to update the values for $Trip_k(\mu_i, {\sigma_i}^2)$, $i = 1, \ldots, (k-1)$. At the same time the routing table will be updated by incrementing the goodness of the last node from where it comes when node 4 is the destination node, that is, incrementing P_{4j}, $j = k+1, j \neq 4$ and decrementing the values of P for the other neighbors (here not shown). The increment will be a function of the trip time experienced by the forward ant going from node k to its destination node 4.

We used the time measure as a reinforcement signal to provide structural and temporal credit assignment. The observed times are absolute measures and there is no way we can associate with them a precise error measure. We cannot score the trip times according to an absolute scale because "optimal" times depend on traffic and/or components failure states, and they have to be considered from a network wide point of view.

If the network is in a congested state, all the trip times will score poorly with respect to times observed in low load situations, but we should nevertheless be able to understand that the network is under congestion and therefore we should give an adequate reward for a trip time high from an absolute point of view but significantly low with respect to the trip times observed in the congested situation. We can see that our credit assignment problem is the typical one arising in the reinforcement learning field (Bertsekas & Tsitsiklis, 1996; Kaelbling, Littman, & Moore, 1996): we cannot associate the realized performance (trip time) with an exact error measure. We can only give "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 a dimensionless 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 hand, if the mean is not stable, the raw values r' cannot be completely considered reliable and, in this case, the quantity

$$U(\sigma, \mu; a') = 1 - e^{-a'\,\frac{\sigma}{\mu}}\ ,$$

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 generate a more effective traffic-dependent evaluation measure. The situation is similar to that in Actor-Critic systems (Barto, Sutton, & Anderson, 1983): a raw "reinforcement" signal (the experienced trip time, in our case) is processed by a critic module which is learning a model of the underlying process, and then is fed to the learning system.

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 final 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 stigmergy paradigm for agent communication. In fact, each agent is complex enough 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 performance 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 it natural and convenient to use a *blackboard* type of inter-agent 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* (Grassé, 1959; Schoonderwoerd et al., 1997a; Stone & Veloso, 1996). We used 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).

6. Routing Algorithms Used for Comparison

To evaluate the performance of *AntNet*, we selected a set of competitors algorithms from the shortest paths class. We wanted to compare our algorithm with the current Internet standards and with some improved versions of them closely reflecting state-of-the-art for routing algorithms.

The following algorithms have been implemented.

OSPF: is our implementation of the official Internet routing algorithm (Moy, 1994). 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, as described in section 4. Therefore, the original algorithm is reduced to a simple static shortest path calculation. Shortest paths for a sample data packet of size 512 bytes are computed and stored in routing tables at the start of the simulation.

BF: is an implementation of the asynchronous distributed Bellman-Ford algorithm with dynamic metrics (Bertsekas & Gallager, 1992). The algorithm has been implemented following the guidelines of sections 2.2 and 2.2.1). Vector-distance Bellman-Ford-like algorithms are today in use mainly for intradomain routing, because they are used in

the Routing Information Protocol (RIP) (Malkin & Steenstrup, 1995) supplied with the BSD version of Unix.

SPF: is the prototype of link-state algorithms with dynamic metric for link costs evaluations. It has been implemented respecting the indications of sections 2.2 and 2.2.1 for link-state algorithms. A similar algorithm was implemented in the second version of ARPANET (McQuillan et al., 1980) and in its successive revisions (Khanna & Zinky, 1989). 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 feature that the shortest paths are computed on the basis of locally updated information and that the costs of far links are all set to the same value. The algorithm makes "myopic" predictions and does not trust delayed information arriving from far areas of the network.

Daemon: is an approximation of an ideal algorithm. It defines an empirical bound on the achievable performance. It gives some information about how much improvement is still possible. In the absence of any *a priori* assumption on traffic statistics, the empirical bound can be defined by an algorithm possessing a "daemon" able to read in every instant the state of all the queues in the network and then calculating instantaneous "real" costs for all the links and assigning paths on the basis of a network wide shortest paths re-calculation for every packet hop. Links costs used in shortest paths calculations are the following:

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

where d_{l_i} is the delay for link l_i, b_{l_i} is its bandwidth, S_p is the size (in bits) of the data packets doing the hop, $S_{Q(l_i)}$ is the size (in bits) of the queue of link l_i, $\bar{S}_{Q(l_i)}$ is the exponential mean of the size of link's queue and it is a correction to the actual size of the link queue on the basis of what observed until that moment. This correction is weighted by the α value 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 (Pinelli, 1996).

1. The links costs are all set to 1. This is in reality a static metric counting only the number of hops (for this and the next metrics, a value of infinity is assigned to the cost in case of link failure).

2. The link cost is measured by the link queue occupation during the last time-window. More precisely, by the fraction of time of non-empty queue with respect to the empty-queue period.

3. The link cost is set to the sum of the mean delay of a packet in the link queue plus the transmission delay, where the average is done over the last time-window. We call this metric DEL.

