A society of ant-like agents for adaptive routing in networks

by

Gianni Di Caro

IRIDIA, Université Libre de Bruxelles
50, av. F. Roosevelt, CP 194/6, 1050 - Brussels, Belgium
gdicaro@iridia.ulb.ac.be

Supervised by:

Marco Dorigo

Ph.D., Maître de Recherches du FNRS
IRIDIA, Université Libre de Bruxelles
50, av. F. Roosevelt, CP 194/6, 1050 - Brussels, Belgium

May, 2002

A thesis submitted in partial fulfilment of the requirements of the *Université Libre de Bruxelles, Faculté des Sciences Appliquées*, for the *Diplome d'Etudes Approfondies (DEA).*

Abstract

This thesis introduces AntNet, a novel approach to the adaptive learning of routing tables in communications networks. AntNet is a distributed, mobile agents based Monte Carlo system that was inspired by recent work on the ant colony metaphor for solving optimization problems. AntNet's agents concurrently explore the network and exchange collected information. The communication among the agents is indirect and asynchronous, mediated by the network itself. This form of communication is typical of social insects and is called *stigmergy*. We compare our algorithm with six state-of-the-art routing algorithms coming from the telecommunications and machine learning fields. The algorithms' performance is evaluated over a set of realistic testbeds. We run many experiments over real and artificial IP datagram networks with increasing number of nodes and under several paradigmatic spatial and temporal traffic distributions. Results are very encouraging. AntNet showed superior performance under all the experimental conditions with respect to its competitors. We analyze the main characteristics of the algorithm and try to explain the reasons for its superiority.

Preface

This DEA thesis is based on a number of research papers published in different forms during approximately the last four years. In this thesis their content has been revised, to take into account for the new developments both in the research field and in the way we think about our work. In particular, most of the general background on best-effort networks and the material concerning with the AntNet and AntNet-FA algorithms has been imported here from the following sources: (i) a main paper published in the Journal of Artificial Intelligence Research (JAIR) (Di Caro & Dorigo, 1998b), (ii) several conference papers (Di Caro & Dorigo, 1997c, 1998a, 1998c, 1998d, 1998f), (iii) some preparatory unpublished technical reports (Di Caro & Dorigo, 1997a, 1997b), and (iv) talks in conference without proceedings (Di Caro & Dorigo, 1997a, 1997b). The algorithm AntNet++ is introduced for the first time in this thesis. It has been designed on the basis of the previous two algorithms but represents the result of a process at the same time of logical abstraction and expansion of the range of activities to a much wider spectrum of services than the simple best-effort. In the same stream of research, along the road that took us to AntNet++, we designed other two algorithms (Di Caro & Dorigo, 1998e; Di Caro & Vasilakos, 2000) that have been presented in two conferences but that are not described here. This thesis is the result of a process of synthesis and abstraction over this four years work in the field of adaptive routing in networks. This process happened concurrently to a similar process of synthesis and abstraction that we have been doing in the field of algorithms inspired by the foraging behavior of ant colonies. This process led to the definition of the general framework for optimization called *ant colony optimization* (ACO), which has been published in a journal (Dorigo, Di Caro, & Gambardella, 1999), a book chapter and a conference proceedings (Dorigo & Di Caro, 1999a). More recently, this work has been further expanded, leading to the definition of the *ant programming* class of algorithms (Birattari, Di Caro, & Dorigo, 2000), that provides ACO with a more formal language and clarifies its relationships with the frameworks of optimal control and reinforcement learning. The research on ACO provided the basis for the development of the algorithms in this thesis. The brief review of the work on ACO given in the Introduction Chapter in based on the above papers.

The work described in this DEA thesis is at the same time an updated revision of previous work and a presentation of a new algorithm in the perspective of a novel, unifying, view. Both ACO and the previously developed algorithms are seen in their historical and conceptual development, ending with the definition of AntNet++, a new general multi-agent architecture for distributed learning and control in networks. AntNet++ is still undergoing development and testing. It will be applied in the context of networks providing quality-of-service (QoS), which represents one of the most challenging and, at the same time, "hot" fields of research and application in networks. Our personal target (and hope) is that the research on AntNet++, together with an expanded version of the contents presented in this DEA thesis, will provide sufficient material to compile a doctoral thesis in a near future.

This thesis is the outgrowth of almost four years of research. During this period several people, in different ways, gave a contribution to make this thesis happening. It is impossible to adequately acknowledge all of them. Anyway, it is impossible not to mention about all the members of Iridia, no one excluded, who have been at the same time stimulating colleagues and delightful friends. In particular, my supervisor Marco Dorigo, who has been an exceptional scientific mentor and who teached me (or at least, tried to teach me) the most important things, those that are not written on any book but that are essentials to be a researcher. And Mauro Birattari, a dear friend from whom I could learn so many things, concerning the most different fields, that I would probably need another life if I had to learn all those things by myself! From Italy, my old dear friend Silvana Valensin, always close in the most difficult moments and always open to insightful scientific discussions that have changed my way of thinking about so many aspects that gave a contribution to this thesis. I am Italian, so I cannot forget “La mamma”, my mother, to close the list of people who have given a really special contribution to make this thesis happening, and to make my life more enjoyable. Grazie!

Part of the work in this thesis was supported by a Madame Curie Fellowship (CEC-TMR Contract N. ERBFMBICT 961153) awarded to applicant by the European Union.

Contents

Chapter 1

Introduction

The demand and supply of telecommunication services is a worldwide phenomenon showing an impressive exponential growth. Internet, mobile phones, local area networks, satellite services, cable TV, just to mention few significant examples, already have and are continually changing the way we communicate among us, the way we receive, discover and store information, the way we control external devices. Of course, the business associated with telecommunications is of huge proportions and the telecommunication industry is incredibly active in the production of more and more powerful and flexible communication devices. This scenario urges the development of new techniques for network control and management. Techniques suitable for the characteristics of the ever new technologies and able to exploit them at the best. To offer the widest possible set of services to billions of customers willingful to pay to explore the new possibilities given by the era of telecommunications.

One of the main aspect of the world of nowadays telecommunications is the really big number of the possible different types/classes of devices, networks and services. "Just" restricting the focus on cabled networks, we can easily list so many popular different network technologies and/or protocols like Ethernet, ATM, FDDI, IP, Aloha, just to mention a very few. Moreover, on top of these network and basic protocols, several different services can be offered, like best-effort traffic, packet datagrams, circuit-oriented, guaranteed quality, fair-share best-effort, etc. Building a control and management system good for any possible context is practically impossible and ineffective. Nevertheless, it is possible to devise some open and modular architectures that can be used over a wide range of situations once situation-specific components can be easily added and/or modified. Given that in general in networks are always present several interplaying physical and software components and layers, it results rather natural to think about the networks (and about their systems for management and control) in terms of *distributed systems of multiple interacting agents* (Weiss, 1999; Stone & Veloso, 2000). Actually, agent technology is receiving an ever increasing attention from telecommunication system engineers, researchers and practitioners. Agent modeling account in a straightforward way for the modularity of network components and, more importantly, they overcome the typical client-server model of communication since they can carry their own execution code and they can be used as autonomous component of a global distributed and fault-tolerant system (among a plethora of ever increasing works, see for example (Hayzelden & Bigham, 1999; Kotz & Gray, 1999; Gray, Cybenko, Kotz, & Rus, 2001; Tintin & Lee, 1999; Küpper & Park, 1998; Vigna, 1998)). Probably, in the scenario of the next future networks will be completely managed and controlled by *societies of agents* (Shoham & Tennenholtz, 1995; Moulin, 1998) that will move, act, interact, adapt, replicate, die, ... This scenario will be more and more close to reality as long as networks will change to really be *active networks* (Tennenhouse, Smith, Sincoskie, Wetherall, & Minden, 1997), where packets will be able to carry their own execution or description code and all the network nodes will be able to perform, *as normal status of operations*, customized computations on the packets passing through them. Nowadays networks are more like a collection of high speed switching boxes, but in the future those boxes will be replaced by "network computers" and the whole network will appear as a huge multiprocessor system where societies of intelligent agents will support all our actions.

The work of this thesis is ultimately addressed at the above issue of defining an open multi-agent architecture for network control and management. In particular, we focus on the routing component of network control. Routing is at the core of any network control system, determining, in conjunction with the admission, flow, and congestion control components, the overall network performance in terms of both quality and quantity of delivered service.

Routing refers to the distributed activity of building and using *routing tables*, one for each node in the network, which tell incoming data packets which outgoing link to use to continue their travel towards the destination node. Routing protocols and policies have to accommodate conflicting objectives and constraints imposed by the different network technology and wishes of the customers (Sandick & Crawley, 1997; The ATM Forum, 1996). Here we "restrict" our attention to generic networks providing (possibly both) best-effort services in connection-less settings (e.g., Internet) and quality-of-service (QoS) in connection oriented settings (e.g., ATM).[1] While QoS and best-effort cover much of the field of possible services, we keep anyway the discussion and the proposed algorithms at a quite general level, without entering into details for any specific type of network technology or protocol.

In the above perspective of developing an open multi-agent architecture for adaptive routing in (hopefully active) best-effort and QoS networks, we describe in this thesis a family of three rather general algorithms. We call this family the *AntNet family*, from the name of the first and quite successful instance whose development started in 1997. The "AntNet family" includes three main algorithms: *AntNet* itself (Di Caro & Dorigo, 1998b), *AntNet-FA* (Di Caro & Dorigo, 1998f) and *AntNet++*, and two other algorithms, *AntNet-SELA* (Di Caro & Vasilakos, 2000) and *AntNet-FairShare* (Di Caro & Dorigo, 1998e) which are mentioned but not described in detail in this thesis. The term family is somehow justified by the fact that AntNet-FA is a version of AntNet where one single, but rather important aspect has been modified/optimized, while AntNet++, still possessing the core structure of AntNet-FA, is a much more general architecture with important additional adaptive components. Moreover, AntNet has been specifically designed for best-effort connection-less networks, AntNet-FA's modification has been made also to open AntNet to connection-oriented situations, AntNet-FairShare adapted AntNet-FA to handle best-effort connection-oriented traffic with fair-share allocations (Ma, Steenkiste, & Zhang, 1996), AntNet-SELA considered QoS traffic in ATM networks, and, finally, AntNet++ has been designed to be used in current/forthcoming networks where different types of services and traffic characteristics will necessarily coexist.

Much of the focus of this thesis (in terms of both experimental results and discussions) is on AntNet, given that it provided the solid roots for all the further developments. AntNet-FA is a basically a "smart" improvement over AntNet, therefore most of the considerations made for AntNet equally apply to AntNet-FA, while, in terms of performance, AntNet-FA always showed a monotonic improvement over the AntNet performance. A bit different is the situation for AntNet++, that is still undergoing a phase of development, testing and research. Both the design and implementation phases have been completed but further work would be necessary before assessing sound experimental and/or theoretical results being really innovative and competitive. Because of this fact, much of the discussion about AntNet++'s characteristics is suitably mixed with its description, while no experimental data are reported. Actually the completion of the research work on AntNet++ represents one of the most important achievements that have to be reached in the perspective of giving to this research a level of significance that could hopefully lead to a doctoral thesis in a near future.

[1]The literature on networks is very extensive, we suggest here the three references (Bertsekas & Gallager, 1992; Steenstrup, 1995; Walrand & Varaiya, 1996; Ma, 1998) that cover extensively and in depth the main concepts and algorithms for routing in the cases we are considering.

Trying to capture the characteristics of AntNet in a definition, it can be said that AntNet is a mobile-agent-based, online Monte Carlo technique based on previous work on algorithms inspired by the foraging behavior of ant colonies and, more generally, by the notion of *stigmergy* (Grassé, 1959). Therefore, AntNet comes from a recipe mixing together stochastic sampling, multi-agency, and examples of collective behaviors regulated by foraging strategies and stigmergic feedback from the ethological world. These two last aspects deserves some additional explanations. Stigmergy is the indirect asynchronous communication taking place among the individuals in a population through modifications induced in the environment they act upon. In the special case of networks, the network nodes where AntNet's agents read and write information, represent the environment that mediates a collective stigmergic behavior (see the paper (Dorigo et al., 1999) and the other papers in the same journal issue for an insightful discuss on stigmergy). On the other hand, algorithms inspired by the specific foraging behavior of ant colonies find their common framework in the *ant colony optimization* (ACO) metaheuristic, which provides the structural basis to AntNet. In ACO a set of (possibly computationally light) agents, called *ants*, collectively solve the problem under consideration through: (i) an *iterative process* of concurrent generation of feasible solutions to the problem at hand, where, (ii) at each iteration each ant builds a (possibly different) solution through (iii) a multi-stage decision process, where at each decision step a new component is selected to be added to the solution by a *stochastic greedy policy* parametrized by some statistical information (called *pheromone trail*) "local" to the decision point, (iv) the built solutions are evaluated and the results of the evaluation are used to modify the values of the pheromone estimates to bias in turn the generation process of the ants in the subsequent iterations. Some more details about ACO will be given in the next subsection of this Introduction but it is already immediately evident that this process well suits the characteristics of typical network environments once we think at the nodes as the above decision points, at the network routing tables as the local pheromone information, and at the whole network as the topological space where the ACO's ants concurrently and asynchronously move on to generate heuristic solutions. Where a candidate "solution" in this context is just a feasible path connecting two nodes, as sampled by the ant. Actually a path connecting two nodes is only a "component of the solution" of the whole routing problem, which requires the specification of a path for each of the possible pairs of nodes in the network. As anticipated before, AntNet follows this basic scheme of the ACO framework, introducing some suitable modification to account for the specificity of network problems with respect to the combinatorial optimization problems to whom ACO was initially addressed to.

AntNet is described with abundance of details in the whole Section 2.1.1, while its performance have been extensively studied with the results reported in Chapter 3. In the context of networks it is usually quite hard to analyze the performance of an algorithm from an analytical point of view, considered the huge possible variability of the traffic patters and of the the topological, and technological characteristics of the networks. Therefore, the to ensure a meaningful validation of the performance of an algorithm it is customary to set up a realistic and well differentiated simulation environment for what concern network characteristics, communications protocol and traffic patterns. This is what we have done to give necessary soundness to our results. In particular, we considered IP-like (Internet Protocol) datagram networks mapping both real-world and artificially generated topologies, with the number of nodes ranging from 8 to 150. On each network several paradigmatic temporal and spatial traffic distributions have been simulated and the performance of the algorithms have been measured in terms of packet delay and network throughput. AntNet was designed to be an algorithm to be potentially used in real networks, therefore we we report on its behavior as compared to some of the most known static and adaptive state-of-the-art routing algorithms, both vector-distance and link-state (see Appendix A), and to some recently introduced algorithms based on machine learning techniques.

AntNet shows the best performance and the most stable behavior for all the considered situations with respect to the considered competitors. In many experiments its superiority is striking. On the other side, AntNet-FA, whose behavior has been tested only in a subset of all the experiments, always reported better performance than AntNet. The improvement increasing with the size of the network. We discuss the results and the main properties of our algorithms, as compared with the competitor algorithms.

In spite of the more than encouraging performance showed by AntNet, and the even better performance of AntNet-FA, these two algorithms are missing something. First, they are missing some important adaptive components "necessary" in a dynamic and distributed environment as that of networks. Second, AntNet and AntNet-FA do not possess an architecture that would really match the characteristics and requirements of current and future networks. As it will be explained with many details in Sections 2.3.1 2.3.2, AntNet and AntNet-FA are somehow too constrained by the characteristics of ACO, their model of origin, thought for combinatorial optimization, to accommodate the very specific needs of network environments quickly discussed at the beginning of this Introduction. It is in this perspective that AntNet++ has been developed, rooted in AntNet, but going beyond. AntNet++ is an heterogenous society of static and mobile agents. The static agents are reinforcement learning agents, called *node managers*, each managing and controlling the activities of a single network node. The mobile agents are very similar to the AntNet's ants, but this time they play the role of *active perception* and *effector* agents under the explicit control of the node managers. AntNet++ is a distributed learning and adaptive control system whose single components are designed to match the characteristics of network problems and to be open at the utilization in active networks and/or networks where best-effort and QoS traffic could arbitrarily be mixed together.

While part of the material presented in this thesis has already been published (especially for what concern AntNet), here we describe both an updated revision of such material and a novel unifying view. The three described algorithms are seen in their historical and conceptual development, starting from ACO and "ending" in the definition of a general multi-agent architecture for distributed learning and control. We made the effort of putting all the pieces together, trying to have a clearer view of what is missing and what can be seen under a more general perspective, for this family of routing algorithms. This effort is also complemented by the parallel work (Birattari et al., 2000) where ACO itself is considered in a more general context and its relationships with the frameworks of optimal control and reinforcement learning are deeply investigated.

Summarizing, this thesis is organized as follows. In the following two sections some introductory background on ACO and routing is provided. More precisely, in Section 1.1 a quick overview on the ACO framework is provided, while in Section 1.2 are reported some definitions, taxonomy and characteristics of the routing problems. In Section 1.2.3 we describe the model of communication network we have used in our experiments, to make clear on the base of which assumptions are we operating.
Chapter 2 describes the AntNet family of algorithms: Section 2.1.1 describes AntNet in detail, Section 2.2 shows the characteristics of AntNet-FA, while Section 2.3 reports the description and the discussion of the characteristics of AntNet++.
Chapter 3 reports about experimental settings and results: the algorithms which we used for comparison are described in Section 3.1, while in Section 3.2 the experimental settings are described in terms of traffic, networks and algorithm parameters, Section 3.3 reports several experimental results for both AntNet and AntNet-FA.
In Chapter 4 we discuss these results for AntNet and try to explain its superior performance. While the discussion is specifically referred only to AntNet, most of the results apply also to AntNet-FA and to AntNet++, since basically they incorporate new additional features on top of the AntNet's ones, without disrupting the properties of these latters.
Finally, in Chapter 5, we discuss related work, and in Chapter 6, we draw some conclusions and outline directions for future research.
Appendix A gives some additional information on shortest path routing algorithms and concepts, while in Appendix B a brief history of the routing algorithms in the Internet is reported.

1.1 A quick look at Ant Colony Optimization (ACO)

Colonies of social insects can exhibit an amazing variety of rather complex behaviors and have captured, since ever, the interest of biologists and entomologists. More recently, computer scientists have found in the study of social insects behavior a source of inspiration for the design and implementation of novel distributed multi-agent algorithms. In particular, the study of ant colonies behavior turned out to be very fruitful, giving raise to a completely novel field of research, now broadly known as *ant algorithms* (see (Bonabeau, Dorigo, & Théraulaz, 1999; Bonabeau, Dorigo, & Theraulaz, 2000) for overviews on algorithms inspired by swarms and colonies of insects, and (Bonabeau et al., 2000) and the other papers in the same journal issue for a review more specific on ant algorithms).
The ants' feature that attracted the most the attention of computer scientists has been their ability to discover shortest paths (Goss, Aron, Deneubourg, & Pasteels, 1989; Beckers, Deneubourg, & Goss, 1992) between the colony's nest and sources of food in the environment. This effect emerges, at the global level of the colony, by means of a simple auto-catalytic mechanism based on the trail of a chemical substance (called *pheromone*) deposited by the ants while walking in search of food. This pheromone-based mechanism of ant colonies inspired the seminal work of Dorigo and his co-workers who in 1991 implemented the probably first algorithm based on this mechanism (Dorigo, Maniezzo, & Colorni, 1991; Dorigo, 1992; Dorigo, Maniezzo, & Colorni, 1996; Dorigo & Gambardella, 1997). Their algorithm, called Ant System, was applied to the traveling salesman problem (TSP). This first implementation, even if rather limited in its performance, stimulated the interest of other researchers, leading to a constant growth in the number of researchers getting inspiration from the initial Ant System and developing algorithms showing state-of-the-art performance in the heuristic solution of combinatorial optimization and network problems.[2]

More recently, the Ant Colony Optimization (ACO) metaheuristic (Dorigo et al., 1999; Dorigo & Di Caro, 1999b, 1999a) has been defined as a unifying *a posteriori* general framework that accounts for all the features common to the wide set of ten-years algorithms that have, more or less directly, inspired by the original work of Dorigo and co-workers. Actually, ACO has been further abstracted and generalized by the work of Birattari, Di Caro and Dorigo (2000) , who introduced the more formal framework of *ant programming*, which bridges the terminological gap between ACO and the fields of optimal control and reinforcement learning (Sutton & Barto, 1998), with the final goal of exploiting the understanding gained in those fields for a deeper theoretical analysis of ACO.