4. 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. They are used to compute the link cost in the following way (Khanna & Zinky, 1989): $1 - \frac{\bar{T}_l}{\bar{D}_{q(l)}}$. We call this metric HND (hop normalized delay).

5. 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 of the DEL metric with the HND metric.

7. Experimental Settings

The functioning of a communication network is governed by many components which may interact in nonlinear and unpredictable ways. Therefore, the choice of a meaningful testbed to compare competing algorithms is no easy task.

A limited set of classes of tunable components are defined and for each of them our choices are explained:

Topology and physical properties of the net. Topology can be defined on the basis of a real net instance or it can defined by hand, to better analyze the influence of important topological features (like diameter, connectivity degree, etc.).

Switches are mainly characterized by their buffering and processing capacity, whereas links by propagation delay, bandwidth and streams multiplexing scheme. For both, fault probability distributions should be defined.

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 shown in fig. 2. The bandwidth is 1.5 Mbit/s for every link, the assigned link or node fault probability is null, local buffers have infinite capacity and links are accessed through statistical multiplexing.

Traffic patterns. Traffic is defined in terms of open sessions between a pair of active applications situated on different nodes. Traffic patterns can show a huge variety of forms, depending on the characteristics of each session and on their distribution from a geographical and from a temporal point of view.

Each single session is characterized by the number of transmitted packets, and by their size and inter-arrival time distributions. More generally, priority, costs and requested quality of service should be used to completely characterize a session.

An application can be characterized by the number of open sessions and by their inter-arrival distribution. Applications time arrivals will follow some defined statistics. Their geographical distribution over the net is controlled by the probability assigned to each node to be selected as an application start or end-point.

In our experimental settings, we considered only one-session applications, so in the following we will use the terms session and application interchangeably. 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) (Pinelli, 1996).

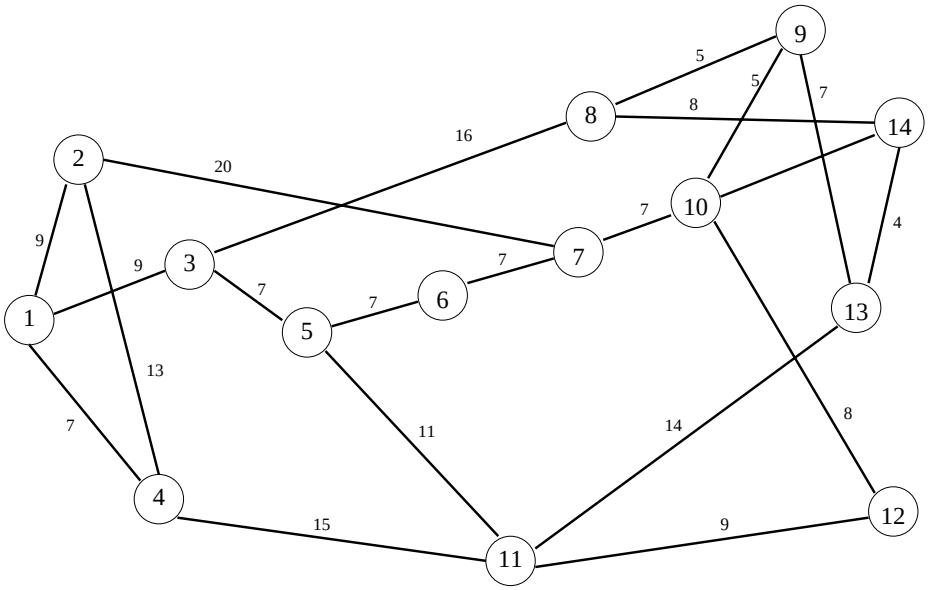


Figure 2: NSFNET. Numbers within circles are node identifiers, while numbers on links are propagation delays in msec. Each edge in the graph represents a pair of directed links.

- 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 stationary situation. The packet inter-arrival time is a constant value of 150 ms and the packet size distribution is a negative exponential with mean 512 bytes. This means a flow of 27.3 Kbit/s for each session.
- In the *dynamic* model, sessions are activated following a negative exponential distribution for the inter-arrival times. The mean value of the distribution is fixed to 15 sec. The total number of packets per session, their sizes and their inter-arrival times, are negative exponentially distributed, with respectively mean values of 50, 512 bytes and 10 μsec. In this case, sessions are "bursty" sessions: they generate all the packets in a very short time (500 μsec on average) after their set up, and, after their termination, a long time interval (15 seconds on average) elapses before the creation of a new session. Hence, data flows are highly irregular both from a temporal and a spatial point of view.

The geographical distribution of the traffic patterns has been modeled considering 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 $n \geq N^2$ open sessions. Start and end-points are selected uniformly randomly (i.e., all the nodes have the same probability to be selected).

- *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): similarly to the UDHS case, two different types of load are concurrently present. One is the same as in the UR case and the other is the same hot-spots component described for UDHS.

Metrics for performance evaluation. Depending on the type of services delivered on the network and on their associated costs, many performance metrics could be defined.