We already briefly introduced the basic mechanisms characterizing the ACO. Actually many different readings can be given to the ACO's behavior. Here we do not want to give the mathematical details that the interested reader can found in the ACO's cited papers. Therefore, the probably most expressive way of describing the basic characteristics of ACO is by using a graphical model: ACO is a class of algorithms based on the concurrent iterative generation and evaluation of paths on a weighted graph which conveniently encodes the combinatorial problem at hand. The graph is such that each solution of the combinatorial problem can be put in correspondence to a path on the graph. The weights associated to the edges of the graph are such that the cost of a path, i.e. the sum of the weights of its composing edges, can be directly put in relation with the cost for the associated solution in the combinatorial problem. In this sense, the goal of ant colony optimization is to find a path of minimum cost. To this end, a number of paths are generated in a Monte Carlo fashion, and the cost of such paths is used to bias the generation of further paths. This process is iterated with the aim of gathering more and more information on the graph to eventually produce a path of minimum cost.

ACO was originated from the observation of the foraging behavior of real ant colonies, therefore, the just described algorithm can be conveniently visualized in terms of a *metaphor*

[2]The interested reader can find in (Dorigo et al., 1999; Dorigo & Di Caro, 1999b; Dorigo, Di Caro, & (Editors), 2000) lists and descriptions of implementations of ACO for a variety of combinatorial optimization and network problems like, among others, traveling salesman, quadratic assignment, graph coloring, frequency allocation, adaptive routing.

in which the generation of a path is described as the walk of an ant that, at each node, stochastically selects the following one on the basis of local information called pheromone trail. In turn, the pheromone trail is modified by the ants themselves in order to bias the generation of future paths toward better solutions.

1.2 Routing: Definition and characteristics

Routing in distributed systems can be characterized as follows. Let $G = (V, E)$ be a directed weighted graph, where each node in the set V represents a processing/queuing and/or forwarding unit and each edge is a transmission system. The main task of a routing algorithm is to direct data flow from source to destination nodes maximizing network performance. In the problems we are interested in, the data flow is not statically assigned and it follows a stochastic profile that is very hard to model.

In the specific case of communications networks (Bertsekas & Gallager, 1992; Steenstrup, 1995), the routing algorithm has to manage a set of basic functionalities and it tightly interacts with the congestion and admission control algorithms, with the links' queuing policy, and with the user-generated traffic. The core of the routing functions is (i) the acquisition, organization and distribution of information about user-generated traffic and network states, (ii) the use of this information to generate feasible routes maximizing the performance objectives, and (iii) the forwarding of user traffic along the selected routes.

The way the above three functionalities are implemented strongly depends on the underlying network switching and transmission technology, and on the features of the other interacting software layers. Concerning point (iii), two main forwarding paradigms are in use: *circuit* and *packet-switching* (also indicated with the terms *connection-oriented* and *connection-less*). In the circuit-switching approach, a setup phase looks for and reserves the resources that will be assigned to each incoming session. In this case, all the data packets belonging to the same session will follow the same path. Routers are required to keep state information about active sessions. In the packet-switching approach, there is no reservation phase, no state information is maintained at routers and data packets can follow different paths. In each intermediate node an autonomous decision is taken concerning the node's outgoing link that has to be used to forward the data packet toward its destination.

The work on AntNet, the main focus of this thesis, is addressed at routing in connection-less networks, while AntNet++ (as also AntNet-FairShare and AntNet-SELA do) considers the more general case of networks with mixed characteristics.

A further distinction in network types is made according to the nature of the delivered traffic. Networks like Internet are said providing *best-effort* traffic, in the sense that there is no guarantee on the quality of the delivered performance. For example, according to the traffic load, downloading via ftp the same big file in different moments can show a big variability in the completion time and the user has no way to "complain" or to impose some constraints on the downloading time. Moreover, some applications, by chance (in the sense of a favorable routing situation) can occupy much more bandwidth than others, and, again, there is no internal or external way of avoiding these phenomena. On the contrary, in connection-oriented networks where for example a *fair-share* routing scheme is implemented, the routing system is such that it tries to to make the network utilization equally distributed over all the already active and the new coming applications. Creating a "fair sharing" of the resources.

At the other extreme of the best-effort behavior, in networks providing *quality-of-service* (QoS) the application (the user) can specify the resource it needs and the underlying system tries to reserve them, or it is forced to refuse the application, then likely losing money. Actually all the new data and (interactive) multimedia network applications require some form of hard or soft QoS, in terms, for example, of guaranteed end-to-end delay, bandwidth, delay jitter, loss rate. QoS technologies provide the tools to deliver mission critical business over the networks. This fact is opening a wide range of different possibilities in network utilization and pricing. Actually this perspective has attracted the interest of a big number of companies and governmental institutions, making the research in the field of QoS very active. The range of proposed solutions is quite wide, also because different type of networks (e.g., ATM, IP, wireless, frame relay) have different characteristics at the data, transport, network, middleware, etc., layers. In particular, research is very active for ATM and IP networks, where also international committees are working to define specifications, requirements and algorithms. Given the strategic importance of QoS, it is worth here to spend some more words, even if in this thesis we do not report experimental data for QoS situations but only the description of an algorithm, AntNet++, which has specific QoS components.

QoS routing is the first, essential step toward achieving QoS guarantees. QoS routing identifies and selects a path that meets the QoS constraints while providing an efficient utilization of the network resources. But this is only one aspect of the problem. In the general case, when a new application arrives and requires some specific network resources, the local connection admission control (CAC) component makes use of routing information to check if there is any path connecting the application end-points that meets the QoS requirements of the application. If one or more paths can be found the CAC evaluates the convenience (in terms of costs, benefits, forecasting of network status, etc.) of accepting or not the application. In the positive case, one (or more) of the QoS-feasible paths are chosen to route the application and the asked resources are reserved on them. In general, to provide and sustain QoS, the resource management system must consider resources availability and allocation and control policies. The system must compute/estimate the feasibility and convenience of allocating a new QoS application. At the same time, it must ensure that the contracted QoS is sustained, monitoring the network status and reallocating resources in response to anomalies or heavily unbalanced situations. From this picture it is quite clear that when QoS is involved, the situation becomes much more complex than the best-effort case. In particular the *metrics* that are used to score the whole network performance become much more complex and are made up by several, often conflicting components.

1.2.1 A broad taxonomy

A common feature of all the routing algorithms is the presence in every network node of a data structure, called *routing table*, holding all the information used by the algorithm to make the local forwarding decisions. The routing table is both a local database and a local model of the global network status. The type of information it contains and the way this information is used and updated strongly depends on the algorithm's characteristics. A broad classification of routing algorithms is the following:

- centralized *versus* distributed;
- static *versus* adaptive.

In *centralized* algorithms, a main controller is responsible for updating all the node's routing tables and/or to make every routing decision. Centralized algorithms can be used only in particular cases and for small networks. In general, the delays necessary to gather information about the network status and to broadcast the decisions/updates make them infeasible in practice. Moreover, centralized systems are not fault-tolerant. In this work, we will consider exclusively distributed routing.

In *distributed* routing systems, the computation of routes is shared among the network nodes, which exchange the necessary information. The distributed paradigm is currently used in the majority of network systems.

In *static* (or *oblivious*) routing systems, 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 only 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. 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 connection-less than for connection-oriented networks (Bertsekas & Gallager, 1992).

Another interesting way of looking at routing algorithms is from an optimization perspective. In this case the main paradigms are:

- minimal routing *versus* non-minimal routing;
- optimal routing *versus* shortest path routing.

Minimal routers allow packets to choose only minimal cost paths, while *non-minimal* algorithms allow choices among all the available paths following some heuristic strategies (Bolding, Fulgham, & Snyder, 1994).

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) (Bertsekas & Gallager, 1992).

Shortest path routing has a source-destination pair perspective: there is no global cost function to optimize. Its objective is to determine the shortest path (minimum cost) between two nodes, where the link costs are computed (statically or adaptively) following some statistical description of the link states. This strategy is based on individual rather than group rationality (Wang & Crowcroft, 1992). Considering the different content stored in each routing table, shortest path algorithms can be further subdivided into two classes called *distance-vector* and *link-state* (Steenstrup, 1995).

Optimal routing is static (it can be seen as the solution of a multicommodity flow problem) and requires the knowledge of all the traffic characteristics. Shortest paths algorithms are more flexible, they don't require a priori knowledge about the traffic patterns and they are the most widely used routing algorithms.

In Appendix A a more detailed description of the properties of optimal and shortest path routing algorithms is reported.

Algorithms in the AntNet family actually share the same optimization perspective as (minimal or non-minimal) shortest path algorithms but not their usual implementation paradigms (as depicted in the Appendix).

1.2.2 Main characteristics of routing problems

The main characteristics of most of the routing problems in communications networks can be briefly summarized as follows:

- *Intrinsically distributed* with strong *real-time* constraints: in fact, the database and the decision system are completely distributed over all the network nodes, and failures and status information propagation delays are not negligible with respect to the user's traffic patterns. It is impossible to get complete and up-to-date knowledge of the distributed state, that remains hidden. At each decision node, the routing algorithm can only make use of local, up-to-date information, and of non-local, delayed information coming from the other nodes.
- *Stochastic and time-varying*: the session arrival and data generation process is, in the general case, non-stationary and stochastic. Moreover, this stochastic process interacts recursively with the routing decisions making it infeasible to build a working model of the whole system (to be used for example in a dynamic programming framework).
- *Multi-objective*: several conflicting *performance measures* are usually taken into account. The most common are *throughput* (bit/sec) and *average packet delay* (sec). The former measures the quantity of service that the network has been able to offer in a certain amount of time (amount of correctly delivered bits per time unit), while the latter defines the quality of service produced at the same time. Citing Bertsekas and Gallager (1992), page 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". Other performance measures consider the impact of the routing algorithm on the network resources in terms of memory, bandwidth and computation, and the algorithm simplicity, flexibility, etc. Moreover, as previously already emphasized, in the special case of QoS networks the number of objective to take into account (or, equivalently, constraints) grows dramatically, making the situation much harder to be managed in a efficient.

- *Multi-constraint*: constraints are imposed by the underlying network technology, the network services provided and the user services requested. In general, users ask for low-cost, high-quality, reliable, distributed multimedia services available across heterogeneous static and mobile networks. Evaluating technological and commercial factors, network builders and service providers try to accommodate these requests while maximizing some profit criteria. Moreover, a high level of *fault-tolerance* and *reliability* is requested in modern high-speed networks, where user sessions can formulate precise requests for network resources. In this case, once the session has been accepted, the system should be able to guarantee that the session gets the resources it needs, under any recoverable fault event.

It is interesting to note that the above characteristics make the problem of routing belong to the class of *reinforcement learning* (Sutton & Barto, 1998) problems with imperfect information (Hauskrecht, 1997; Kaebling, Littman, & Cassandra, 1998; McCallum, 1995; Chrisman, 1992; Bertsekas & Tsitsiklis, 1996). A distributed system of agents, the components of the routing algorithm in each node, determine a continual and online learning of the best routing table values with respect to network's performance criteria. An exact measure of evaluation that scores forwarding decisions is not available, neither online nor in the form of a training set. Moreover, because of the distributed nature of the problem and of its constraints, the complete state of the network is hidden to each agent.

1.2.3 The communication network model

In this thesis we focus on networks with an IP-like network layer (in the ISO-OSI terminology) and a very simple transport layer. The assumed topology is irregular, and both connection-less and connection-oriented schemes are in principle admitted. Anyway, in the following we describe the situation for the connection-less case. The connection-oriented is a quite straightforward modification that requires to keep per-application or per-destination state information at the nodes. In particular, we focus on a model of wide-area networks (WAN). In these cases, *hierarchical* organization schemes are adopted.[3] Roughly speaking, sub-networks are seen as single host nodes connected to interface nodes called gateways. Gateways perform fairly sophisticated network layer tasks, including routing. Groups of gateways, connected by an arbitrary topology, define logical areas. Inside each area, all the gateways are at the same hierarchical level and "flat" routing is performed among them. Areas communicate only by means of area border gateways. In this way, the computational complexity of the routing problem, as seen by each gateway, is much reduced (e.g., in the Internet, OSPF areas typically group 10 to 300 gateways), while the complexity of the design and management of the routing protocol is much increased.

The instance of our communication network is mapped on a directed weighted graph with N processing/forwarding nodes. All the links are viewed as bit pipes characterized by a bandwidth (bit/sec) and a transmission delay (sec), and are accessed following a statistical multiplexing scheme. For this purpose, every node, of type store-and-forward, holds a buffer space where the incoming and the outgoing 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

[3]A hierarchical structure is adopted on the Internet, organized in hierarchical Autonomous Systems and multiple routing areas inside each Autonomous System (Moy, 1998).

packets in the same class have the same priority, so they are queued and served on the basis of a first-in-first-out policy, but routing packets have a greater priority than data packets. The workload is defined in terms of applications whose arrival rate is dictated by a selected probabilistic model. By application (or session, or connection in the following), 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. We didn't make any distinction among nodes, they act at the same time as hosts (session end-points) and gateways/routers (forwarding elements). The adopted workload model incorporates a simple flow control mechanism implemented by using a fixed production window for the session's packets generation. The window determines the maximum number of data packets waiting to be sent. Once sent, a packet is considered to be acknowledged. This means that the transport layer neither manages error control, nor packet sequencing, nor acknowledgements and retransmissions.[4]

For each incoming packet, the node's routing component uses the information stored in the local routing table to assign the outgoing link to be used to forward the packet toward its target node. When the link resources are available, they are reserved and the transfer is set up. The time it takes to move a packet from one node to a neighboring one depends on the packet size and on the link transmission characteristics. If, on a packet's arrival, there is not enough buffer space to hold it, the packet is discarded. Otherwise, a service time is stochastically generated for the newly arrived packet. This time represents the delay between the packet arrival time and the time when it will be put in the buffer queue of the outgoing link the local routing component has selected for it.

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 its 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.

We end this section with some remarks concerning two features of the model.

First, we chose not to implement a "real" transport layer for a proper management of error, flow, and congestion control. In fact, each additional control component has a considerable impact on the network performance,[5] making very difficult to evaluate and to study the properties of each control algorithm without taking in consideration the complex way it interacts with all the other control components. Therefore, we chose to test the behavior of our algorithm and of its competitors in conditions such that the number of interacting components is minimal and the routing component can be evaluated in isolation, allowing a better understanding of its properties. To study routing in conjunction with error, flow and congestion control, all these components should be designed at the same time, to allow a good match among their characteristics to produce a synergetic effect.

Second, for what concerns the experimental part of this thesis we chose to work with connection-less and not with connection-oriented networks because connection-oriented schemes are mainly used in networks able to deliver QoS (Crawley, Nair, Rajagopalan, & Sandick, 1996), which is not the case, for example, of the current Internet, even if some suitable extension to the IP protocol are under active investigation. Anyway, the "problem" here is that analyzing the performance of a routing scheme in a generic QoS network can be quite hard. In fact, in this case, suitable admission control algorithms have to be introduced,

[4]This choice is the same as in the "Simple_Traffic" model in the MaRS network simulator (Alaettinoğlu, Shankar, Dussa-Zieger, & Matta, 1992). It can be seen as a very basic form of File Transfer Protocol (FTP).

[5]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) by changing from the Reno version to the Vegas version of the TCP (Peterson & Davie, 1996) (the current Internet Transport Control Protocol), and other authors even claimed improvements ranging from 40 to 70% (Brakmo, O'Malley, & Peterson, 1994).

taking into account for many economic and technological factors (Sandick & Crawley, 1997). As in the case of TCP, it would be quite hard to well separate the contribution of the admission control algorithm from that of the routing algorithm. Therefore, as a first step, we think that it is more reasonable to try to check the validity of a routing algorithm by reducing the number of components heavily influencing the network behavior. In any case, AntNet++ is supposed to be applied, and then tested, in a QoS situation in the near future.

Chapter 2

The AntNet family of multi-agent systems for adaptive routing

In this chapter we describe AntNet, AntNet-FA and AntNet++, the three multi-agents systems for adaptive routing that are the main subject of this thesis. The three algorithms have been developed over the time according to an incremental and hierarchical process. AntNet has been the first system implemented, designed as a specialization of the ACO paradigm to the solution of problems of adaptive routing in best-effort connection-less networks. Description and results for various AntNet's versions have been the object of several publications and conference presentations (Di Caro & Dorigo, 1998b), (Di Caro & Dorigo, 1997c, 1998a, 1998c, 1998d, 1998f) (Di Caro & Dorigo, 1997a, 1997b), (Di Caro & Dorigo, 1997a, 1997b). AntNet-FA improves the AntNet's performance by modifying an essential aspect in the way the ants move while build a path, and in the way the path itself is evaluated. AntNet-FA has been introduced in (Di Caro & Dorigo, 1998f), while in (Di Caro & Dorigo, 1998e) it has been suitably expanded to be used in connection-oriented networks with fair-share routing (AntNet-FairShare). AntNet++ is the latest, and still under development, routing system in the AntNet family. While its basic components are inherited by those of ACO in general and AntNet in particular, AntNet++ represents somehow a departure from the previous schemes. AntNet++ is an open multi-agent architecture designed to accommodate the specific characteristics of modularity, distribution, non-stationarity, flexibility of current and future network environments were multiple different services (QoS, best-effort, fair-share,etc.) and connectivity schemes are provided. Moreover, AntNet++ is more a distributed learning and control system than a "simple" routing algorithm. An earlier, specific implementation of AntNet++ for ATM networks, AntNet-SELA, has been reported in (Di Caro & Vasilakos, 2000), but basically AntNet++ is firstly presented in this thesis.

In spite of the fact that part of the material presented in this thesis has already been published, here we describe both an updated revision of such material and a novel unifying view. The three algorithms are not considered separately but are seen in their historical and conceptual development, starting from ACO and "ending" in the definition of a general multi-agent architecture for distributed learning and control.

2.1 Basic concepts behind AntNet

As already pointed out, the characteristics of the routing problem make it well suited to be solved by a mobile multi-agent approach (Weiss, 1999; Hayzelden & Bigham, 1999; Gray et al., 2001). This processing paradigm is a good match for the distributed and non-stationary (in topology and traffic patterns) nature of the problem, presents a high level of redundancy and fault-tolerance, and can handle multiple objectives and constraints in a flexible way.