We focused on standard metrics for performance evaluation, considering only sessions with equal costs, benefits and priority and without the possibility of requests for special services like real-time. In this framework, the measures we are interested in are: *throughput* (delivered bits/sec), *average delay* for data packets (sec), and *network's capacity usage* (for data and routing packets), expressed as the sum of the used link capacities divided by the total available link capacity.

Routing algorithms parameters. In the previous sections we described the implemented routing algorithms; here we report the setting of parameters, that are kept constant for all the experiments.

For *AntNet*, the generation interval of ants is set to 1 (sec), start and end-points are sampled uniformly over the network, the exploration probability is set to 0.002, the ant size is 24 bytes for the forward ant and $24 + N_h$ bytes,N_h =number of hops, for the backward ant. The ant processing time is 2 ms.

For shortest paths algorithms (see sec. 2.2), the length of the time interval between consecutive routing information broadcasting and the length of the time window I_i to average link costs c_{t_i}, 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 (these size values have been set on the basis of the real implementations of shortest path algorithms). Concerning the links costs calculations at the end of every time window I_i (see sec. 2.2.1), settings are: the weight factor α for the long term exponential average is set to 0.4, the link costs can take values in the interval $[c_{min}, c_{max}]$ set to $[0.005, 2.0]$ (sec) with max admitted variation Δ_{max} of 0.05 (sec), the slope a of the linear saturation function is 1.0 and its offset b is set to 0.0. As we said in section 6, after several experiments we observed that the best performance was achieved using the hybrid metric HDN-DEL for link costs evaluations, so we kept it constant during all the experiments.

8. Results

We report results according to the different spatial and temporal traffic distributions considered in sect. 7. That is, for each one of the two temporal traffic patterns, CBR and VBR, results relative to the four cases of spatial traffic distribution are presented. Some

"extreme" cases are considered. In fact, for very low, uniform and almost stationary traffic loads, the six presented algorithms behave almost in the same way and their performance is near to optimal. Increasing traffic load and/or considering strongly asymmetric spatial distributions, it is possible to appreciate different behaviors in algorithms performance. We report results for some meaningful cases, which well represent algorithm behavior under heavy traffic conditions. The time length of the simulation has been set to 120 seconds. We observed that after this time interval the behavior of the algorithm is already well characterized. A 100 seconds of "learning" time has been assigned to the algorithms to learn routing tables over the network in absence of traffic.

Reported values are averaged over 10 trials and a bootstrap (Efron & Tibshirani, 1993) procedure has been applied to get a better estimate of the variance. In fact, we have to take in consideration that large fluctuations can appear among trials because similar traffic situations (in terms of defined traffic parameters and distributions) can led to very different congestion states leading to significant oscillations in performance.

In tables 1-8, observed mean values (and their associated 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, that is, uniform-deterministic, uniform-random, uniform-deterministic-hot-spots and uniform-random-hot-spots. Tables 5-8 follow the same scheme but are relative to the CBR case for temporal traffic distribution. Experimental conditions are explained with detail in tables captions with the following conventions: N_{UD} =number of active sessions between all the node pairs; N_{UR} =number of total randomly selected sessions in the network, except for hot-spot sessions; N_{HSN} =number of hot-spot nodes; N_{HSS} =number of hot-spot sessions.

From the values in the tables is evident that *BF*, the algorithm based on distributed Bellman-Ford is "out-of-competition", in fact it scores very poorly with respect to the others, in all the above situations. Therefore, in the graphs 3-7 its results are not presented. Plots refer to some of the cases considered in tables. For each of the considered situations, temporal evolution of the average packet delay is shown for every algorithm. Throughput plots are not shown because of space reasons and because throughput performance presents much less striking differences than that for packet delays as it is possible to see from the tables. For space reasons, not all the data are reported; we chose the most meaningful ones.

Results about the network usage metric are not reported in detail since no differences among algorithms were observed for both data and routing packets. For all the "pathological" cases considered, the traffic caused by the routing packets was always negligible with respect to traffic generated by data packets which caused almost the saturation of the network bandwidth.

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 $N_{HSN} = 5$.

	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. $N_{UR} = 840, N_{HSS} = 5$ and $N_{HSN} = 5$.

	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. $N_{UR} = 840$.

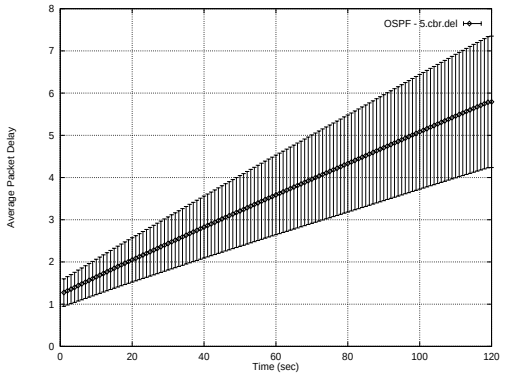
	AntNet	OSPF	SPF	SPF_1F	Daemon	BF
Mean Delay (sec)	0.79 (0.18)	4.63 (1.03)	2.01 (0.50)	2.36 (0.67)	0.06 (0.01)	3.90 (1.05)
Throughput (10^7 bits/sec)	2.219 (0.011)	2.013 (0.011)	2.171 (0.023)	2.141 (0.008)	2.205 (0.007)	1.280 (0.065)

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

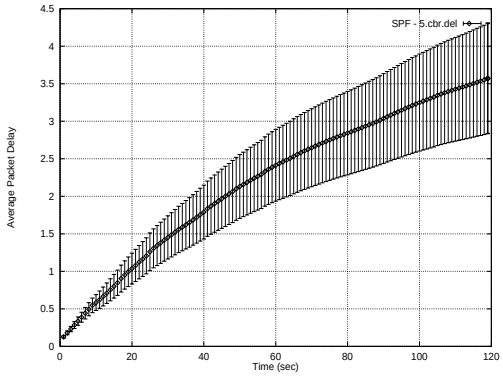
	AntNet	OSPF	SPF	SPF_1F	Daemon	BF
Mean Delay (sec)	3.37 (0.60)	12.00 (2.25)	9.60 (1.44)	11.48 (1.52)	3.28 (0.54)	4.19 (1.97)
Throughput (10^7 bits/sec)	3.134 (0.060)	2.128 (0.044)	2.815 (0.047)	2.480 (0.054)	3.140 (0.058)	1.250 (0.131)

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

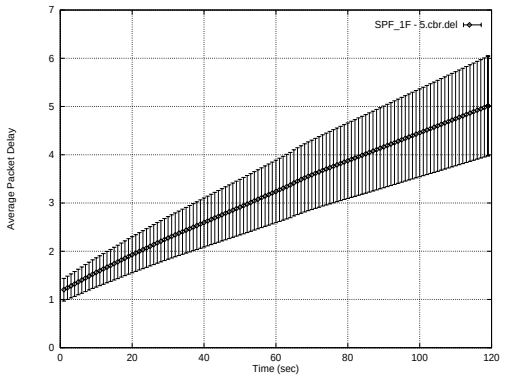
	AntNet	OSPF	SPF	SPF_1F	Daemon	BF
Mean Delay (sec)	3.18 (0.75)	10.30 (1.94)	9.42 (0.85)	9.25 (1.00)	2.83 (0.78)	5.09 (1.28)
Throughput (10^7 bits/sec)	2.986 (0.014)	2.235 (0.009)	2.619 (0.008)	2.350 (0.021)	3.012 (0.008)	1.321 (0.140)