AntNet, the ancestor of all the routing algorithms that we report in this thesis, is a mobile agents system showing some essential features of parallel replicated Monte Carlo systems (Streltsov & Vakili, 1996). AntNet takes inspiration from previous work on ACO algorithms for combinatorial optimization and, in particular, from the work (Schoonderwoerd, Holland, Bruten, & Rothkrantz, 1996; Schoonderwoerd, Holland, & Bruten, 1997) which represent a first application of ant-inspired algorithms to routing, in telephone networks. As described with some detail in the chapter of the Introduction, at the core of ACO the are: the use of repeated and concurrent simulations carried out by a population of artificial agents, called ants, to generate new solutions to the problem at hand, (ii) the use by the agents of stochastic local search to build the solutions in an incremental way, and (iii) the use of information collected during past simulations to direct future search for better solutions.

In the ant colony optimization approach, following an iterative process, each ant builds a solution by using two types of information locally accessible: problem-specific information (for example, distance among cities in a traveling salesman problem), and information added by ants during previous iterations of the algorithm. In fact, while building a solution, each ant collects information on the problem characteristics and on its own performance, and uses this information to modify the representation of the problem, as seen locally by the other ants. The representation of the problem is modified in such a way that information contained in past good solutions can be exploited to build new better solutions. This form of indirect communication mediated by the environment is called *stigmergy*, and is typical of social insects (Grassé, 1959).

In AntNet, we retain the core ideas of ACO and we apply them to solve in an adaptive way the routing problem in best-effort datagram networks.

Informally, the AntNet algorithm and its main characteristics can be summarized as follows.

- At regular intervals, and concurrently with the data traffic, from each network node mobile agents are asynchronously launched towards randomly selected destination nodes.
- Agents act concurrently and independently, and communicate in an indirect way, through the information they read and write locally to the nodes.
- Each agent searches for a minimum cost path joining its source and destination nodes.
- Each agent moves step-by-step towards its destination node. At each intermediate node a greedy stochastic policy is applied to choose the next node to move to. The policy makes use of (i) local agent-generated and maintained information, (ii) local problem-dependent heuristic information, and (iii) agent-private information.
- While moving, the agents collect information about the time length, the congestion status and the node identifiers of the followed path.
- Once they have arrived at the destination, the agents go back to their source nodes by moving along the same path as before but in the opposite direction.
- During this backward travel, local models of the network status and the local routing table of each visited node are modified by the agents as a function of the path they followed and of its goodness.
- Once they have returned to their source node, the agents die.

In the following subsections the above scheme is explained, all its components are explicated and discussed, and a more detailed description of the algorithm is given.

2.1.1 Algorithm description and main characteristics

AntNet is conveniently described in terms of two sets of *homogeneous mobile agents* (Stone & Veloso, 2000), called in the following *forward* and *backward ants*. Agents[1] in each set

[1]In the following, we will use interchangeably the terms ant and agent.

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. They can be broadly classified 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. Agents communicate in an indirect way, according to the stigmergy paradigm, through the information they concurrently read and write in two data structures stored in each network node k (see Figure 2.1):

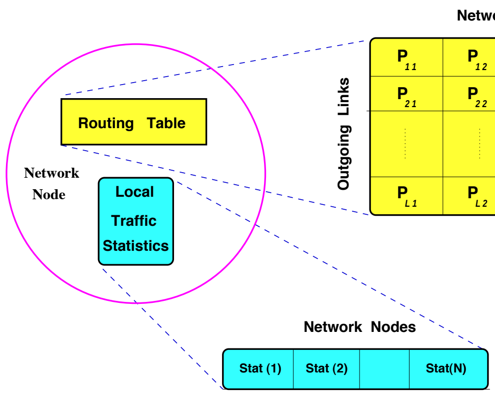


Figure 2.1: Node structures used by mobile agents in AntNet for the case of a node with L neighbors and a network with N nodes. The routing table is organized as in vector-distance algorithms, but the entries are probabilistic values. The structure containing statistics about the local traffic plays the role of a local adaptive model for the traffic toward each possible destination.

i) A routing table $\mathcal{T}_k$, organized as in vector-distance algorithms (see Appendix A), but with probabilistic entries. $\mathcal{T}_k$ defines the probabilistic routing policy currently adopted at node k: for each possible destination d and for each neighbor node n, $\mathcal{T}_k$ stores a probability value P_{nd} expressing the goodness (desirability), under the current network-wide routing policy, of choosing n as next node when the destination node is d:

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

ii) An array $\mathcal{M}_k(\mu_d, {\sigma_d}^2, \mathcal{W}_d)$, of data structures defining a simple parametric statistical model for the traffic distribution over the network as seen by the local node k. The model is adaptive and described by sample means and variances computed over the trip times experienced by the mobile agents, and by a moving observation window $\mathcal{W}_d$ used to store the best value W_{best_d} of the agents' trip time.
For each destination d in the network, an estimated mean and variance, μ_d and ${\sigma_d}^2$, give a representation of the expected time to go and of its stability. We used arithmetic, exponential and windowed strategies to compute the statistics. Changing strategy does not affect performance much, but we observed the best results using the exponential model:[2]

$$\begin{aligned} \mu_d &\leftarrow \mu_d + \eta(o_{k\to d} - \mu_d), \\ {\sigma_d}^2 &\leftarrow {\sigma_d}^2 + \eta((o_{k\to d} - \mu_d)^2 - {\sigma_d}^2), \end{aligned} \tag{2.1}$$

where $o_{k\to d}$ is the new observed agent's trip time from node k to destination d.[3]
The moving observation window $\mathcal{W}_d$ is used to compute the value W_{best_d} of the best

[2]This is the same model as used by the Jacobson/Karels algorithm to estimate retransmission timeouts in the Internet TCP(Peterson & Davie, 1996).

[3]The factor η weights the number of most recent samples that will really affect the average. The weight of the t_i-th sample used to estimate the value of μ_d after j samplings, with $j > i$, is: $\eta(1-\eta)^{j-i}$. In this way, for example, if $\eta = 0.1$, approximately only the latest 50 observations will really influence the estimate, for $\eta = 0.05$, the latest 100, and so on. Therefore, the number of effective observations is $\approx 5(1/\eta)$.

agents' trip time towards destination d as observed in the last w samples. After each new sample, w is incremented modulus $|\mathcal{W}|_{max}$, and $|\mathcal{W}|_{max}$ is the maximum allowed size of the observation window. The value W_{best_d} represents a short-term memory expressing a moving empirical lower bound of the estimate of the time to go to node d from the current node.

$\mathcal{T}$ and $\mathcal{M}$ can be seen as memories local to nodes capturing different aspects of the network dynamics. The model $\mathcal{M}$ maintains absolute distance/time estimates to all the nodes, while the routing table gives relative probabilistic goodness measures for each link-destination pair under the current routing policy implemented over all the network.

The AntNet algorithm is described as follows.

1. At regular intervals Δt from every network node s, a mobile agent (forward ant) $F_{s\to d}$ is launched toward a destination node d to discover a feasible, low-cost path to that node and to investigate the load status of the network. Forward ants share the same queues as data packets, so that they experience the same traffic loads. Destinations are locally selected according to the data traffic patterns generated by the local workload: if f_{sd} is a measure (in bits or in number of packets) of the data flow $s \to d$, then the probability of creating at node s a forward ant with node d as destination is

$$p_d = \frac{f_{sd}}{\sum_{d'=1}^{N} f_{sd'}}. \tag{2.2}$$

   In this way, ants adapt their exploration activity to the varying data traffic distribution.
2. While traveling toward their destination nodes, the agents keep memory of their paths and of the traffic conditions found. The identifier of every visited node k and the time elapsed since the launching time to arrive at this k-th node are pushed onto a memory stack $S_{s\to d}(k)$.
3. At each node k, each traveling agent headed towards its destination d selects the node n to move to choosing among the neighbors it did not already visit, or over all the neighbors in case all of them had been previously visited. The neighbor n is selected with a probability (goodness) P'_{nd} computed as the normalized sum of the probabilistic entry P_{nd} of the routing table with a heuristic correction factor l_n taking into account the state (the length) of the n-th link queue of the current node k:

$$P'_{nd} = \frac{P_{nd} + \alpha l_n}{1 + \alpha(|\mathcal{N}_k| - 1)}. \tag{2.3}$$

   The heuristic correction l_n is a [0,1] normalized value proportional to the length q_n (in bits waiting to be sent) of the queue of the link connecting the node k with its neighbor n:

$$l_n = 1 - \frac{q_n}{\sum_{n'=1}^{|\mathcal{N}_k|} q_{n'}}. \tag{2.4}$$

   The value of α weights the importance of the heuristic correction with respect to the probability values stored in the routing table. l_n reflects the instantaneous state of the node's queues, and assuming that the queue's consuming process is almost stationary or slowly varying, l_n gives a quantitative measure associated with the queue waiting time. The routing tables values, on the other hand, are the outcome of a continual learning process and capture both the current and the past status of the whole network as seen by the local node. Correcting these values with the values of l allows the system to be more "reactive", at the same time avoiding following all the network fluctuations. Agent's decisions are taken on the basis of a combination of a long-term learning process and an instantaneous heuristic prediction.

In all the experiments we ran, we observed that the introduced correction is a very effective mechanism. Depending on the characteristics of the problem, the best value to assign to the weight α can vary, but if α ranges between 0.2 and 0.5, performance doesn't change appreciably. For lower values, the effect of l is vanishing, while for higher values the resulting routing tables oscillate and, in both cases, performance degrades.

4. 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. If the cycle lasted longer than the lifetime of the ant before entering the cycle, (that is, if the cycle is greater than half the ant's age) the ant is destroyed. In fact, in this case the agent wasted a lot of time probably because of a wrong sequence of decisions and not because of congestion states. Therefore, the agent is carrying an old and misleading memory of the network state and it is counterproductive to use it to update the routing tables (see below).
5. When the destination node d is reached, the agent $F_{s\to d}$ generates another agent (backward ant) $B_{d\to s}$, transfers to it all of its memory, and dies.
6. The backward ant takes the same path as that of its corresponding forward ant, but in the opposite direction.[4] At each node k along the path it pops its stack $S_{s\to d}(k)$ to know the next hop node. Backward ants do not share the same link queues as data packets; they use higher priority queues, because their task is to quickly propagate to the routing tables the information accumulated by the forward ants.
7. Arriving at a node k coming from a neighbor node f, the backward ant updates the two main data structures of the node, the local model of the traffic $\mathcal{M}_k$ and the routing table $\mathcal{T}_k$, for all the entries corresponding to the (forward ant) destination node d. With some precautions, updates are performed also on the entries corresponding to every node $k' \in S_{k\to d}, k' \neq d$ on the "sub-paths" followed by ant $F_{s\to d}$ after visiting the current node k. In fact, if the elapsed trip time of a sub-path is statistically "good" (i.e., it is less than $\mu + I(\mu, \sigma)$, where I is an estimate of a confidence interval for μ), then the time value is used to update the corresponding statistics and the routing table. On the contrary, trip times of sub-paths not deemed good, in the same statistical sense as defined above, are not used because they don't give a correct idea of the time to go toward the sub-destination node. In fact, all the forward ant routing decisions were made only as a function of the destination node. In this perspective, sub-paths are side effects, and they are intrinsically sub-optimal because of the local variations in the traffic load (we can't reason with the same perspective as in dynamic programming, because of the non-stationarity of the problem representation). Obviously, in case of a good sub-path we can use it: the ant discovered, at zero cost, an additional good route. In the following two items the way $\mathcal{M}$ and $\mathcal{T}$ are updated is described with respect to a generic "destination" node $d' \in S_{k\to d}$.

   i) $\mathcal{M}_k$ is updated with the values stored in the stack memory $S_{s\to d}(k)$. The time elapsed to arrive (for the forward ant) to the destination node d' starting from the current node is used to update the mean and variance estimates, $\mu_{d'}$ and $\sigma_{d'}{}^2$, and the best value over the observation window $\mathcal{W}_{d'}$. In this way, a parametric model of the traveling time to destination d' is maintained. The mean value of this time and its dispersion can vary strongly, depending on the traffic conditions: a poor time (path) under low traffic load can be a very good one under heavy traffic load. The statistical model has to be able to capture this variability and to follow in a robust way the fluctuations of the traffic. This model plays a critical role in the routing table updating process (see item (ii) below). Therefore, we investigated several ways to build effective and computationally inexpensive models, as described in the following Section 2.1.2.

[4]This assumption requires that all the links in the network are bi-directional. In modern networks this is a reasonable assumption.

ii) The routing table $\mathcal{T}_k$ is changed by incrementing the probability $P_{fd'}$ (i.e., the probability of choosing neighbor f when destination is d') and decrementing, by normalization, the other probabilities $P_{nd'}$. The amount of the variation in the probabilities depends on a measure of goodness we associate with the trip time $T_{k \to d'}$ experienced by the forward ant, and is given below. This time represents the only available explicit feedback signal to score paths. It gives a clear indication about the goodness r 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 (the forward ants share the same queues as data packets).

The time measure T, composed by all the sub-paths elapsed times, cannot be associated with an exact error measure, given that we don't know the "optimal" trip times, which depend on the whole network load status.[5] Therefore, T can only be used as a reinforcement signal. This gives rise to a credit assignment problem typical of the reinforcement learning field (Bertsekas & Tsitsiklis, 1996; Kaelbling, Littman, & Moore, 1996). We define the reinforcement $r \equiv r(T, \mathcal{M}_k)$ to be a function of the goodness of the observed trip time as estimated on the basis of the local traffic model. r is a dimensionless value, $r \in (0, 1]$, used by the current node k as a positive reinforcement for the node f the backward ant $B_{d \to s}$ comes from. r takes into account some average of the so far observed values and of their dispersion to score the goodness of the trip time T, such that the smaller T is, the higher r is (the exact definition of r is discussed in the next subsection). The probability $P_{fd'}$ is increased by the reinforcement value as follows:

$$P_{fd'} \leftarrow P_{fd'} + r(1 - P_{fd'}). \tag{2.5}$$

In this way, the probability $P_{fd'}$ will be increased by a value proportional to the reinforcement received and to the previous value of the node probability (that is, given a same reinforcement, small probability values are increased proportionally more than big probability values, favoring in this way a quick exploitation of new, and good, discovered paths).

Probabilities $P_{nd'}$ for destination d' of the other neighboring nodes n implicitly receive a negative reinforcement by normalization. That is, their values are reduced so that the sum of probabilities will still be 1:

$$P_{nd'} \leftarrow P_{nd'} - rP_{nd'}, \quad n \in \mathcal{N}_k,\ n \neq f. \tag{2.6}$$

It is important to remark that every discovered path receives a positive reinforcement in its selection probability, and the reinforcement is (in general) a non-linear function of the goodness of the path, as estimated using the associated trip time. In this way, not only the (explicit) assigned value r plays a role, but also the (implicit) ant's arrival rate. This strategy is based on trusting paths that receive either high reinforcements, independent of their frequency, or low and frequent reinforcements. In fact, for any traffic load condition, a path receives one or more high reinforcements only if it is much better than previously explored paths. On the other hand, during a transient phase after a sudden increase in network load all paths will likely have high traversing times with respect to those learned by the model $\mathcal{M}$ in the preceding, low congestion, situation. Therefore, in this case good paths can only be differentiated by the frequency of ants' arrivals.

Assigning always a positive, but low, reinforcement value in the case of paths with high traversal time allows the implementation of the above mechanism based on

[5] When the network is in a congested state, all the trip times will score poorly with respect to the times observed in low load situations. Nevertheless, a path with a high trip time should be scored as a good path if its trip time is significantly lower than the other trip times observed in the same congested situation.

the frequency of the reinforcements, while, at the same time, avoids giving excessive credit to paths with high traversal time due to their poor quality.

The use of probabilistic entries is very specific to AntNet and we observed it to be effective, improving the performance, in some cases, even by 30%-40%. Routing tables are used in a probabilistic way not only by the ants but also by the data packets. This has been observed to improve AntNet performance, which means that the way the routing tables are built in AntNet is well matched with a probabilistic distribution of the data packets over all the good paths. Data packets are prevented from choosing links with very low probability by re-mapping the $\mathcal{T}$'s entries by means of a power function $f(p) = p^{\alpha}, \alpha > 1$, which emphasizes high probability values and reduces lower ones (in our experiments we set α to 1.2).

Figure 2.2 gives a high-level description of the algorithm in pseudo-code, while Figure 2.3 illustrates a simple example of the algorithm behavior. A detailed discussion of the characteristics of the algorithm is postponed to Section 4, after the performance of the algorithm has been analyzed with respect to a set of competitor algorithms. In this way, the characteristics of AntNet can be meaningfully evaluated and compared to those of other state-of-the-art algorithms.

2.1.2 How to score the goodness of the ant trip time

The reinforcement r is a critical quantity that has to be assigned by considering three main aspects: (i) paths should receive an increment in their selection probability proportional to their goodness, (ii) the goodness is a relative measure, which depends on the traffic conditions, that can be estimated by means of the model $\mathcal{M}$, and (iii) it is important not to follow all the traffic fluctuations. This last aspect is particularly important. Uncontrolled oscillations in the routing tables are one of the main problems in shortest paths routing (Wang & Crowcroft, 1992). It is very important to be able to set the best trade-off between stability and adaptivity.

We investigated several ways to assign the r values trying to take into account the above three requirements:

- The simplest way is to set $r = \textit{constant}$: independently of the ant's "experiment outcomes", the discovered paths are all rewarded in the same way. In this simple but meaningful case, what is at work is the implicit reinforcement mechanism due to the differentiation in the ant arrival rates. Ants traveling along faster paths will arrive at a higher rate than other ants, hence their paths will receive a higher cumulative reward.[6] The obvious problem of this approach lies in the fact that, although ants following longer paths arrive delayed, they will nevertheless have the same effect on the routing tables as the ants who followed shorter paths.

  In the experiments we ran with this strategy, the algorithm showed moderately good performance. These results suggest that the "implicit" component of the algorithm, based on the ant arrival rate, plays a very important role. Of course, to compete with state-of-the-art algorithms, the available information about path costs has to be used.
- More elaborate approaches define r as a function of the ant's trip time T, and of the parameters of the local statistical model $\mathcal{M}$. We tested several alternatives, by using different linear, quadratic and hyperbolic combinations of the T and $\mathcal{M}$ values. In the following we limit the discussion to the functional form that gave the best results, and that we used in the reported experiments:

$$r = c_1 \left(\frac{W_{best}}{T} \right) + c_2 \left(\frac{I_{sup} - I_{inf}}{(I_{sup} - I_{inf}) + (T - I_{inf})} \right). \tag{2.7}$$

[6]In this case, the core of the algorithm is based on the capability of "real" ants to discover shortest paths communicating by means of pheromone trails (Goss et al., 1989; Beckers et al., 1992).

```
$t$ := Current time;
$t_{end}$ := Time length of the simulation;
$\Delta t$ := Time interval between ants generation;
foreach (Node)    /* Concurrent activity over the network */
  $\mathcal{M}$ = Local traffic model;
  $\mathcal{T}$ = Node routing table;
  while ( $t \leq t_{end}$ )
    in_parallel    /* Concurrent activity on each node */
      if ( $t \bmod \Delta t = 0$ )
        destination_node := SelectDestinationNode(data_traffic_distribution);
        LaunchForwardAnt(destination_node, source_node);
      end if
      foreach (ActiveForwardAnt[source_node, current_node, destination_node])
        while (current_node $\neq$ destination_node)
          next_hop_node := SelectLink(current_node, destination_node, $\mathcal{T}$, link_queues);
          PutAntOnLinkQueue(current_node, next_hop_node);
          WaitOnDataLinkQueue(current_node, next_hop_node);
          CrossTheLink(current_node, next_hop_node);
          PushOnTheStack(next_hop_node, elapsed_time);
          current_node := next_hop_node;
        end while
        LaunchBackwardAnt(destination_node, source_node, stack_data);
        Die();
      end foreach
      foreach (ActiveBackwardAnt[source_node, current_node, destination_node])
        while (current_node $\neq$ destination_node)
          next_hop_node := PopTheStack();
          WaitOnHighPriorityLinkQueue(current_node, next_hop_node);
          CrossTheLink(current_node, next_hop_node);
          UpdateLocalTrafficModel($\mathcal{M}$, current_node, source_node, stack_data);
          reinforcement := GetReinforcement(current_node, source_node, stack_data, $\mathcal{M}$);
          UpdateLocalRoutingTable($\mathcal{T}$, current_node, source_node, reinforcement);
        end while
      end foreach
    end in_parallel
  end while
end foreach
```

Figure 2.2: AntNet's top-level description in pseudo-code. All the described actions take place in a completely distributed and concurrent way over the network nodes (while, in the text, AntNet has been described from an individual ant's perspective). All the constructs at the same level of indentation inside the context of the statement `in_parallel` are executed concurrently. The processes of data generation and forwarding are not described, but they can be thought as acting concurrently with the ants.

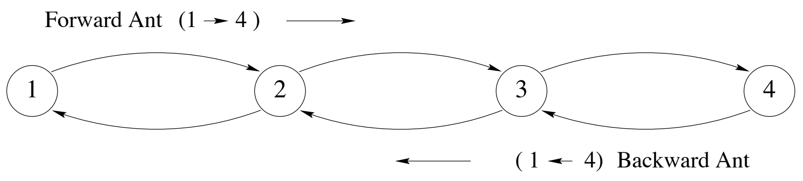


Figure 2.3: Example of AntNet behavior. The forward ant, $F_{1\to 4}$, moves along the path $1 \to 2 \to 3 \to 4$ and, arrived at node 4, launches the backward ant $B_{4\to 1}$ that will travel in the opposite direction. At each node k, $k = 3, \ldots, 1$, the backward ant will use the stack contents $S_{1\to 4}(k)$ to update the values for $\mathcal{M}_k(\mu_4, {\sigma_4}^2, \mathcal{W}_4)$, and, in case of good sub-paths, to update also the values for $\mathcal{M}_k(\mu_i, {\sigma_i}^2, \mathcal{W}_i)$, $i = k+1, \ldots, 3$. At the same time the routing table will be updated by incrementing the goodness P_{j4}, $j = k+1$, of the last node $k+1$ the ant $B_{4\to 1}$ came from, for the case of node $i = k+1, \ldots, 4$ as destination node, 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 destination node i. As for $\mathcal{M}$, the routing table is always updated for the case of node 4 as destination, while the other nodes $i' = k+1, \ldots, 3$ on the sub-paths are taken in consideration as destination nodes only if the trip time associated to the corresponding sub-path of the forward ant is statistically good.

In Equation 2.7, W_{best} is the best trip time experienced by the ants traveling toward the destination d, over the last observation window $\mathcal{W}$. The maximum size of the window (the maximum number of considered samples before resetting the W_{best} value) is assigned on the basis of the coefficient η of Equation 2.1. As we said, η weights the number of samples effectively giving a contribution to the value of the μ estimate, defining a sort of moving exponential window. Following the expression for the number of effective samples as reported in footnote 3, we set $|\mathcal{W}|_{max} = 5(c/\eta)$, with $c < 1$. In this way, the long-term exponential mean and the short-term windowing are referring to a comparable set of observations, with the short-term mean evaluated over a fraction c of the samples used for the long-term one. I_{sup} and I_{inf} are convenient estimates of the limits of an approximate confidence interval for μ. I_{inf} is set to W_{best}, while $I_{sup} = \mu + z(\sigma/\sqrt{|\mathcal{W}|})$, with $z = 1/\sqrt{(1-\gamma)}$ where γ gives the selected confidence level.[7] There is some level of arbitrariness in our computation of the confidence interval, because we set it in an asymmetric way and μ and σ are not arithmetic estimates. Anyway, what we need is a quick, raw estimate of the mean value and of the dispersion of the values (for example, a local bootstrap procedure could have been applied to extract a meaningful confidence interval, but such a choice is not reasonable from a CPU time-consuming perspective).

The first term in Equation 2.7 simply evaluates the ratio between the current trip time and the best trip time observed over the current observation window. This term is corrected by the second one, that evaluates how far the value T is from I_{inf} in relation to the extension of the confidence interval, that is, considering the stability in the latest trip times. The coefficients c_1 and c_2 weight the importance of each term. The first term is the most important one, while the second term plays the role of a correction. In the current implementation of the algorithm we set $c_1 = 0.7$ and $c_2 = 0.3$. We observed that c_2 shouldn't be too big (0.35 is an upper limit), otherwise performance starts to degrade appreciably. The behavior of the algorithm is quite stable for c_2 values in the range 0.15 to 0.35 but setting c_2 below 0.15 slightly degrades performance. The algorithm is very robust to changes in γ, which defines the confidence level: varying the confidence level in the range from 75% to 95% changes performance little. The best results have been obtained for values around

[7]The expression is obtained by using the Tchebycheff inequality that allows the definition of a confidence interval for a random variable following any distribution (Papoulis, 1991) Usually, for specific probability densities the Tchebycheff bound is too high, but here we can conveniently use it because (i) we want to avoid to make assumptions on the distribution of μ and, (ii) we need only a raw estimate of the confidence interval.

75%÷80%. We observed that the algorithm is very robust to its internal parameter settings and we didn't try to "adapt" the set of parameters to the problem instance. All the different experiments were carried out with the same "reasonable" settings. We could surely improve the performance by means of a finer tuning of the parameters, but we didn't because we were interested in implementing a robust system, considering that the world of networks is incredibly varied in terms of traffic, topologies, switch and transmission characteristics, etc.

The value r obtained from Equation 2.7 is finally transformed by means of a squash function $s(x)$:

$$s(x) = \left(1 + exp\left(\frac{a}{x|\mathcal{N}_k|}\right)\right)^{-1}, \qquad x \in \ (0,1], \quad a \in R^+, \tag{2.8}$$

$$r \leftarrow \frac{s(r)}{s(1)}. \tag{2.9}$$

Squashing the r values allows the system to be more sensitive in rewarding good (high) values of r, while having the tendency to saturate the rewards for bad (near to zero) r values: the scale is compressed for lower values and expanded in the upper part. In such a way an emphasis is put on good results, while bad results play a minor role.

The coefficient $a/|\mathcal{N}_k|$ determines a parametric dependence of the squashed reinforcement value on the number $|\mathcal{N}_k|$ of neighbors of the reinforced node k: the greater the number of neighbors, the higher the reinforcement (see Figure 2.4). The reason to do this is that we want to have a similar, strong, effect of good results on the probabilistic routing tables, independent of the number of neighbor nodes.

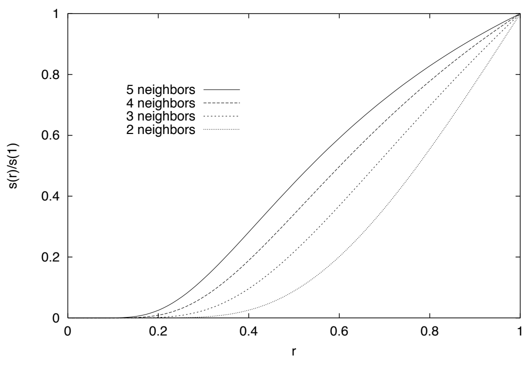


Figure 2.4: Examples of squash functions with a variable number of node neighbors.

2.2 AntNet-FA: AntNet with flying ants

In AntNet forward ants make use of the same queues used by data packets. In this way they behave like a data packet and experience the same trip time a data packet would experience. The problem with this approach is that if there is congestion along the followed path, the journey of the forward ant can last a considerably long time. Accordingly, the associated reinforcement signal will be very low and the ant will be delayed in reinforcing its path. On the contrary, ants which followed less congested paths will update earlier and with higher intensity the routing tables along the paths they followed, "correctly" biasing in this way the choices of subsequent ants and data packets. What is wrong with this picture? Let's think about the following scenario: an ant is following a very good path but there is a temporarily congestion at the current node the ant is. The ant is then forced to wait for some long time. When it can leave, the jamming problem was resolved but the ant had been hopelessly slow down and the whole path will not receive the appropriate rewarding. In an an even worse scenario a path gets congested "behind" the ant. The ant arrives quickly but its picture of the path is not anymore conformal to the reality. This can happen with a probability which is increasing with the increasing of the number of nodes in the path. Two problems can be readily identified: (i) waiting on the same, low priority, queues as data packets can determine the fact that acquired view of the traffic status be completely out-of-date when the routing tables are updated, (ii) using the forward time to update the routing table when going back can suffer of the same out-of-date/time-shift problem.

How these potential problems could be avoided? We propose the following strategy: (i) forward ants must make use of high-priority queues as backward ants do, and (ii) backward ants can update the routing tables in the visited nodes using estimates of ants' trip times. These estimates are computed at each node k, using a local statistical model $\mathcal{L}_k^l$ capturing the depletion dynamics of each of the local links l. Of course the reliability of this model appears to be critical for the proper functioning of the whole scheme. Actually in this thesis, according to what already have been done in (Di Caro & Dorigo, 1998f), we use a model very simple but able to provide more than satisfactory results. Given these results, we did not feel the need to look for more sophisticated models. The simple model $\mathcal{L}$ we used is the following: given the number of bits q_l of the data packets waiting in the queue of l, the virtual trip time to reach the desired neighbor is computed as $d_l + (q_l + s_a) = B_l$, where B_l is the bandwidth of the link, s_a is the size of the ant packet and d_l is the link's propagation delay. In this new version of AntNet forward ants apply the same stochastic policy to discover a feasible good path as in AntNet, but they do not wait in the data queues, they quickly move to discover a path that is then scored by backward ants using trip times locally estimated by means of the models $\mathcal{L}$. Note that now forward ants are also lighter from a computational point of view, since they do not carry in the memory stack any information about their experienced times $T_{s\rightarrow k}$.

Compared to the previous model, where the ants were slowly "walking" over the available, jammed, "roads", here the ants can be figuratively thought as "flying" over the data queues to quickly reach the target. Because of this expressive way of looking at the algorithm, we call *AntNet-FA* this improved version of AntNet, where the acronym FA stands for "flying ants"!

2.3 The need for a new algorithm: AntNet++

According to the extensive suite of simulation tests that will be reported in Chapter 3, AntNet shows performance superior to several state-of-the-art algorithms in the field of adaptive routing. Experimental results will additionally show that AntNet-FA provides performance even much superior to those of AntNet, with the improvement in performance being related to the increasing in the size of the network.

In spite of these very encouraging evidences, AntNet and AntNet-FA are both deficient in some aspects that turn out to be very important in the context of current and forthcoming network environments, AntNet and AntNet-FA from one side miss some adaptive and learning components "necessary" in a dynamic and distributed environment as that of telecommunication networks, on the other side, they are not really open to the forthcoming scenarios of active and quality-of-service networks. Generally speaking, three main high-level components can be easily recognized as definitely missing:

- adaptivity in the scheduling of new ants and in the definition of the values for their internal parameters,
- adaptivity in the decision and updating policy of the single ant in order to cope with the specificity of local situations,
- explicit support for the management and allocation of network resources, as required in the context of connection-oriented and quality-of-service networks.

The problem issued in the last item could be "easily" overcame by suitably expanding the algorithms' structure to accommodate for new *ad hoc* components. On the contrary, the questions on adaptivity raised in the first two items deserve a deeper discussion.

2.3.1 To be adaptive or not to be adaptive?

The rationale behind the lackness of adaptivity in many internal components, as well as in general the lackness of some important network-specific structures, can be found back in the characteristics of the ACO framework where AntNet and AntNet-FA have been originally developed. As it has been previously pointed out, ACO algorithms have been originally developed to solve combinatorial optimization problems. While formally the problem of online model-free routing to which AntNet algorithms are addressed can be considered as a problem of combinatorial optimization, in the practice the term combinatorial optimization is usually referred only to deterministic or model-based stochastic (but stationary) combinatorial problems where global information is always accessible. It is to this class of problems that the majority of ACO algorithms have been in fact applied to, with AntNet(s)[8] being a somehow "natural" outgrowth of this stream of algorithms with few additional network-specific features and hackings. As the experimental results in Chapter 3 show, this process has been quite successful in designing a novel and well performing class of multi-agent algorithms for routing, but at the same time it brought into the new algorithms some assumptions that were reasonable for combinatorial optimization problems but that are not longer effective for the specific environment of telecommunication networks. In particular, when the problem dynamics has characteristics of stationarity and global information is available, as usually assumed in the above context of combinatorial optimization, we do not "need" to be adaptive. Yes, learning and adaptivity can be *optional* design choices to improve the algorithm performance but as a matter of fact, most of ACO algorithms for combinatorial optimization do not have many adaptive and/or learning components. The basic ACO structure is a learning process by itself (see Section 1.1) with respect to the estimation of the action-value function at the decision points, but the various components of most algorithms (e.g., number of ants used at each iteration) are instantiated in a quite straightforward/heuristic and non-adaptive way. Given the characteristics of the problems this is quite reasonable. Making everything adaptive would dramatically increase the number of coupled components that should be tuned. Increasing in this way the algorithm complexity, hiding some essential and general properties, and, likely, reducing its robustness and/or performance in case of a non too much accurate tuning process.

The situation is somewhat different in the field of telecommunication networks, where being adaptive and possessing learning capabilities[9] is somehow *necessary* to take into account for the intrinsic variability and spatial distribution of the network environments. So, the recipe would appear easy: heavily bring adaptive and learning components inside algorithms for network problems. Well, the dark side of the story is that this is very hard to do. Networks suffer of what can be called *perceptual aliasing* (see for example (Chrisman, 1992)), where different situations that should be treated in a different way are unfortunately not distinguishable on the only basis of perceptual input, that makes them appearing as a same unique situation. Similarly, in networks all the possible decision and learning processes happen at the local level of the nodes on the base of some *belief states* about what is going on all over the network. Beliefs are built on the base of some "perceptual" information, which, in networks, is by its same nature incomplete and/or delayed. Nodes' beliefs cannot capture a complete and updated representation of the real state of the network, they suffer of perceptual aliasing. Many different traffic situations are seen as the same situation.
To explain this with some detail, let us consider the following example. At node s an ant reports at time t_1 a trip time $T^{t_1}_{n \to d}$ much bigger than the average of the trip times observed up to that moment when passing through neighbor n to go to d from s. $T^{t_1}_{n \to d}$ and the past records and stored statistics for $T_{n \to d}$ are all the node s can "perceive" about the network

[8]Here and in the following AntNet(s) is used as a shortcut for "AntNet and AntNet-FA".

[9]Here the discussion is limited to the aspects of adaptivity and learning, but it could be extended to include also the aspects concerning with concurrency and multi-agency. In ACO algorithms groups of ants concurrently generate solutions through and iterative process. This use of concurrency (as opposed for example to the limit case of a process where only one ant is used at each iteration) is not supported by any theoretical justification. It is pure heuristic. On the contrary, in telecommunication networks it is in general *necessary* to use multiple concurrent agents to deal with the problems under consideration.

status for the considered pair n, d. Node's beliefs are built on the basis of this incomplete knowledge. What can be said about the cause of the increase in the expected delay when going to d by passing through n? Precisely, nothing much. The observed perceptual information can be associated to different network situations that cannot be easily unaliased. The same perception can be in fact associated to the following rather different situations: (i) there is a general congestion status all over the network, (ii) the congestion is low but the routing is bad because a memory of a previous congestion is still influencing the routing choices, (iii) there is a congestion only between the two considered nodes and in particular along the most used path connecting them, (iv) there is some node or link fault situation not yet properly propagated in the routing tables, (v) ....

Therefore, given the above facts, how is it possible to be adaptive in a proper way? Adaptiveness gives the ability to modify behaviors and parameters to match at best a new situation, but what if the situation itself cannot be clearly understood? In general terms, "proper" adaptiveness needs to possess the *state* description of a situation, not an incomplete belief. In networks this is in general impossible. Therefore, adaptiveness must be used with extreme care and some suspiciousness. This does not mean that algorithms' components should be made non-adaptive or at least very slowly adaptive. Here we claim that adaptivity can be a very powerful additive for network algorithms but it must be used with extreme care and adopting a *conservative* attitude. Conservative in the sense that before making big changes it is probably better to be somehow on hold, collect more data to reduce the probability of incurring in aliases and act without generating big modifications if the the situation appears too ambiguous. For example, in the example just described, after collecting the data at time t_1 the strategy of the algorithm that we are going to describe below is to send a new ant from s to d through all the neighbor nodes of s to have a more clear idea about where the congestion could be. The idea we like to follow is that even given the hard constraints of network environments it is possible to dramatically increase the performance of an algorithm by adding degrees of adaptivity *cum grano salis*.

The issue of adaptivity at a general level has been struggling Internet administrators since ever. As briefly reported in Appendix B, after several attempts to use an adaptive algorithm, engineers ended up with using OSPF, an essentially static algorithm. In the Appendix some rationale behind this choice is given. Here we report this fact to emphasize that is well understood that adaptivity can give a big positive jump in performance. On the other side, it is also well known that is easy to incur in the opposite effect of a big negative jump if the whole system is not properly designed. Concerning with this issue, an advantage that the AntNet family of algorithms probably has with respect to the algorithms that have been previously considered for routing on the Internet is the following. All the previous algorithms propagate distance or cost estimates from one node to the other. At every node, estimates are updated in a recursive manner using this "waves" of information coming from other nodes. This system works at the best under assumptions of stationarity but is prone to show potentially uncontrolled oscillations under non-stationary situations. Additionally, the rate at which new cost information is injected from each node into the network is a very critical parameter. In AntNet algorithms the situation is slightly different. AntNet's ants do not propagate information. They sample the network status and report this information to update local statistics. No one single estimate is directly propagate from one node to the other. As it will be discussed more in detail in Chapter 4, AntNet algorithms are pure Monte Carlo algorithms and they do not bootstrap (see for example (Sutton & Barto, 1998) for a good discussion about Monte Carlo vs. methods like dynamic programming that bootstrap statistical estimates). This aspect would expect to make the performance of AntNet algorithms inferior to those of bootstrapping algorithms in case of stationary or quasi-stationary conditions. On the other side, AntNet is expected to be more robust to wrong estimates, traffic fluctuations and to the utilization of an higher degree of adaptiveness.

2.3.2 AntNet++: an open agent architecture for networks

AntNet++, the algorithm introduced in this section, tries to overcome the limitations of AntNet(s) discussed in the previous section. At the same time, in the development of this new algorithm we did not forget about the precautions that should be always used when adaptivity comes into the matter. As pointed out before, some rationale behind the "limitations" in AntNet(s) can be found in their context of origin. In this sense, AntNet++ goes further, making a step ahead with respect to the strict ACO's "tradition"' to which AntNet(s) belong to. In fact, while AntNet++'s basic components and operational structure firmly root in those of AntNet(s) (and then, of ACO), its logical architecture has been redesigned to match the specific, but at the same time general, characteristics of the class of problems, the network problems, to which the algorithm should be applied to. In particular, the guidelines for the design of AntNet++ have been defined having in mind the following facts of strategic importance for network environments:

Non-stationarity - Network dynamics is intrinsically non-stationary and information about the current network status cannot be globally accessed;

Distributed decisions - Networks are a distributed decision and management system, where every node must act on the basis of only locally available information, projection of some global network status.