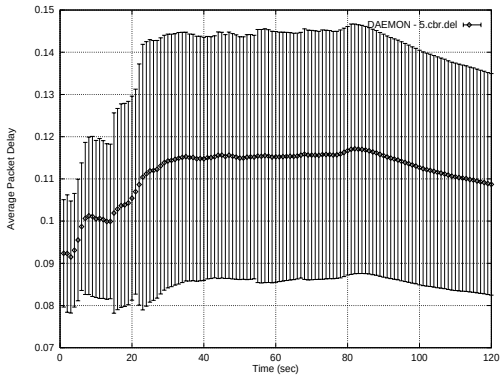
OSPF



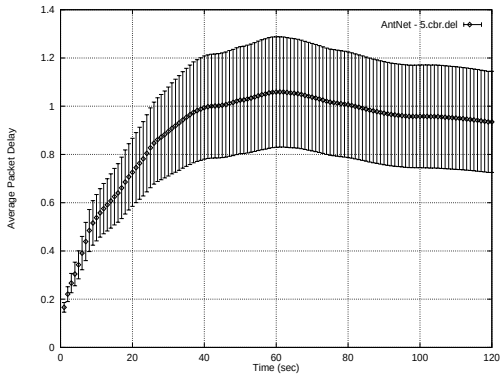
SPF



SPF_1F

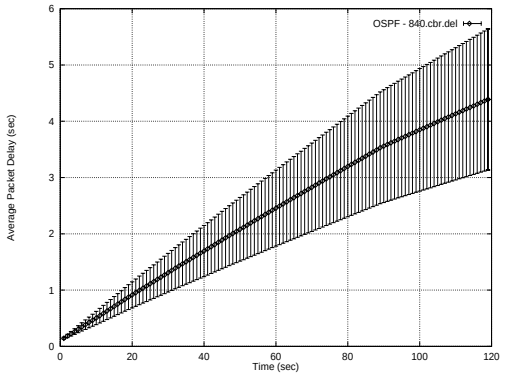


Daemon

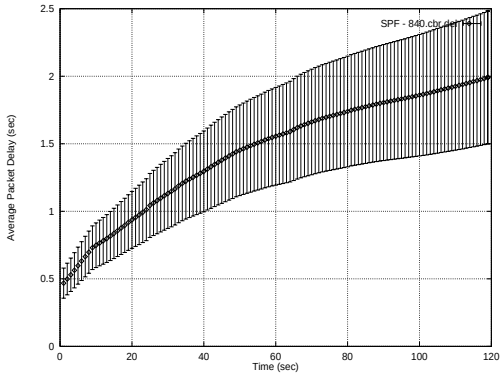


AntNet

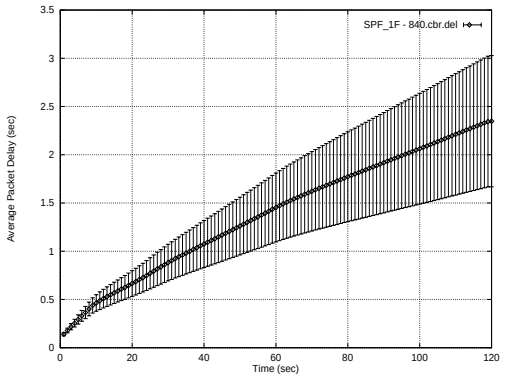
Figure 3: Average packet delay for *CBR* temporal traffic distribution and *Uniform-deterministic* spatial traffic distribution. $N_{UD} = 5$. Note that the graphs have different scales.



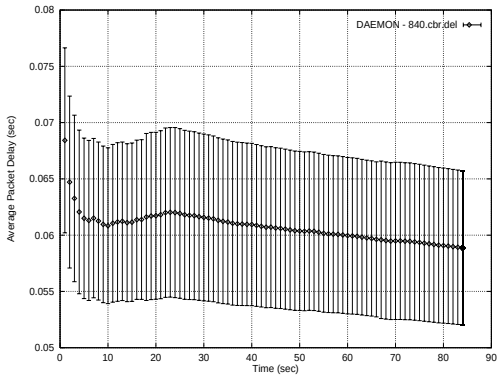
OSPF



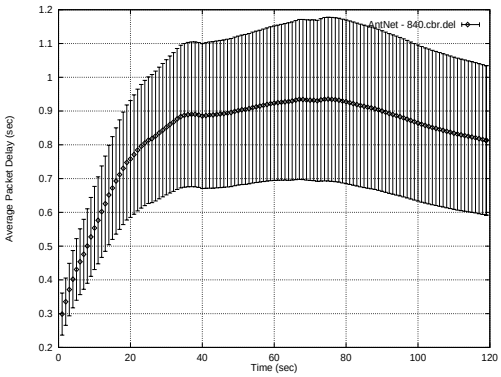
SPF



SPF_1F



Daemon



AntNet

Figure 4: Average packet delay for *CBR* temporal traffic distribution and *Uniform-random* spatial traffic distribution. $N_{UR} = 840$. Note that the graphs have different scales.

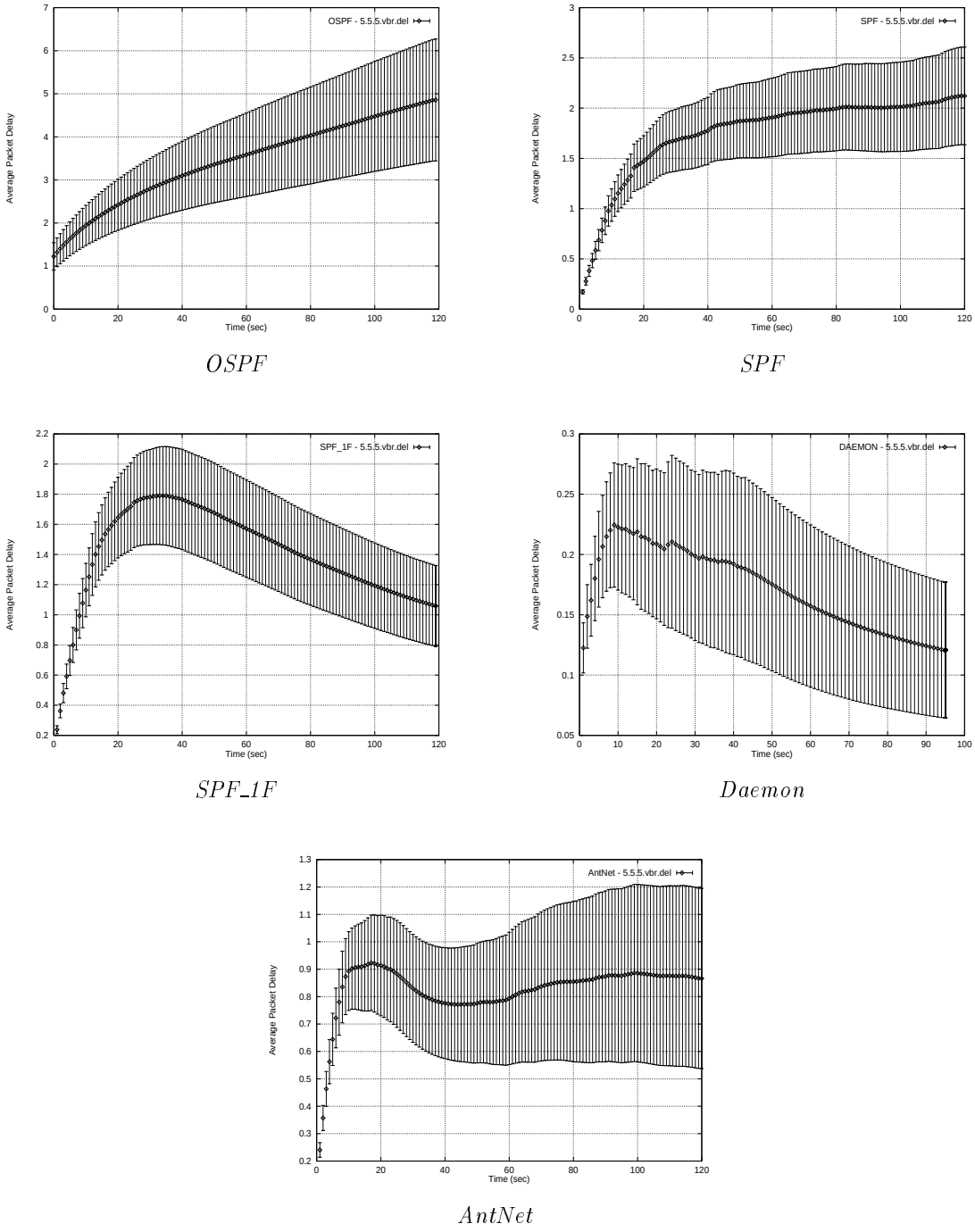
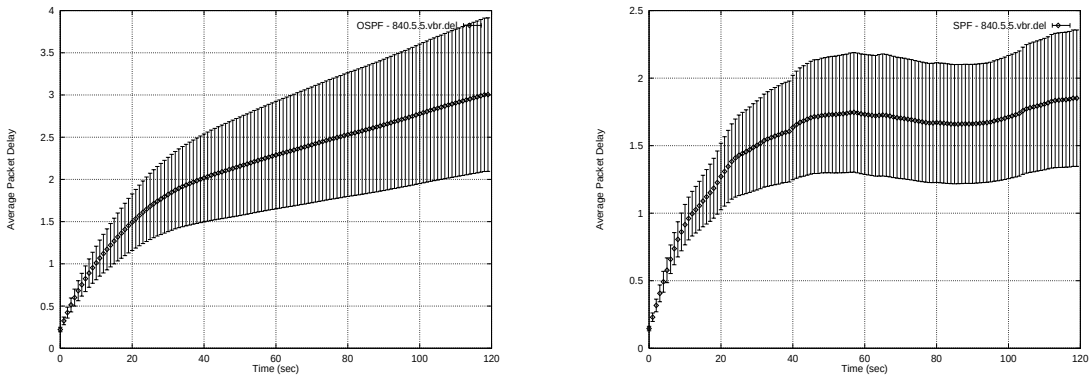
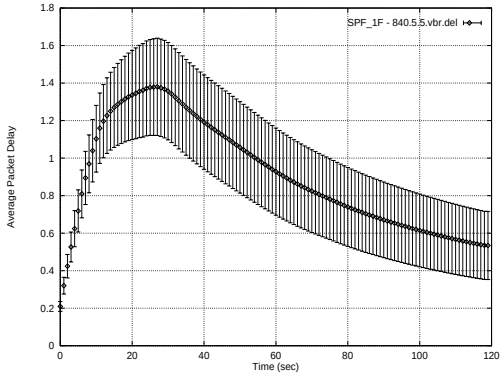


Figure 5: Average packet delay for *VBR* temporal traffic distribution and *Uniform-deterministic-hot-spots* spatial traffic distribution. $N_{UD} = 5, N_{HSS} = 5$ and $N_{HSN} = 5$. Note that the graphs have different scales.

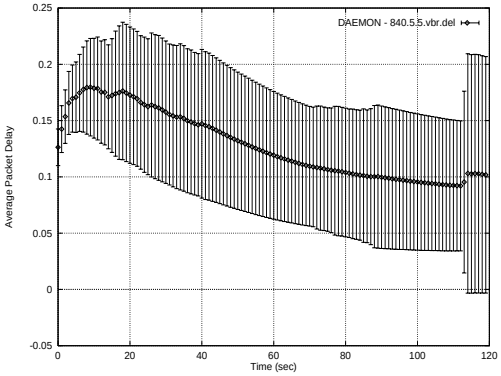


OSPF

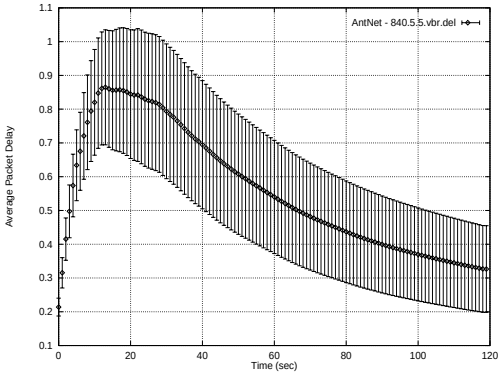
SPF



SPF_1F



Daemon



AntNet

Figure 6: Average packet delay for *VBR* temporal traffic distribution and *Uniform-random-hot-spots* spatial traffic distribution. $N_{UR} = 840, N_{HSN} = 5$ and $N_{HSN} = 5$. Note that the graphs have different scales.