Activation - Forthcoming networks will be active: packets will be able to carry their own execution or description code and all the network nodes will be able to perform, as normal status of operations, customized computations on the packets passing through them.

Quality-of-service - New broadband technologies make possible to deliver mission critical business over the networks. This opens a wide range of different possibilities in network utilization and pricing. At this aim the software components of a network must be able to exploit the ever increasing communication capacity to provide services with guaranteed performance: required resources must be efficiently discovered and allocated to make good profits.

To account for the above four set of issues, in AntNet++ several additional self-adaptive components have been introduced and its logical architecture has been designed in such a way that it can be suitably described in terms of a *distributed society of mobile and static agents*, where the *learning*, *perception*, *decision* and *action* components are kept on distinct levels and managed by different agents.

As it will be clearer in the following, this characterization in terms of a distributed society of heterogenous adaptive agents reflects what most likely will be the future scenario in the world of telecommunications. Mobile agents, carrying their own code and specifications, will move across the networks, acting and interacting with other mobile or statically bound agents. They will be able to adapt themselves to new situations, to learn from the past experience, to replicate, to cooperate or compete with other agents, to die if not anymore useful...

The descriptive "metaphor" of AntNet++ in terms of a society of learning agents is not intended to be viewed as in opposition to the metaphor of ant colonies which inspired the ACO framework. On the contrary, it is intended to provide a representational language that extends the one used by ACO in the direction of being more suitable to deal with the complexity of real-world environments. Informally speaking, in AntNet++ the emphasis is put on the word "society" as a synonym for the word "colony" and it is admitted that to obtain state-of-the-art performance and flexibility in real-world situations some "intelligence" (in terms of adaptivity, learning and social rules) must be added to the members of the colony.

Without any pretentiousness, AntNet++ is an initial attempt to bridge the gap between real-world environments and agent, multi-agent learning and swarm intelligence systems.

Having as a target specific real-world problems defines a strong constraint: the system must show performance competitive with respect to other (possibly) commercially available products. Therefore, the system cannot be just a “proof of concept” for some specific computational paradigm. It must take into account the complexity and limitations of the reality while providing good performance.

At this aim the approach of AntNet++ appears quite sound. In fact, AntNet++ take into account the intricate mosaic of different aspects that play an important role in the problem under consideration and maps it directly into a general modular architecture where different computational paradigms can be used in the actual implementation of the different modules. In this respect AntNet++ is not strictly only an instance of an algorithm. More appropriately, AntNet++ should be seen as the definition of a novel multi-agent architecture for network control and management. In fact, AntNet++ defines the components of a specific agent society, their relationships and the functionalities that must be associated to each component of the society. The description of the structure and of the components of this society is the object of the next section. Additionally, in Section 2.3.3 we provide “also” some details about a specific instantiation of an AntNet++ system. Anyway, since AntNet++ is still undergoing tests and development, this description will be often kept on a rather general and informal level, mostly discussing the characteristics of the sub-problems that should be solved, the possible alternative solutions and the rationale behind them.

2.3.3 Learning, perceptual and effector agents in AntNet++

As pointed out above, the non-stationarity and the distributed control view are two major issues that we have to take into primary consideration while designing an effective routing system. In these terms a routing system can be seen as made up by a collection of intelligent control agents, each statically bound to a different network node. Each agent is (possibly) intelligent in the sense that it accomplishes its routing goals by observing the environment and the effect of its decisions on the environment, and makes use of this information to adaptively change its decision policy in the direction that appears more appropriate to maximize the network’s global performance. Let us call these *static* agents *node managers*, being devoted to several node-based activities, like the monitoring of local traffic, the building of routing tables, the routing of data packets. A node manager interacts with its (statically assigned) local environment by making *observations* of the flow of data and routing packets, and taking routing *decisions*. Additionally the node manager can generate *active perceptions*, that is, specific actions can be purposely issued at the aim of collecting additional, possibly non-local, information. Making local observations, as well as active perceptions, has associated the cost of using some computational and bandwidth resources. Therefore, the node manager at each time must decide which kind of “action” (local observation, remote active perception, packet routing, etc.) looks more appropriate according to the node status and to the estimated costs/benefits.

As just described a node manager is a “classic” adaptive controller acting in open-loop in a complex environment. Here the situation is hopelessly complicated by the already emphasized facts that (i) the environment dynamics is not stationary, (ii) local observations bring only a local and necessarily incomplete view of the network status, while remote perceptions can bring only delayed information, (iii) node managers must act concurrently without any global coordination. Because of these three facts, each controller can be also more appropriately identified as an agent involved in a *continual* task of *reinforcement learning*.

According to this general but at the same time quite expressive way of looking at a routing system, the architecture of AntNet++ has been conveniently designed in the terms of a hierarchical society of heterogeneous agents:

- The *node managers*, one for each node, non-mobile reinforcement learning agents all situated at the same hierarchical level.

- The *active perceptions*, mobile reactive agents ancillary to the node managers who can generate them in an arbitrary number.
- The *effectors*, mobile executor agents, their too ancillary to and generated by the node managers, to carry out pre-compiled and highly specialized tasks.

In the following a somehow detailed description for the activities of each of such agents is provided, organized in lists of items. The basic global behavior of AntNet++ is not explicitly described since it results from the asynchronous and concurrent actions of all the individual agents active on the network. In general, AntNet++'s global behavior is very similar to that of AntNet-FA, on top of which it has been built.[10] Therefore, all the not explicitly mentioned mechanisms are assumed to be the same as those of AntNet-FA.

Node managers

- For each network node there is a *node manager*, a static agent whose task is the adaptive (reinforcement) learning of network traffic models that in turn it uses for the management of the local packet routing and resource allocation.
- Node managers can straightforwardly observe the local traffic flow but this information can be insufficient to provide good performance. Therefore, each node managers can expand its "sensory field" by adaptively generating *active perceptions* agents, that is, ant-like mobile agents that move away from the node, explore the network and gather back to the node manager the information it required.

  The term "active" is referred to the fact that each of these perceptual acts are generated with a possibly different set of parameters to precisely get information about a specific area of the network by using specific parameters for the path construction strategy. That is, the node manager has a high degree of control over the type of information that must be collected by its sensors, the perceptual agents.

  Node Managers must behave socially, so every node manager is allowed to read the information carried by any perceptual agent, but no one else but the creator of a perception can possibly modify the perception internal state in a direct way.
- Active perceptions are scheduled both on a *event-driven* and *event-independent* basis:

  Event-driven. Events that in AntNet++ trigger the generation of new perception agents are the following two:

  (i) The setup of a new application. In the case of an application in a best-effort connection-less network, at the start up of a new application $A_{s\to d}$ a burst of $m \propto |\mathcal{N}(s)|$ new perception agents are generated toward d by the node manager in s to collect updated information about the best ways to forward A's packets. We also call these perception agents *application support perceptions*. Support perceptions are generated all long the application lifetime, at a rate that is adaptively tuned to adequately support the packet generation rate of the application (see below, in the section describing the active perceptions, for the additional agents generated in the case of the setup of an application in a connection-oriented or QoS network).

  (ii) The acquisition of some data that seem to strongly disagree with what had been learned up to that moment about the goodness of choosing a specific link for a certain destination. If a newly sampled trip time $T_{s\to d}$ indicates a journey value much bigger than the one estimated up to that moment, then it can be worth to understand what is going on. In fact, this "unexpected"

[10]AntNet-FA showed performance superior than those of AntNet and at the same time is more suitable for connection-oriented services. Therefore, it has been chosen as basis for the development of AntNet++ instead of AntNet.

value could have been caused by a "wrong" choice made by the perception due to the stochasticity in its decision policy, or, it could have been caused by a some situation of congestion. If the link in question was among the best possibles, then before keeping trusting it or reducing its utilization, it can be worth to generate some perception agents with destination d through all the possible local outgoing links to gather updated wide-spectrum information.

Event-independent. A *background flow* ω_i of active perception agents is continually and independently generated by each node manager i to keep an updated view of the network status. The frequency of this background flow is made *adaptive* in AntNet++, while it was heuristically assigned and kept constant in AntNet(s). While experimental results in figure 3.15 in Section 3.3.5 suggest that the AntNet is quite robust with respect to the choice of the background frequency, they also show that, with an appropriate tuning, performance could greatly be improved. In general, answering the question "At which rate routing agents should be generated ?" is at the same time extremely difficult and of fundamental importance for any adaptive routing algorithm. In Section 2.3.1 we already discussed, in a more general way this issue. Here we add some further considerations and then we discuss some practical ways of answering the question that have been implemented in AntNet++.

First, let us pointing out four main factors that play a major role in this issue of how adaptively define ω_i: (i) the global and local network transmission bandwidth, (ii) the current traffic/congestion profile, (iii) the actual global impact of the routing traffic on the global traffic situation, (iv) the global impact of each local assignments to ω_i.

The value of the network transmission capacity pointed out in (i) is a very useful starting point for the definition of an initial value and an upper limit for ω_i. Actually, considering the whole capacity would be misleading. On the contrary, the total transmission capacity B_i at every node i can be used to define both a local upper bound, $\omega_i \leq aB_i,\ a \in (0, fracmax)$, where possibly $fracmax < 0.5$, and an initial value, $\omega_i = cB_i$, where c could reasonably be in $c \in (0, 0.1)$.

The issue in (ii) concerns, again, with the very general and of hard solution question discussed at the beginning of this Section and in Section 2.3.1. The point (iii) and (iv) are related to the general question (ii) but concern with a less general aspect. Our belief/assumption here is that, in spite of the impossibility to have a clear view of the impact of the routing agents on the traffic load, still some useful quantities can be locally measured in order to produce reliable estimates of it. Our belief is supported by the important fact that the node managers are supposed to have, by design, a *social attitude* with respect to the perception generation rate. That is, every node manager must try to set ω_i in the way that it can acquire useful additional information, avoiding, at the same time and with higher priority, the generation of an excessive and unwanted load of routing traffic. Once we admit that all the node managers will behave in this social way then, we can also assume that the observation of the local routing flow can be sufficient to set ω_i in an appropriate way. Restated, this argumentation, would sound like: since all the node managers adopt the same type of social strategy in setting ω_i, and since they are the only responsibles for the generation of the routing load, then, changing each ω_i in an adaptive and independent way on the base of only local information, can be expected to produce good results. Where good results means not overloading the network with routing data, collecting, at the same time, as much as possible useful perceptual information.

What type of information can be locally observed in order to have a reliable view about the impact of the generation of routing agents? In AntNet++ we focus on: (i) the status of the local buffers for both routing and data packets, (ii) the behavior of the local throughput for data and routing packets over an assigned time window. The strategy used for setting ω is summarized in the following piece of simple C++ code:

```
// Init $\omega$
$I_\omega = 0.01$;
$\omega = I_\omega \cdot$ locally_available_bandwidth;

$C_{th} = 0.2$; $C^-_{buf} = 0.1$; $C^+_{buf} = 0.001$; $C^-_w = 0.2$; $C^+_w = 1.2$; $C^+_{wr} = 0.0001$;

// Check if data are slowed down because of routing packets

if( routing_window_throughput $\geq C_{th} \cdot$ data_window_throughput )
{ // The routing throughput appears too higher than that of data ...

  if( data_bits_waiting $\geq C^-_{buf} \cdot$ total_size_of_local_buffers )
    { // ... and too many data bits are waiting: decrease $\omega$

      $\Delta = (C^-_{buf} \cdot$ total_size_of_local_buffers) / data_bits_waiting;
      $\omega = \Delta \cdot \omega$;
    }
  else if( routing_bits_waiting > Cw $\cdot$ data_bits_waiting )
    { // ... not too many data are queued but waiting routing bits are excessive: decrease

      $\Delta$ = (Cw $\cdot$ data_bits_waiting) / routing_bits_waiting;
      $\omega = \Delta \cdot \omega$;
    }

  else if( (data_bits_waiting < $C^+_{buf} \cdot$ total_size_of_local_buffer) &&
               ( (data_bits_waiting > $C^+_w \cdot$ routing_bits_waiting) ||
                 (routing_bits_waiting < $C^+_{wr} \cdot$ total_size_of_local_buffers) ) )
    { // ... few data bits are waiting and even less routing bits too: let's try to increase

      $\Delta$ = data_bits_waiting / ($C^+_{buf} \cdot$ total_size_of_local_buffer);
      $\omega = (1 + \Delta) \cdot \omega$;
    }
  else
    { // ... the situation is hard to evaluate, it is better to be conservative and not to
change $\omega$
      $\Delta = 1$;
      $\omega = \Delta \cdot \omega$;
    }
}

else if( (data_bits_waiting $\geq C^-_{buf} \cdot$ total_size_of_local_buffers) &&
          (routing_bits_waiting > $C^-_w \cdot$ data_bits_waiting) )
{ // ... data throughput has not been decreased by routing but buffers are too full: decrease

  $\Delta = (C^-_w \cdot$ data_bits_waiting) / routing_bits_waiting;
  $\omega = \Delta \cdot \omega$;
}

else
{ // ... throughput has not been decreased, buffers situation is not worrying, so we can
increase
  $\Delta = 0.001 \cdot \omega$;
  $\omega = \omega + \Delta$;
}
```

At a first glance the used strategy looks rather intricate. Admittedly, this is true, but the characteristics of the problem itself force to reason in a such a way. We considered the variables of interest (throughputs and length of buffers) and tried to define, over their whole domain of variability, those range of values that could be associated to limit, pathological situations (e.g., buffers too full, data throughput considerably slowed down by routing throughput, etc.). Such situations are more easily recognized, the problem of perceptual aliasing is much reduced. More importantly, because of their same nature, they would require an adaptive change in the value of ω to move towards less pathological regimes. The non pathological situations are hard to be unaliased and then properly understood. In these cases AntNet++ follows a strategy conservative with respect to the current value of ω.

The rather big number of parameters could leave the reader a bit perplex about their necessity and the real "adaptivity" of the algorithm. Actually is important to notice that all the parameters, as well as the variable ranges, are not defined on an absolute scale of values but as perceptual coefficients. The specific values assigned to these percentages correspond to the amount of risk accepted *a priori*: for example, which is the maximum fraction of the data throughput that the routing throughput should not be tolerated to surpass?

- Node managers would generate every active perception with a set of parameters *ad hoc* for the type of sensorial task the agent has been created for. We are still exploring how an efficient selection of parameters could be done. Apart from some specific cases (see below for the case of active perceptions supporting the setup of a QoS application) where it is quite clear which kind of characteristics the perception agent should have, in general it is not so obvious which strategy is the best. At the moment AntNet++ makes use of a randomized sampling strategy for assigning some of the active perceptions' parameters. Values are probabilistically sampled from a rather skewed gamma distribution. The parameters that are generated in this way are: some parameters regulating the level of stochasticity in the agent decision policy, the coefficient a (formula 2.8) that regulates the intensity of the updating reinforcements, the weight parameter α (2.3 for the balancing between instantaneous and long-term estimates. This probabilistic selection of some of the agent characteristics determines in the agent population an high level of *diversity*, which is an effective feature in a multi-agent system operating in a non stationary environment.

- In the case of *best-effort connection-less traffic*, the policy used by node managers to update the routing tables and the statistical models basically follows the same rules as in AntNet-FA, at least in this preliminary version of AntNet++.
  In the case of *QoS traffic*, the structure of the used routing tables and statistical data can be different from the best-effort case. Accordingly the updating policy can have a different form. For example, Link-state (see appendices) tables can be used in conjunction with the usual distance-vector tables to combine source with non-source routing. This is the approach followed in (Di Caro & Vasilakos, 2000), where the structure of each node manager include a stochastic estimator learning automata (Vasilakos & Papadimitriou, 1995). Here we do not give details about this implementation, being the work still in progress and under testing.

Active perceptions and Effectors

- *Active perceptions* are essentially the "usual" ants of AntNet(s), enriched with some new functionalities, as described below, but deprived of their decisional power for updating the node data structures. In AntNet++, in fact, these agents communicate the carried information to the node managers but it is the node manager's responsibility to use this information to update their private statistical models.

  This model of interaction is typical of agent and object-oriented programming. Node managers and active perceptions are different computational entities with a hierarchical relationship. The active perception is created by the node manager with the only purpose of collecting useful information and therefore it should not directly modify the internal state of the hierarchically superior node manager. This should happen for at least three main reasons. First, by design the component for "intelligent" processing of the sensorial data is supposed to be assigned to the node manager, the learning component of the systems, and not to its perception. Second, for security reasons: what if a competing and "malicious" network provider would generate intruder perceptions carrying wrong information? AntNet++ does not address this problem directly but shielding a potential enemy from gaining access to the main node data structures is an issue of critical importance. Third, a direct modification of the node data structures would be against the principles of information hiding at the base of all the modern (object-oriented) programming. The perception does not need to know about which kind of models, structures, data are used by the node managers, all it does need to know to do its job is the protocol to communicate its data. While internal data representations could change very often if, for example, different learning algorithms are tested, the communication protocol, playing the role of an interface, once established, is expected to have a much longer lifetime if designed in a quite general way.

- AntNet++'s active perceptions have the peculiar characteristic of *replicating* when more than one single outgoing link seems to be very appealing. Let $F_{s \to d}$ be a "forward" perception moving from s to d. Let be k the current node. An AntNet(s) ant would just pick-up the probabilistically best link and move over it. Acting in this way, the ant just does not care about other potentially very promising links. In AntNet++ we care. If after the computations of formula 2.3 there is no one single "best" link but a set $\{n_1, n_2, \ldots, n_b\}, b > 1$, of "best" links at a same level of estimated goodness, then the perception replicates in b copies. Each copy proceeds on a different link $n_i, i = 1, \ldots, b$ to explore all these equivalent possibilities.
  To avoid an uncontrolled proliferation of perception agents, a replica, as well as an agent that already underwent a replication, is allowed to replicate only once more. Of course, this rule is purely heuristic, as well as the threshold value that should decide at which level the goodness of the links should be considered as equivalent.

- Active perceptions can be of *different type* according to the type of network. In the simplest case of a best-effort connection-less network there is only one type of active perception agent, described above.
  In the case of *connection-oriented* or *QoS* networks there are more than one single type of perception. In particular, at the setup of a new application a burst of perception agents is generated to quickly discover an effective path to be allocated for the new application. The agents' exploratory attitude in this case is minimal because it is important to find a very good path in a as much as short time. While moving each agent reserves the resources that would be necessary to the application on the node path. If, while constructing a path, the characteristics of the path no longer match the QoS requirements, the perception moves backward, frees the previously reserved resources and terminates itself once back at the origin node. The first agent coming back with a feasible path (in the sense of QoS requirements) allows the application to start its activities using that path. If in the following other agents come back reporting additional good feasible paths, the traffic flow of the application is split over all the *multi-paths*. Once the application finishes its activity, an *effector agent* is generated to free the used resources. Also, when an active perception reports a path that is not accepted by the node manager, an effector agent is generated to free the temporarily reserved resources. Alternatively, node managers can use source routing to directly define a possible path to allocate the application. In this case no active perceptions are used but effector agents, that move over the pre-specified path and reserve the resources, if possible (note that the described QoS routing system is basically the same described in (Di Caro & Dorigo, 1998e; Di Caro & Vasilakos, 2000)).