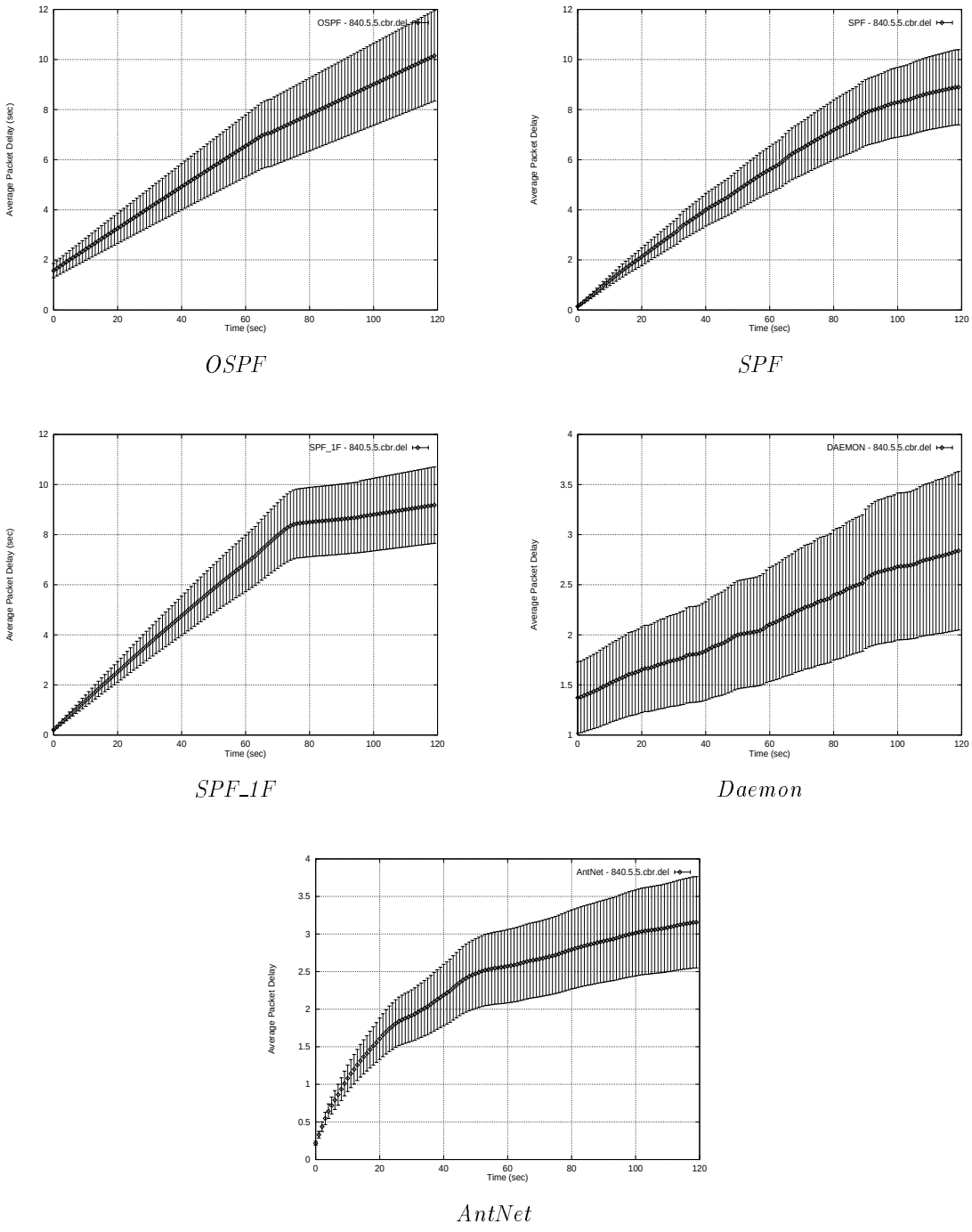


Figure 7: Average packet delay for *CBR* temporal traffic distribution and *Uniform-random-hot-spots* spatial traffic distribution. $N_{UR} = 840, N_{HSS} = 5$ and $N_{HSN} = 5$. Note that the graphs have different scales.

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 performance. Of course, the *Daemon* algorithm has always the best performance, as expected, and comparing its performance with that of *AntNet* we can see that in the half of the cases *AntNet* performance is almost the same within statistical uncertainties, confirming in this way the excellent behavior of our algorithm in accordance with 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 significant fluctuations, both in terms of absolute performance and of stability. *AntNet* was more stable in performance and in behavior, that is, always moving rapidly toward a good stable delay value after an initial transitory phase.

9. Summary 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 a primitive form of communication among agents.

Its behavior with respect to throughput, mean packet delay and resources usage, has been compared to the behavior of other five shortest paths routing algorithms. As a testbed we considered heavy traffic conditions for some representative temporal and spatial traffic distributions for a real network instance.