- One of the most critical parameters of the whole AntNet family of algorithms is α, that in formula 2.3 weights the relative importance of the current queue length (instantaneous local situation) versus the value contained in the routing table (long-term memory). In spite of its importance, we can only rely on heuristics to assign a suitable value to α. AntNet++ active perceptions try to adapt in a simple way their first guess of α, as assigned by the generating node manager, to the local level of congestion. If all the link queues are almost in saturation, then the value of α is proportionally reduced. Similar variation happen if the queues are uniformly almost empty. In this way the importance attributed to the queue lengths is much reduced. These are somehow pathological situations where the information coming from the length of the queues is not really meaningful and should then reduce its impact on the decision.

- There is not so much to say about the *effector agents*. We have already introduced them in a previous item to illustrate their utilization in the case of allocation and deallocation of resources in a QoS network. In general they are "blind" *executor agents*, used to carry out tasks whose specifications have been completely defined at the level of the generating node manager.

- Node managers are the *learning agents*, they are the real actors of the whole system. On the other side, individual active perceptions are *reactive agents*, but they make use of the adaptive information contained in the routing tables. Therefore, in some sense we can say that the set of active perceptions, coupled with the stigmergic mechanism, can be globally seen as a *collective learning system*.

A society of agents

Using the language of the visual ant metaphor, the previous description of the components of AntNet++ could be somehow rephrased as follows: (i) the node managers can be seen as control centers for the activities of the colony, and then we can call them *colony controllers*, (ii) the active perceptions, somehow the "usual" ants of ACO, can be seen more appropriately as *scout ants*, and, (iii) the effector agents are like *worker ants*, that carry out the routine jobs for the colony as issued at the colony control centers.

No matter which descriptive metaphor is chosen, at global level the above system can be naturally described in terms of a *society of reinforcement learning agents*, (the static node managers or, alternatively, the colony controllers). Society's agents share the same environment and the same common global goals but each of them locally takes decisions in an independent way and act without any explicit coordination with the others in the generation of the additional perceptual and effector mobile agents. Anyway, they must be "social" in the sense that their actions concur at the same target, therefore both the generation of active perceptions and remote resource allocation must be done being aware of the other agents needs, without creating unwanted congestion in other network areas.

The interplay between static and mobile agents, as well as the clear distinction among the components devoted to learning, perception and local and remote actions, provide AntNet++ with a repertoire of interpretations, possibilities and applications much wider than that of AntNet and AntNet-FA. Moreover, as the ACO meta-heuristics defines a general framework for dealing with combinatorial optimization problems, AntNet++ defines a general multi-agent architecture for the management and control of a wide range of types of networks, ranging from best-effort to quality-of-service. According to the characteristics and constraints of the actual network, the different components can be accordingly instantiated in a different way.

Chapter 3

Experimental setting and results

In this Chapter we report about experimental settings and results to evaluate the performance of AntNet and AntNet-FA for best-effort traffic in connection-less networks. The reason why there are no experimental results reported for AntNet++ has been explained in the Introduction Chapter and the section specifically devoted to the description of AntNet++. The chapter is organized as follows:the algorithms which we used for comparison are described in Section 3.1, while in Section 3.2 the experimental settings are described in terms of traffic, networks and algorithm parameters, finally Section 3.3 reports several experimental results for both AntNet (in majority) and AntNet-FA.

3.1 Routing algorithms used for comparison

To evaluate the performance of AntNet and AntNet-FA we compared them with state-of-the-art routing algorithms from the telecommunications and machine learning fields. The following algorithms, belonging to the various possible combinations of static and adaptive, distance-vector and link-state classes (see Appendix A), have been implemented and used to run comparisons.

OSPF (static, link state): is our implementation of the current Interior Gateway Protocol (IGP) of Internet (Moy, 1998). Being interested in studying routing under the assumptions described in Section 1.2.3, the routing protocol we implemented does not mirror the real OSPF protocol in all its details. It only retains the basic features of OSPF. Link costs are statically assigned on the basis of their physical characteristics and routing tables are set as the result of the shortest (minimum time) path computation for a sample data packet of size 512 bytes. It is worth remarking that this choice penalizes our version of OSPF with respect to the real one. In fact, in the real Internet link costs are set by network administrators who can use additional heuristic and on-field knowledge they have about traffic workloads.

SPF (adaptive, link-state): 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 (McQuillan, Richer, & Rosen, 1980) and in its successive revisions (Khanna & Zinky, 1989). Our implementation uses the same flooding algorithm, while link costs are assigned over a discrete scale of 20 values by using the ARPANET hop-normalized-delay metric[1] (Khanna & Zinky, 1989) and the the statistical window average method described in (Shankar, Alaettinoğlu, Dussa-Zieger, & Matta, 1992a). Link costs are computed as weighted averages between short and long-term

[1]The transmitting node monitors the average packet delay $\overline{d}$ (queuing and transmission) and the average packet transmission time $\overline{t}$ over fix observation windows. From these measures, assuming an M/M/1 queueing model (Bertsekas & Gallager, 1992), a link utilization cost measure is calculated as $1 - \overline{t}/\overline{d}$.

real-valued statistics reflecting the delay (e.g., utilization, queueing and/or transmission delay, etc.) over fixed time intervals. Obtained values are rescaled and saturated by a linear function. We tried several additional discrete and real-valued metrics but the discretized hop-normalized-delay gave the best results in terms of performance and stability. Using a discretized scale reduces the sensitivity of the algorithm but at the same time reduces also undesirable oscillations.

BF (adaptive, distance-vector): is an implementation of the asynchronous distributed Bellman-Ford algorithm with dynamic metrics (Bertsekas & Gallager, 1992; Shankar et al., 1992a). The algorithm has been implemented following the guidelines of Appendix A, while link costs are assigned in the same way as described for SPF above. Vector-distance Bellman-Ford-like algorithms are today in use mainly for intra-domain routing, because they are used in the Routing Information Protocol (RIP) (Malkin & Steenstrup, 1995) supplied with the BSD version of Unix. Several enhanced versions of the basic adaptive Bellman-Ford algorithm can be found in the literature (for example the Merlin-Segall (Merlin & Segall, 1979) and the Extended Bellman-Ford (Cheng, Riley, Kumar, & Garcia-Luna-Aceves, 1989) algorithms). They focus mainly on reducing the information dissemination time in case of link failures. When link failures are not a major issue, as in this thesis, their behavior is in general equivalent to that of the basic adaptive Bellman-Ford.

Q-R (adaptive, distance-vector): is the Q-Routing algorithm as proposed by Boyan and Littman (1994). This is an online asynchronous version of the Bellman-Ford algorithm. Q-R learns online the values $Q_k(d,n)$, which are estimates of the time to reach node d from node k via the neighbor node n. Upon sending a packet P from k to neighbor node n with destination d, a back packet P_{back} is immediately generated from n to k. P_{back} carries the information about the current time estimate $t_{n\to d} = \min_{n'\in\mathcal{N}_n} Q_n(d,n')$ held at node n about the time to go for destination d, and the sum $t_{P_{k\to n}}$ of the queuing and transmission time experienced by P since its arrival at node k. The sum $Q_{new}(d,n) = t_{n\to d} + t_{P_{k\to n}}$ is used to compute the variation $\Delta Q_k(d,n) = \eta(Q_{new}(d,n) - Q_k(d,n))$ of the Q-learning-like value $Q_k(d,n)$.

PQ-R (adaptive, distance-vector): is the Predictive Q-Routing algorithm (Choi & Yeung, 1996), an extension of Q-Routing. In Q-routing the best link (i.e., the one with the lowest $Q_k(d,n)$) is deterministically chosen by packets. Therefore, a link that happens to have a high expected $Q_k(d,n)$, for example because of a temporary load condition, will never be used again until all the other links exiting from the same node have a worse, that is higher, $Q_k(d,n)$. PQ-R learns a model of the rate of variation of links' queues, called the recovery rate, and uses it to probe those links that, although not having the lowest $Q_k(d,n)$, have a high recovery rate.

Daemon (adaptive, optimal routing): 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:

$$C_l = d_l + \frac{S_p}{b_l} + (1-\alpha)\frac{S_{Q(l)}}{b_l} + \alpha\,\frac{\bar{S}_{Q(l)}}{b_l},$$

where d_l is the transmission delay for link l, b_l is its bandwidth, S_p is the size (in bits) of the data packet doing the hop, $S_{Q(l)}$ is the size (in bits) of the queue of link l, $\bar{S}_{Q(l)}$ is the exponential mean of the size of links queue and it is a correction to the actual

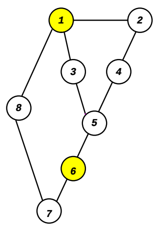


Figure 3.1: *SimpleNet*. Numbers within circles are node identifiers. Shaded nodes have a special interpretation in our experiments, described later. Each edge in the graph represents a pair of directed links. Link bandwidth is 10 Mbit/sec, propagation delay is 1 msec.

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. Of course, given the arbitrariness we introduced in calculating C_l, it could be possible to define an even better Daemon algorithm.

3.2 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 is defined and for each class our choices are explained.

3.2.1 Topology and physical properties of the networks

Topology can be defined on the basis of a real net instance, or it can be designed by hand, to better analyze the influence of important topological features (like diameter, connectivity, etc.), or it can be generated in some random way.

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

In our experiments, we used one simple hand designed network, two networks modeled on the characteristics of two real-world networks, two classes of randomly generated networks with a quite big number of nodes. For all of them but the random ones we describe the main characteristics and we summarize the topological properties by means of a triple of numbers (μ, σ, N) indicating respectively the mean shortest path distance, in terms of hops, between all pairs of nodes, the variance of this average, and the total number of nodes. From these three numbers we can get an idea about the degree of connectivity and balancing of the network. The difficulty of the routing problem roughly increases with the value of these numbers.

- *SimpleNet* (1.9, 0.7, 8) is a small network specifically designed to study some aspects of the behavior of the algorithms we compare. Experiments with SimpleNet were designed to closely study how the different algorithms manage to distribute the load on the different possible paths. SimpleNet is composed of 8 nodes and 9 bi-directional links with a bandwidth of 10 Mbit/s and propagation delay of 1 msec. The topology is shown in Figure 3.1.
- *NSFNET* (2.2, 0.8, 14) is the old USA T1 backbone (1987). NSFNET is a WAN composed of 14 nodes and 21 bi-directional links with a bandwidth of 1.5 Mbit/s. Its topology is shown in Figure 3.2. Propagation delays range from 4 to 20 msec. NSFNET is a well balanced network.
- *NTTnet* (6.5, 3.8, 57) is the major Japanese backbone. NTTnet is the NTT (Nippon Telephone and Telegraph company) fiber-optic corporate backbone. NTTnet is a 57

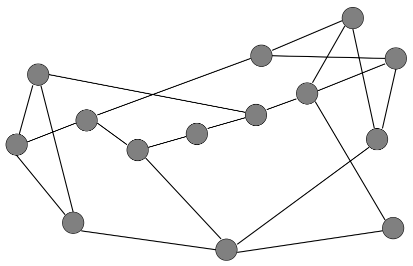


Figure 3.2: *NSFNET.* Each edge in the graph represents a pair of directed links. Link bandwidth is 1.5 Mbit/sec, propagation delays range from 4 to 20 msec.

nodes, 162 bi-directional links network. Link bandwidth is of 6 Mbit/sec, while propagation delays range around 1 to 5 msec. The topology is shown in Figure 3.3. NTTnet is not a well balanced network.

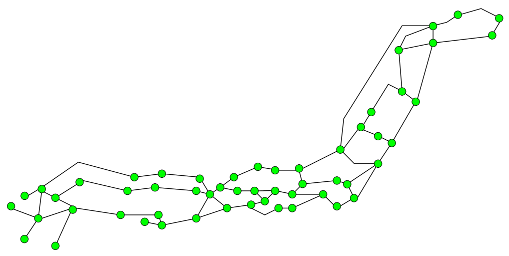


Figure 3.3: *NTTnet.* Each edge in the graph represents a pair of directed links. Link bandwidth is 6 Mbit/sec, propagation delays range from 1 to 5 msec.

- *Random Networks* (4.7, 1.8, 100) and (5.5, 2.1, 150) are randomly generated networks such that the level of connectivity of each node has been forced to lie between 2 and 5. The values for the mean shortest path distance and the variance of this average are averages over 10 randomly generated networks that have been used for the experiments. Every bi-directional link has the same bandwidth of 1 Mbit/sec, while the propagation delays have been generated in a random way uniformly over the interval [0.01, 0.001].

All the networks are simulated with zero link-fault and node-fault probabilities, local node buffers of 1 Gbit capacity, and data packets maximum time to live (TTL) set to 15 sec.

3.2.2 Traffic patterns

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

Sessions over the network can be characterized by their inter-arrival time distribution and by their geographical distribution. The latter is controlled by the probability assigned to each node to be selected as a session start or end-point.

We considered three basic patterns for the temporal distribution of the sessions, and three for their spatial distribution.

Temporal distributions:

- *Poisson* (P): for each node a Poisson process is defined which regulates the arrival of new sessions, i.e., sessions inter-arrival times are negative exponentially distributed.
- *Fixed* (F): at the beginning of the simulation, for each node, a fixed number of one-to-all sessions is set up and left constant for the remainder of the simulation.
- *Temporary* (TMPHS): a temporary, heavy load, traffic condition is generated turning on some nodes that act like hot spots (see below).

Spatial distributions:

- *Uniform* (U): the assigned temporal characteristics for session arrivals are set identically for all the network nodes.
- *Random* (R): in this case, the assigned temporal characteristics for session arrivals are set in a random way over the network nodes.
- *Hot Spots* (HS): some nodes behave as hot spots, concentrating a high rate of input/output traffic. A fixed number of sessions are opened from the hot spots to all the other nodes.

General traffic patterns have been obtained combining the above temporal and spatial characteristics. Therefore, for example, UP traffic means that, for each node, an identical Poisson process is regulating the arrival of new sessions, while in the RP case the process is different for each node, and UP-HS means that a Hot Spots traffic model is superimposed to a UP traffic.

Concerning the shape of the bit stream generated by each session, we consider two basic types:

- *Constant Bit Rate* (CBR): the per-session bit rate is maintained fixed. Examples of applications of CBR streams are the voice signal in a telephone network, which is converted into a stream of bits with a constant rate of 64 Kbit/sec, and the MPEG1 compression standard, which converts a video signal in a stream of 1.5 Mbit/sec.
- *Generic Variable Bit Rate* (GVBR): the per-session generated bit rate is time varying. The term GVBR is a broad generalization of the VBR term normally used to designate a bit stream with a variable bit rate but with known average characteristics and expected/admitted fluctuations.[2] Here, a GVBR session generates packets whose sizes and inter-arrival times are variable and follow a negative exponential distribution. The information about these characteristics is never directly used by the routing algorithms, like in IP-based networks.

The values we used in the experiments to shape traffic patterns are "reasonable" values for session generations and data packet production taking into consideration current network usage and computing power. The mean of the packet size distribution has been set to 4096 bits in all the experiments. Basic temporal and spatial distributions have been chosen to be representative of a wide class of possible situations that can be arbitrarily composed to generate a meaningful subset of real traffic patterns.

3.2.3 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 in best-effort connection-less networks, that is, 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* (correctly delivered bits/sec), *delay distribution* for data packets (sec), and *network capacity*

[2]The knowledge about the characteristics of the incoming CBR or VBR bit streams is of fundamental importance in networks able to deliver Quality of Service. It is only on the basis of this knowledge that the network can accept/refuse the session requests, and, in case of acceptance, allocate/reserve necessary resources.

usage (for data and routing packets), expressed as the sum of the used link capacities divided by the total available link capacity.

It is worth to remark that even if delay and throughput can be considered as correlated variables, it is meaningful to consider them as two separate metrics. In fact, let consider the following situations. If the time interval over which the throughput is observed is allowed to grow asymptotically, then the delay would not affect the throughput, the system would act as a pipeline where the transients would not affect the performance integrate over the "infinite" time interval. In this case it would suffice to observe the delay. If the considered time interval is not allowed to grow asymptotically, then a routing determining big delays could consequently determined a poor throughput too. "Could" because, again, the system could act as a long pipeline working at regime: the delay would be high but the throughput would be maximal. Anyway, if the pipeline is not full at the beginning of the observations, then likely big delays would imply a decrease in throughput if the initial and final transient phases are not negligible with respect to the length of the considered time interval. In even more realistic situations, as those considered in this thesis, node buffers can get full, causing data packet to be dropped off and/or stopping local applications from generating new data packets. In this cases a more severe reduction in throughput will be evidenced. Notice that there is a minimal difference for what concern delays between the situation when buffers are always almost full but never get completely full and the situation when buffers get often full. On the contrary, these two situations can produce a significantly different throughput. Even worse, in a TCP-based system the dropping packet situation (as well as the extinction of a data packet because of expired time-to-live) would cause the generation of a back packet to acknowledge the situation to the source, generating additional spurious traffic that would have an even more negative impact on the situation, for both delays and throughput.

3.2.4 Routing algorithms parameters

All the algorithms used have a collection of parameters to be set. Common parameters are routing packet size and elaboration time. Settings for these parameters are shown in table 3.1. These parameters have been assigned to values used in previous simulation

	AntNet	OSPF & SPF	BF	Q-R & PQ-R
Packet size (byte)	$24 + 8N_h$	$64 + 8\lvert\mathcal{N}_n\rvert$	$24 + 12N$	12
Packet elaboration time (msec)	3	6	2	3

Table 3.1: Routing packets characteristics for the implemented algorithms (except for the Daemon algorithm, which does not generate routing packets). N_h is the incremental number of hops made by the forward ant, $|\mathcal{N}_n|$ is the number of neighbors of node n, and N is the number of network nodes.

works (Alaettinoğlu et al., 1992) and/or on the basis of heuristic evaluations taking into consideration information encoding schemes and currently available computing power (e.g., the size for forward ants has been determined as the same size of a BF packet plus 8 bytes for each hop to store the information about the node address and the elapsed time). Concerning the other main parameters, specific for each algorithm, for the AntNet competitors we used the best settings we could find in the literature and/or we tried to tune the parameters as much as possible to obtain better results. For OSPF, SPF, and BF, the length of the time interval between consecutive routing information broadcasts and the length of the time window to average link costs are the same, and they are set to 0.8 or 3 seconds, depending on the experiment for SPF and BF, and to 30 seconds for OSPF. Link costs inside each window are assigned as the weighted sum between the arithmetic average over the window and the exponential average with decay factor equal to 0.9. The obtained values are discretized over a linear scale saturated between 1 and 20, with slope set to 20 and maximum admitted variation equal to 1. For Q-R and PQ-R the transmission of routing information is totally data-driven. The learning and adaptation rate we used were the same as used by the algorithm's authors (Boyan & Littman, 1994; Choi & Yeung, 1996).

Concerning AntNet, we observed that the algorithm is very robust to internal parameters tuning. We did not finely tune the parameter set, and we used the same set of values for all the different experiments we ran. Most of the settings we used have been previously given in the text at the moment the parameter was discussed and they are not reported in this section. The ant generation interval at each node was set to 0.3 seconds. In Section 3.3.5 it will be shown the robustness of AntNet with respect to this parameter. Regarding the parameters of the statistical model, the value of η, weighting the number of the samples considered in the model (Equation 2.1), has been set to 0.005, the c factor for the expression of $|\mathcal{W}|_{max}$ (sect. 2.1.2) has been put equal to 0.3, and the confidence level factor z (sect. 2.1.2) equal to 1.70, meaning a confidence level of approximately 0.95.

3.3 Results

Experiments reported in this section compare AntNet and AntNet-FA with the competing routing algorithms described in Section 3.1. We studied the performance of the algorithms for increasing traffic load, examining the evolution of the network status toward a saturation condition, and for temporary saturation conditions.

- Under *low load conditions*, all algorithms tested have similar performance. In this case, also considering the huge variability in the possible traffic patterns, it is very hard to assess whether an algorithm is significantly better than another or not.
- Under *high, near saturation, loads*, all the tested algorithms are able to deliver the offered throughput in a quite similar way, that is, in most of the cases all the generated traffic is routed without big losses. On the contrary, the study of packet delay distributions shows remarkable differences among the different algorithms. To present simulation results regarding packet delays we decided either to report the whole empirical distribution or to use the 90-th percentile statistic, which allows one to compare the algorithms on the basis of the upper value of delay they were able to keep the 90% of the correctly delivered packets. In fact, packet delays can be spread over a wide range of values. This is an intrinsic characteristics of data networks: packet delays can range from very low values for sessions open between adjacent nodes connected by fast links, to much higher values in the case of sessions involving nodes very far apart connected by many slow links. Because of this, very often the empirical distribution of packet delays cannot be meaningfully parametrized in terms of mean and variance, and the 90-th percentile statistic, or still better the whole empirical distribution, are much more meaningful.
- Under *saturation* there are packet losses and/or packet delays that become too big, cause all the network operations to slow down. Therefore, saturation has to be only a temporary situation. If it is not, structural changes to the network characteristics, like adding new and faster connection lines, rather than improvements of the routing algorithm, should be in order. For these reasons, we studied the responsiveness of the algorithms to traffic loads causing only a temporary saturation.

All reported data are averaged over 10 trials lasting 1000 virtual seconds of simulation time. One thousand seconds represents a time interval long enough to expire all transients and to get enough statistical data to evaluate the behavior of the routing algorithm. Before being fed with data traffic, the algorithms are given 500 preliminary simulation seconds with no data traffic to build initial routing tables. In this way, each algorithm builds the routing tables according to its own "vision" about minimum cost paths. Results for throughput are reported as average values without an associated measure of variance. The inter-trial variability is in fact always very low, a few percent of the average value.

Parameter values for traffic characteristics are given in the Figure captions with the following meaning (see also previous section): MSIA is the mean of the sessions inter-arrival time distribution for the Poisson (P) case, MPIA stands for the mean of the packet inter-arrival time distribution. In the CBR case, MPIA indicates the fixed packet production rate. HS is the number of hot-spots nodes and MPIA-HS is the equivalent of MPIA for the hot-spot sessions. In the following, when not otherwise explicitly stated, the shape of the session bit streams is assumed to be of GVBR type.

Results for *throughput* and *packet delays* for all the considered network topologies are described in the three following subsections. Results concerning the *network resources utilization* are reported in Section 3.3.5.

3.3.1 SimpleNet

Experiments with SimpleNet were designed to study how the different algorithms manage to distribute the load on the different possible paths. In these experiments, all the traffic, of F-CBR type, is directed from node 1 to node 6 (see Figure 3.1), and the traffic load has been set to a value higher than the capacity of a single link, so that it cannot be routed efficiently on a single path.

Results regarding throughput (Figure 3.4a) in this case strongly discriminate among the algorithms. The type of the traffic workload and the small number of nodes determined significant differences in throughput. AntNet is the only algorithm able to deliver almost all the generated data traffic: its throughput after a short transient phase approaches very closely the level of that delivered by the Daemon algorithm. PQ-R attains a steady value approximately 15% inferior to that obtained by AntNet. The other algorithms behave very poorly, stabilizing on values of about 30% inferior to those provided by AntNet. In this case, it is rather clear that AntNet is the only algorithm able to exploit at best all the three available paths (1-8-7-6, 1-3-5-6, 1-2-4-5-6) to distribute the data traffic without inducing counterproductive oscillations. The utilization of the routing tables in a probabilistic way also by data packets in this case plays a fundamental role in achieving higher quality results. Results for throughput are confirmed by those for packet delays, reported in the graph of Figure 3.4b. The differences in the empirical distributions for packet delays reflect approximatively the same proportions as evidenced in the throughput case.

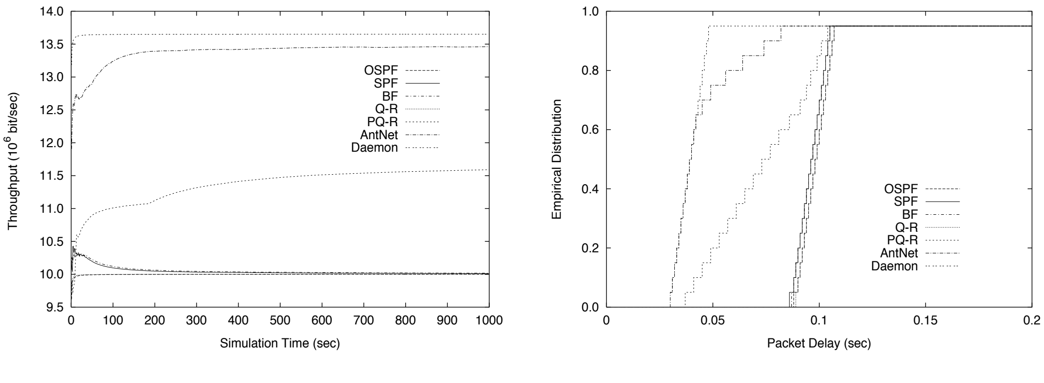
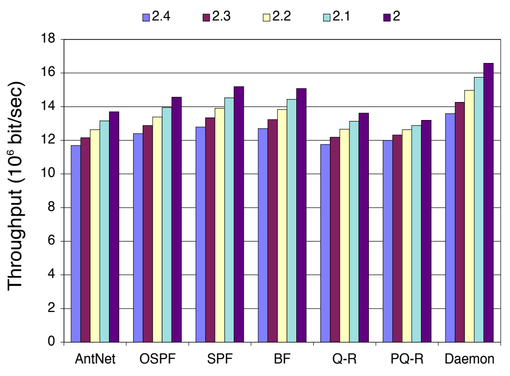


Figure 3.4: *SimpleNet*: Comparison of algorithms for F-CBR traffic directed from node 1 to node 6 (MPIA = 0.0003 sec). (a) Throughput, and (b) packet delays empirical distribution.

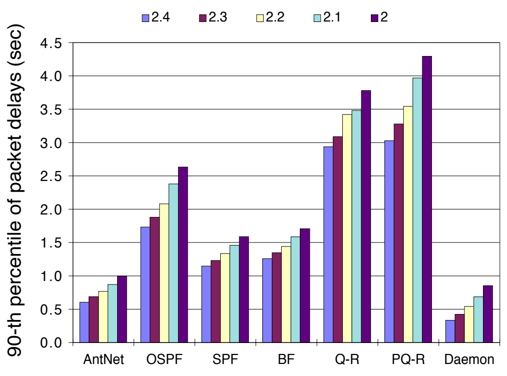
3.3.2 NSFNET

We carried out a wide range of experiments on NSFNET using UP, RP, UP-HS and TMPHS-UP traffic patterns. In all the cases considered, differences in throughput are of minor importance with respect to those shown by packet delays. For each one of the UP, RP and UP-HS cases we ran five distinct groups of ten trial experiments, gradually increasing the generated workload (in terms of reducing the session inter-arrival time). As explained above, we studied the behavior of the algorithms when moving the traffic load towards a saturation region.

In the UP case, differences in throughput (Figure 3.5a) are small: the best performing algorithms are BF and SPF, which can attain performance of only about 10% inferior to those of Daemon and of the same amount better than those of AntNet, Q-R and PQ-R,[3] while OSPF behaves slightly better than these last ones. Concerning delays (Figure 3.5b) the situation is rather different, as can be seen by the fact that all the algorithms but AntNet have been able to produce a slightly higher throughput at the expenses of much worse results for packet delays. This trend in packet delays was confirmed by all the experiments we ran. OSPF, Q-R and PQ-R show really poor results (delays of order 2 or more seconds are very high values, even if we are considering the 90-th percentile of the distribution), while BF and SPF behave in a similar way with performance of order 50% worse than those obtained by AntNet and of order 65% worse than Daemon.



(a)



(b)

Figure 3.5: *NSFNET*: Comparison of algorithms for increasing load for UP traffic. The load is increased reducing the MSIA value from 2.4 to 2 seconds (MPIA = 0.005 sec). (a) Throughput, and (b) 90-th percentile of the packet delays empirical distribution.

In the RP case (Figure 3.6a), throughputs generated by AntNet, SPF and BF are very similar, although AntNet has a slightly better performance. OSPF and PQ-R behave only slightly worse while Q-R is the worst algorithm. Daemon is able to obtain only slightly better results than AntNet. Again, looking at packet delays results (Figure 3.6b) OSPF, Q-R and PQ-R perform very badly, while SPF shows results a bit better than those of BF but of order 40% worse than those of AntNet. Daemon is in this case far better, which indicates that the testbed was very difficult.

For the case of UP-HS load, throughputs (Figure 3.7a) for AntNet, SPF, BF, Q-R and Daemon are very similar, while OSPF and PQ-R clearly show much worse results. Again (Figure 3.7b), packet delays results for OSPF, Q-R and PQ-R are much worse than those of the other algorithms (they are so much worse that they do not fit in the scale chosen to make clear differences among the other algorithms). AntNet is still the best performing algorithm. In this case, differences with SPF are of order 20% and of 40% with respect to BF. Daemon performs about 50% better than AntNet and scales much better than AntNet, which, again, indicates the testbed was rather difficult.

[3]It is worth remarking that in these and in some of the experiments presented in the following, PQ-R's performance is slightly worse than that of Q-R. This seems to be in contrast with the results presented by the PQ-R's authors in the article where they introduced PQ-R (Choi & Yeung, 1996). We think that this behavior is due to the fact that (i) their link recovery rate matches a discrete-time system while in our simulator time is a continuous variable, and (ii) the experimental and simulation conditions are rather different (in their article it is not specified the way they produced traffic patterns and they did not implement a realistic network simulator).

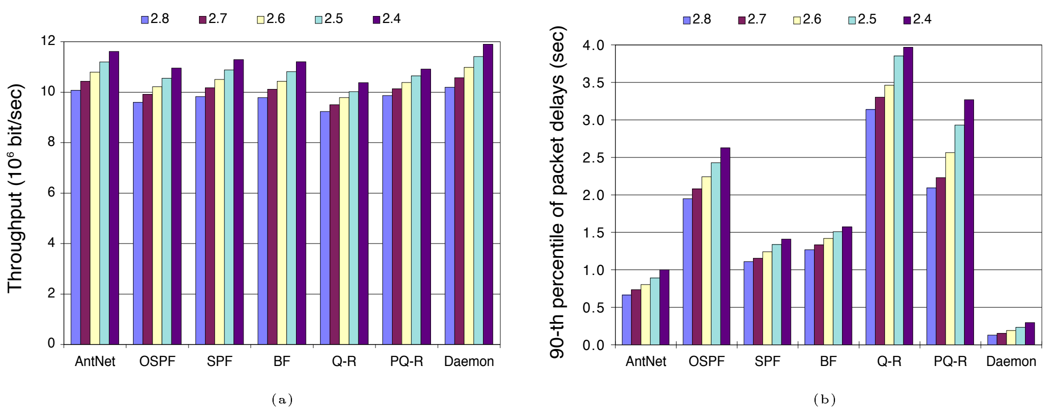


Figure 3.6: *NSFNET*: Comparison of algorithms for increasing load for RP traffic. The load is increased reducing the MSIA value from 2.8 to 2.4 seconds (MPIA = 0.005 sec). (a) Throughput, and (b) 90-th percentile of the packet delays empirical distribution.

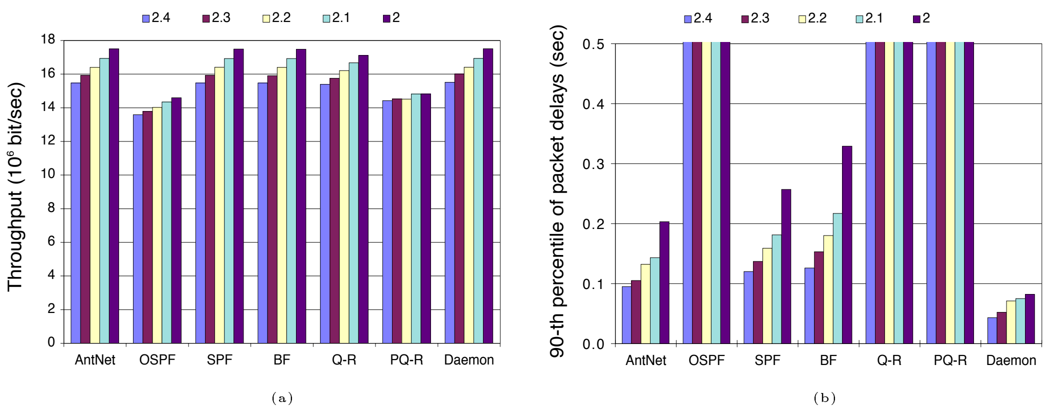


Figure 3.7: *NSFNET*: Comparison of algorithms for increasing load for UP-HS traffic. The load is increased reducing the MSIA value from 2.4 to 2.0 seconds (MPIA = 0.3 sec, HS = 4, MPIA-HS = 0.04 sec). (a) Throughput, and (b) 90-th percentile of the packet delays empirical distribution.

The last graph for NSFNET shows how the algorithms behave in the case of a TMPHS-UP situation (Figure 3.8). At time $t = 400$ four hot spots are turned on and superimposed to the existing light UP traffic. The transient is kept on for 120 seconds. In this case, only one, typical, situation is reported in detail to show the answer curves. Reported values are the "instantaneous" values for throughput and packet delays computed as the average over 5 seconds moving windows. All algorithms have a similar very good performance as far as throughput is concerned, except for OSPF and PQ-R, which lose a few percent of the packets during the transitory period. The graph of packet delays confirms previous results: SPF and BF have a similar behavior, about 20% worse than AntNet and 45% worse than Daemon. The other three algorithms show a big out-of-scale jump, being not able to properly dump the sudden load increase.

3.3.3 NTTnet

The same set of experiments run on the NSFNET have been repeated on NTTnet. In this case the results are even sharper than those obtained with NSFNET: AntNet performance is much better that of all its competitors.

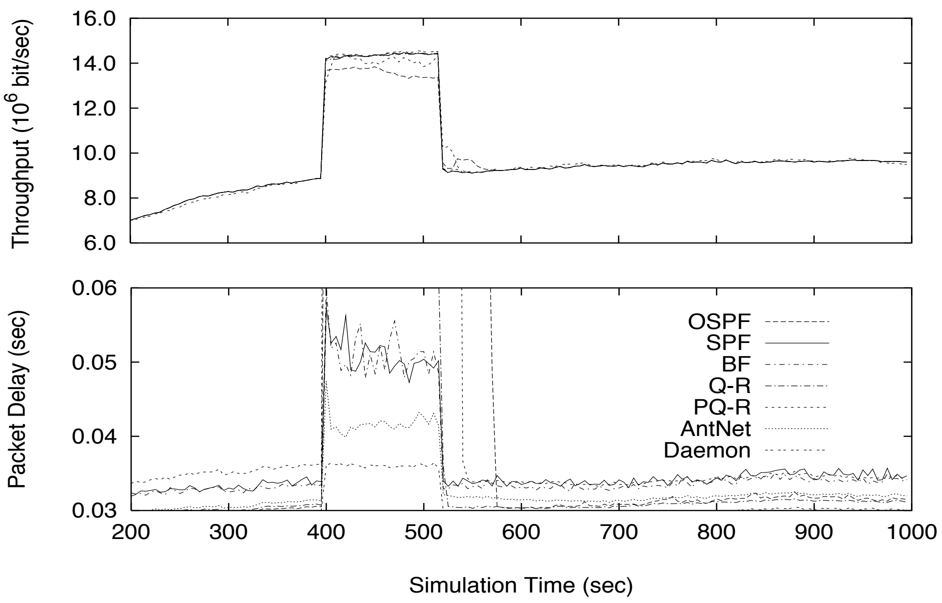


Figure 3.8: *NSFNET*: Comparison of algorithms for transient saturation conditions with TMPHS-UP traffic (MSIA = 3.0 sec, MPIA = 0.3 sec, HS = 4, MPIA-HS = 0.04). (a) Throughput, and (b) packet delays averaged over 5 seconds moving windows.

For the UP, RP and UP-HS cases, differences in throughput are not significant (FIGURES-JAIR 3.9a, 3.10a and 3.11a). All the algorithms, with the OSPF exception, practically behave in the same way as the Daemon algorithm. Concerning delays (FIGURES-JAIR 3.9b, 3.10b and 3.11b), differences between AntNet and each of its competitors are of one order of magnitude. AntNet keeps delays at low values, very close to those obtained by Daemon, while SPF, BF, Q-R and PQ-R perform poorly and OSPF completely collapses.
In the UP and RP cases (Figures 3.9b and 3.10b) SPF and BF performs similarly, even if SPF shows slightly better results, and about 50% better than Q-R and PQ-R.

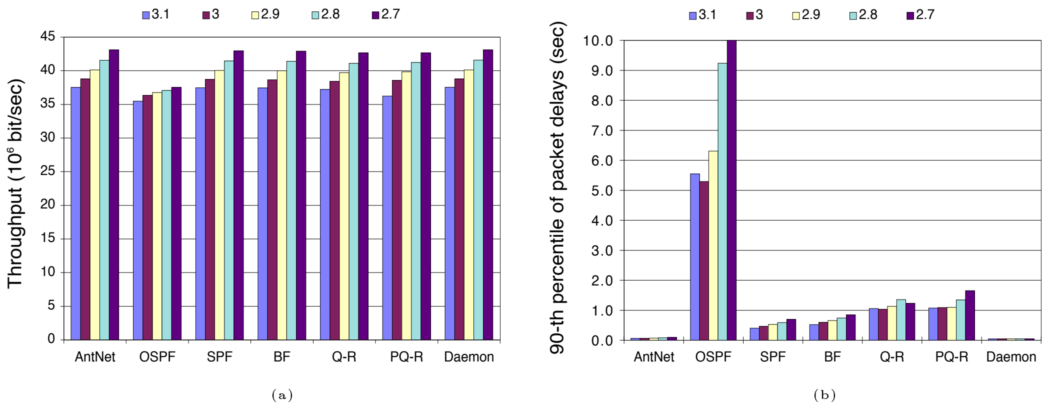
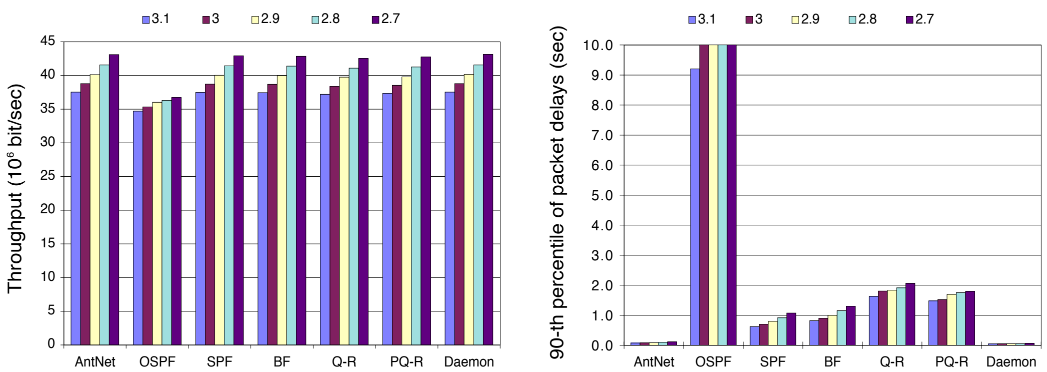


Figure 3.9: *NTTnet*: Comparison of algorithms for increasing load for UP traffic. The load is increased reducing the MSIA value from 3.1 to 2.7 seconds (MPIA = 0.005 sec). (a) Throughput, and (b) 90-th percentile of the packet delays empirical distribution.