AntNet was always, within the statistical fluctuations, among the best performing algorithms, being in some cases the best one. Differently from the other algorithms, *AntNet* showed always a robust behavior, being able to rapidly reach a good stable level in performance. Moreover, the proposed algorithm has a negligible impact on network resources and a small set of robustly tunable parameters.

The features of the proposed algorithm make it an interesting alternative to classical shortest path routing algorithms. For a more complete validation of the approach, a more comprehensive analysis, taking into account congestion and flow control mechanisms, will be necessary.

Acknowledgements

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.

References

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transaction on Systems, Man and Cybernetics, SMC-13*, 834–846.

Beckers, R., Deneubourg, J. L., & Goss, S. (1992). Trails and U-turns in the selection of the shortest path by the ant Lasius Niger. *Journal of Theoretical Biology*, *159*, 397–415.

Bellman, R. (1957). *Dynamic programming*. Princeton University Press.

Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics*, *16*(1), 87–90.

Bertsekas, D. (1995). *Dynamic Programming and Optimal Control*. Athena Scientific.

Bertsekas, D., & Gallager, R. (1992). *Data Networks*. Prentice-Hall.

Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Bolding, K., Fulgham, M. L., & Snyder, L. (1994). The case for chaotic adaptive routing. Tech. rep. CSE-94-02-04, Department of Computer Science, University of Washington, Seattle.

Brakmo, L. S., O'Malley, S. W., & Peterson, L. L. (1994). TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of ACM SIGCOMM 94*.

Cherkassky, B. V., Goldberg, A. V., & Radzik, T. (1994). Shortest Paths Algorithms: Theory and Experimental Evaluation. In Sleator, D. D. (Ed.), *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 94)*, pp. 516–525 Arlington, VA. ACM Press.

Crawley, E., Nair, R., Rajagopalan, B., & Sandick, H. (1996). A framework for QoS-based routing in the internet. Internet Draft (expired in September, 1997) draft-ietf-qosr-framework-00, Internet Engineering Task Force (IEFT).

Danzig, P. B., Liu, Z., & Yan, L. (1994). An Evaluation of TCP Vegas by Live Emulation. Tech. rep. UCS-CS-94-588, Computer Science Department, University of Southern California, Los Angeles.

Dijkstra, E. W. (1959). A Note on Two Problems in Connection with Graphs. *Numer. Math*, *1*, 269–271.

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms (*in Italian). Ph.D. thesis, Politecnico di Milano (IT), Dipartimento di Elettronica ed Informatica.

Dorigo, M., & Gambardella, L. M. (1997). Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, *1*(1), 53–66.

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). The Ant System: An Autocatalytic Optimizing Process. Tech. rep. 91-016, Politecnico di Milano (IT), Dipartimento di Elettronica ed Informatica.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics–PartB*, *26*(1), 29–41.

Efron, B., & Tibshirani, R. (1993). *An Introduction to Bootstrap*. Chapman & Hall.

Ford, L., & Fulkerson, D. (1962). *Flows in Networks*. Prentice-Hall.

Goss, S., Aron, S., Deneubourg, J. L., & Pasteels, J. M. (1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, *76*, 579–581.

Grassé, P. P. (1959). 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.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Khanna, A., & Zinky, J. (1989). The Revised ARPANET Routing Metric. *SIGCOMM 89*, *19*(4), 45–56.

Malkin, G. S., & Steenstrup, M. E. (1995). Distance-Vector Routing. In Steenstrup, M. E. (Ed.), *Routing in Communications Networks*, chap. 3, pp. 83–98. Prentice-Hall.

McQuillan, J. M., Richer, I., & Rosen, E. C. (1980). The New Routing Algorithm for the ARPANET. *IEEE Transactions on Communications*, *28*, 711–719.

Moy, J. (1994). OSPF Version 2. Request For Comments (RFC) 1583, Network Working Group.

Moy, J. (1995). Link-State Routing. In Steenstrup, M. E. (Ed.), *Routing in Communications Networks*, chap. 5, pp. 135—157. Prentice-Hall.

Peterson, L. L., & Davie, B. (1996). *Computer Networks: A System Approach*. Morgan Kaufmann.

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

Schoonderwoerd, R., Holland, O., & Bruten, J. (1997a). Ant-like agents for load balancing in telecommunications networks. In *Proceedings of the First International Conference on Autonomous Agents,*, pp. 209–216 Marina del Rey, CA. ACM Press.

Schoonderwoerd, R., Holland, O., Bruten, J., & Rothkrantz, L. (1997b). Load Balancing in Telecommunications Networks. *Adaptive Behavior*, *5*(2).

Steenstrup, M. E. (Ed.). (1995). *Routing in Communications Networks*. Prentice-Hall.

Stone, P., & Veloso, M. (1996). Multiagent systems: a survey from a Machine Learning persective. Submitted.

Zinky, J., Vichniac, G., & Khanna, A. (1989). Performance of the Revised Routing Metric in the ARPANET and MILNET. In *MILCOM 89*.