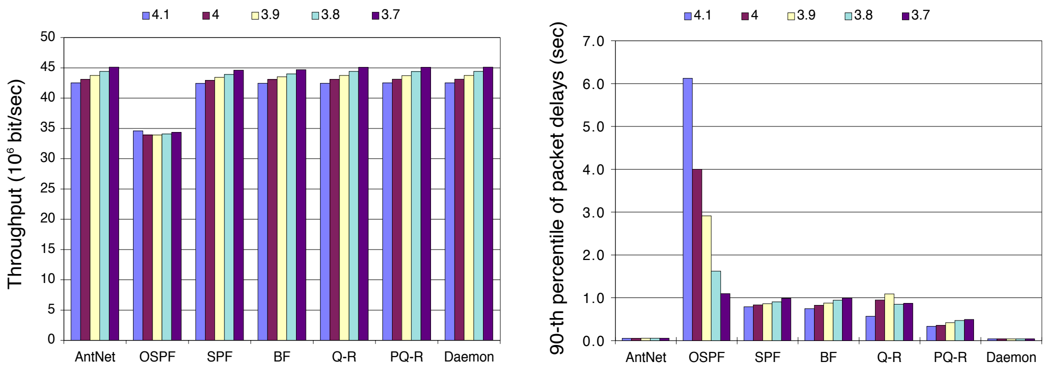
In the UP-HS case, again, SPF and BF show similar results, while Q-R performs comparably but in a much more irregular way and PQ-R can keep delays about 30% lower. OSPF, which is the worse algorithm in this case, shows an interesting behavior. The increase in the generated data throughput determines a decrease or a very slow increase in the delivered throughput while delays decrease (Figure 3.11a and 3.11b). In this case the load was too



(a) (b)

Figure 3.10: *NTTnet*: Comparison of algorithms for increasing load for RP traffic. The load is increased reducing the MSIA value from 3.1 to 2.7 seconds (MPIA = 0.005 sec). (a) Throughput, and (b) 90-th percentile of the packet delays empirical distribution.

high for the algorithm and the balance between the two, conflicting, objectives, throughput and packet delays, showed an inverse dynamics: having a lot of packet losses made it possible for the surviving packets to obtain lower trip delays.



(a) (b)

Figure 3.11: *NTTnet*: Comparison of algorithms for increasing load for UP-HS traffic. The load is increased reducing the MSIA value from 4.1 to 3.7 seconds (MPIA = 0.3 sec, HS = 4, MPIA-HS = 0.05 sec). (a) Throughput, and (b) 90-th percentile of the packet delays empirical distribution.

The TMPHS-UP experiment (Figure 3.12), concerning sudden load variation, confirms the previous results. OSPF is not able to follow properly the variation both for throughput and delays. All the other algorithms are able to follow the sudden increase in the offered throughput, but only AntNet (and Daemon) show a very regular behavior. Differences in packet delays are striking. AntNet performance is very close to those obtained by Daemon (the curves are practically superimposed at the scale used in the Figure). Among the other algorithms, SPF and BF are the best ones, although their response is rather irregular and, in any case, much worse than AntNet's. OSPF and Q-R are out-of-scale and show a very delayed recovering curve. PQ-R, after a huge jump, which takes the graph out-of-scale in the first 40 seconds after hot spots are turned on, shows a trend approaching those of BF and SPF.

3.3.4 Random networks

Up to now we tested the behavior of the routing algorithms on relatively small sized and somehow regular networks. In this section we report experimental results for the case of randomly generated networks with a number of nodes considerably bigger than in the previous

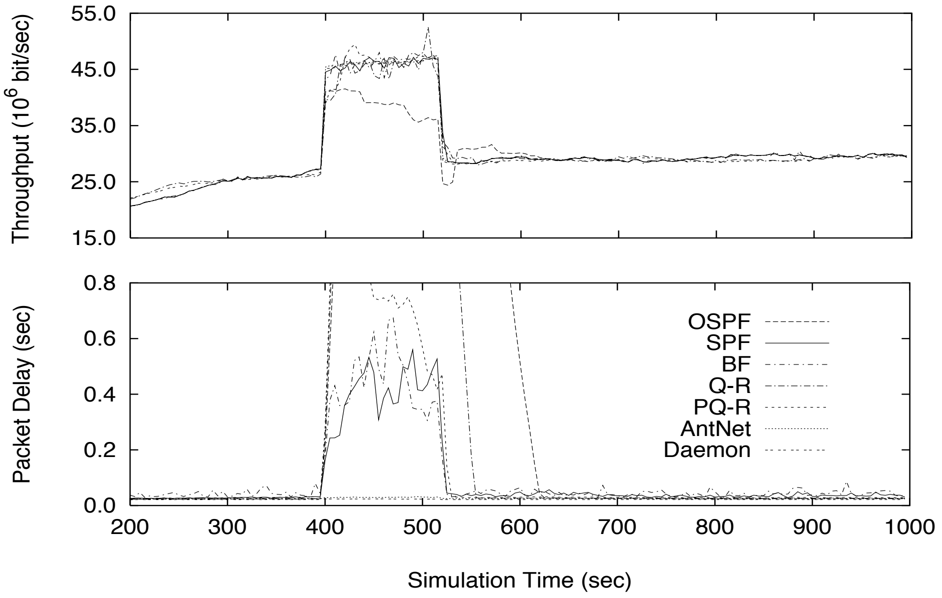


Figure 3.12: *NTTnet*: Comparison of algorithms for transient saturation conditions with TMPHS-UP traffic (MSIA = 4.0 sec, MPIA = 0.3 sec, HS = 4, MPIA-HS = 0.05). (a) Throughput, and (b) packet delays averaged over 5 seconds moving windows.

cases. Moreover, we report results also concerning the performance of AntNet-FA. Reported data are the average over 10 trials, where for each trial a different randomly generated network has been used.

Figure 3.13 shows results for a set of 100-node randomly generated networks with UP load. In this case all the algorithms were able to deliver the same throughput, while, once again, differences in the distribution of packet delays are striking. AntNet-FA is by far the best performing algorithm, followed by AntNet, while all the competitors perform about 30%-40% worse.

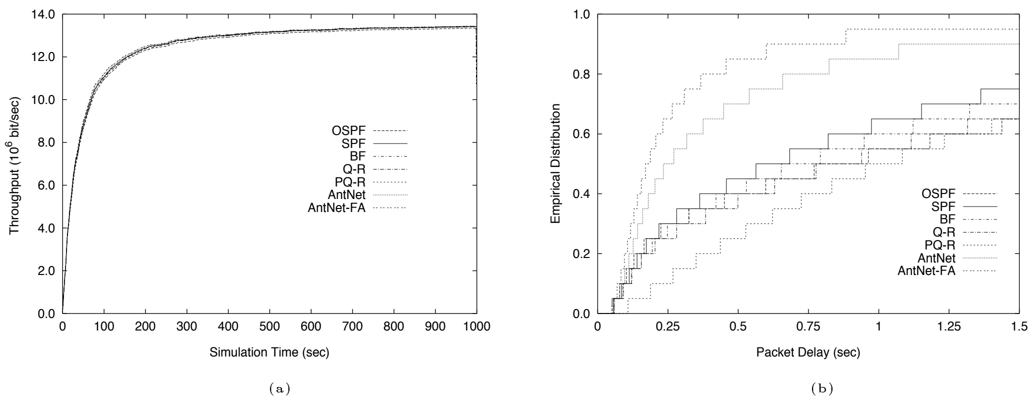


Figure 3.13: *100-Nodes Random Networks*: Comparison of algorithms (including AntNet-FA) for a heavy UP traffic. Average over 10 trials using a different randomly generated 100-node network in each trial (MSIA=15.0, MPIA=0.005). (a) Throughput, and (b) 90-th percentile of the packet delays empirical distribution.

Figure 3.14 reports experimental results for a set of 150-node randomly generated networks with a very heavy RP load. In this case, there are meaningful differences among the algorithms' behavior both for throughput and packet delays. Only AntNet, AntNet-FA and SPF are able to follow the generated throughput without losses, OSPF behaves only slightly worse, while all the other algorithms can deliver a throughput about 35% lower. Concerning packet delays, AntNet-FA is still by far the best performing algorithm. AntNet is the second best one, but it keeps delays much higher than AntNet-FA, about four times higher considering the 90th percentile. SPF keeps delays much higher than AntNet-FA, and about 60% higher than AntNet on the 85th percentile. BF follows, but it had much worse performance on throughput. OSPF, Q-R and PQ-R perform rather poorly.

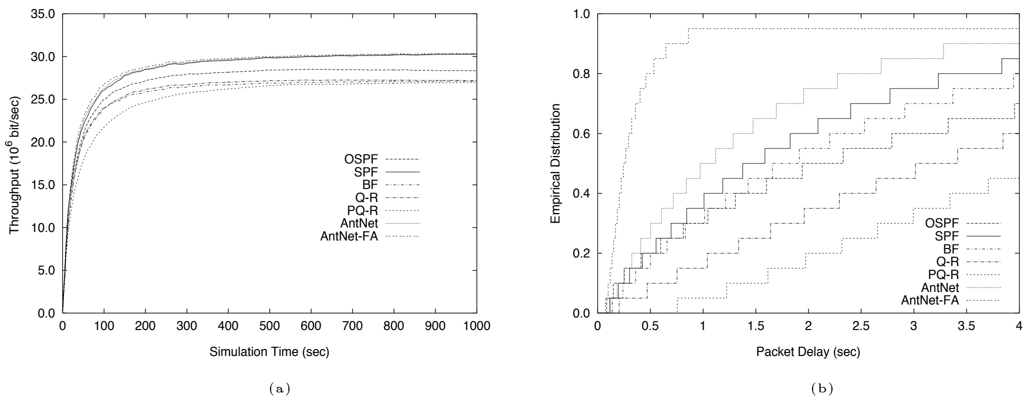


Figure 3.14: *100-Nodes Random Networks*: Comparison of algorithms (including AntNet-FA) for a heavy RP traffic. Average over 10 trials using a different randomly generated 100-node network in each trial (MSIA=10.0, MPIA=0.005). (a) Throughput, and (b) 90-th percentile of the packet delays empirical distribution.

These results globally confirm the AntNet's better performance with respect to the considered competitors even on larger network instances. If this result is very positive and somehow reassuring, it is even more interesting to notice that actually AntNet-FA performs much better than AntNet itself. AntNet-FA showed always the best performance and the difference with the performance obtained by all the other algorithms (AntNet included) increases with the size of the test networks.

The better performance of AntNet-FA with respect to its ancestor AntNet can be understood in terms of its higher reactivity. In AntNet-FA forward ants do not wait in the data queues. In this way, the information is collected and propagated faster, and it is more up-to-date with respect to the current network status, even if it is based on somehow raw estimates. These characteristics become more and more important as the diameter of the network grows and paths become longer and longer. In these cases, AntNet can present very long delays in gathering and releasing traffic information across the network, making completely out-of-date the information it uses to update the routing tables.

3.3.5 Routing overhead

Table 3.2 reports results concerning the overhead generated by routing packets. For each algorithm the network load generated by the routing packets is reported as the ratio between the bandwidth occupied by the routing packets and the total available network bandwidth. Each row in the table refers to a previously discussed experiment (Figs. 3.4 to 3.7 and 3.9 to 3.11). Routing overhead is computed for the experiment with the heaviest load in the increasing load series.

All data are scaled by a factor of 10^{-3}. The data in the table show that the routing overhead is negligible for all the algorithms with respect to the available bandwidth. Among the adaptive algorithms, BF shows the lowest overhead, closely followed by SPF. AntNet generates a slightly bigger consumption of network resources, but this is widely compensated by the much higher performance it provides. Q-R and PQ-R produce an overhead a bit higher

	AntNet	OSPF	SPF	BF	Q-R	PQ-R
SimpleNet - F-CBR	0.33	0.01	0.10	0.07	1.49	2.01
NSFNET - UP	2.39	0.15	0.86	1.17	6.96	9.93
NSFNET - RP	2.60	0.15	1.07	1.17	5.26	7.74
NSFNET - UP-HS	1.63	0.15	1.14	1.17	7.66	8.46
NTTnet - UP	2.85	0.14	3.68	1.39	3.72	6.77
NTTnet - RP	4.41	0.14	3.02	1.18	3.36	6.37
NTTnet - UP-HS	3.81	0.14	4.56	1.39	3.09	4.81

Table 3.2: *Routing Overhead*: ratio between the bandwidth occupied by the routing packets and the total available network bandwidth. All data are scaled by a factor of 10^{-3}.

than that of AntNet. The routing load caused by the different algorithms is a function of many factors, specific of each algorithm. Q-R and PQ-R are data-driven algorithms: if the number of data packets and/or the length of the followed paths (because of topology or bad routing) grows, so will the number of generated routing packets. BF, SPF and OSPF have a more predictable behavior: the generated overhead is mainly function of the topological properties of the network and of the generation rate of the routing information packets. AntNet produces a routing overhead depending on the ants generation rate and on the length of the paths they travel.

The ant traffic can be roughly characterized as a collection of additional traffic sources, one for each network node, producing very small packets (and related acknowledgement packets) at constant bit rate with destinations matching the offered data traffic. On average ants will travel over rather "short" paths and their size will grow of only 8 bytes at each hop. Therefore, each "ant routing traffic source" represents a very light additional traffic source with respect to network resources when the ant launching rate is not excessively high. In Figure 3.15, the sensitivity of AntNet with respect to the ant launching rate is reported.

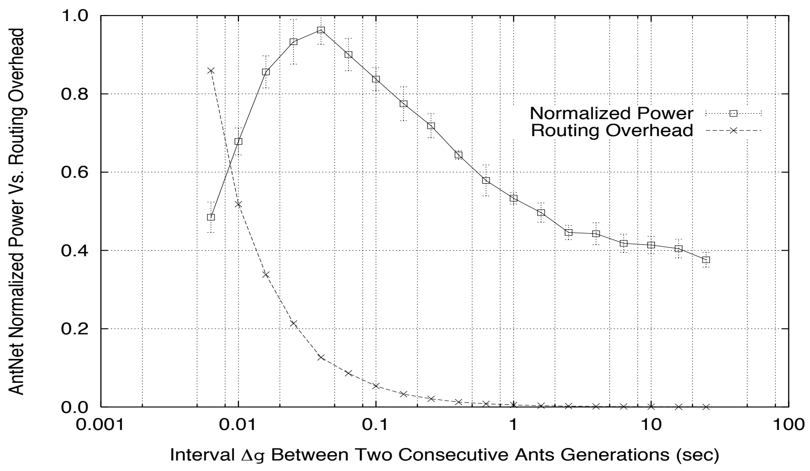


Figure 3.15: *AntNet* normalized power *vs.* routing overhead. Power is defined as the ratio between delivered throughput and packet delay.

For a sample case of a UP data traffic model on NSFNET (previously studied in Figure 3.5) the interval Δg between two consecutive ant generations is progressively decreased (Δg is the same for all nodes). Δg values are sampled at constant intervals over a logarithmic scale ranging from about 0.006 to 25 seconds. The lower, dashed, curve interpolates the generated routing overhead expressed, as before, as the fraction of the available network bandwidth used by routing packets. The upper, solid, curve plots the data for the obtained power normalized to its highest value, where the power is defined as the ratio between the delivered throughput and the packet delay. The value used for delivered throughput is the throughput value at time 1000 averaged over ten trials, while for packet delay we used the 90-th percentile of the empirical distribution.

In the figure, we can see how an excessively small Δg causes an excessive growth of the routing overhead, with consequent reduction of the algorithm power. Similarly, when Δg is too big, the power slowly diminishes and tends toward a plateau because the number of ants is not enough to generate and maintain up-to-date statistics of the network status. In the middle of these two extreme regions a wide range of Δg intervals gives raise to similar, very good power values, while, at the same time, the routing overhead quickly falls down toward negligible values. This figure strongly confirms our previous assertion about the robustness of AntNet's internal parameter settings.

Chapter 4

Discussion

In this section the properties and characteristics of AntNet are carefully analyzed and discussed. For reasons already made clear in the previous chapters, the discussion is here limited to AntNet. Anyway, most of the results of this chapter apply also to AntNet-FA and to AntNet++, since basically they incorporate new additional features on top of the AntNet's ones, without disrupting the properties of these latters.

In AntNet, the continual on-line construction of the routing tables is the emergent result of a collective learning process. In fact, each forward-backward agent pair is complex enough to find a good route and to adapt the routing tables for a single source-destination path, but it cannot solve the global routing optimization problem. It is the interaction between the agents that determines the emergence of a global effective behavior from the network performance point of view. Ants cooperate in their problem-solving activity by communicating in an indirect and non-coordinated way. Each agent acts independently. Good routes are discovered by applying a policy that is a function of the information accessed through the network nodes visited, and the information collected about the route is eventually released on the same nodes. Therefore, the inter-agent communication is mediated in an explicit and implicit way by the "environment", that is, by the node's data structures and by the traffic patterns recursively generated by the data packets' utilization of the routing tables. This communication paradigm, called stigmergy, matches well the intrinsically distributed nature of the routing problem. Cooperation among agents goes on at two levels: (a) by modifications of the routing tables, and (b) by modifications of local models that determine the way the ants' performance is evaluated. Modifications of the routing tables directly affect the routing decisions of following ants towards the same destination, as well as the routing of data, which, in turn, influences the rate of arrival of other ants towards any destination. It is interesting to remark that the used stigmergy paradigm makes the AntNet's mobile agents very flexible from a software engineering point of view. In this perspective, once the interface with the node's data structure is defined, the internal policy of the agents can be transparently updated. Also, the agents could be exploited to carry out multiple concurrent tasks (e.g., collecting information for distributed network management using an SNMP-like protocol or for Web data-mining tasks).

As shown in the previous section, the results we obtained with the above stigmergetic model of computation are excellent. In terms of throughput and average delay, AntNet performs better than both classical and recently proposed routing algorithms on a wide range of experimental conditions. Although this is very interesting *per se*, in the following we try to justify AntNet superior performance by highlighting some of its characteristics and by comparing them with those of the competing algorithms. We focus on the following main aspects:

- AntNet can be seen as a particular instance of a parallel Monte Carlo simulation system with biased exploration. All the other algorithms either do not explore the net or their exploration is local and tightly connected to the flux of data packets.

- The information AntNet maintains at each node is more complete and organized in a less critical way than that managed by the other algorithms.
- AntNet does not propagate local estimates to other nodes, while all its competitors do. This mechanism makes the algorithm more robust to locally wrong estimates.
- AntNet uses probabilistic routing tables, which have the triple positive effect of better redistributing data traffic on alternative routes, of providing ants with a built-in exploration mechanism and of allowing the exploitation of the ants' arrival rate to assign cumulative reinforcements.
- It was experimentally observed that AntNet is much more robust than its competitors to the frequency with which routing tables are updated.
- The structure of AntNet allows one to draw some parallels with some well-known reinforcement learning (RL) algorithms. The characteristics of the routing problem, that can be seen as a distributed time-varying RL problem (see sect. 1.2.2), determines a departure of AntNet from the structure of classical RL algorithms.

These aspects of AntNet are discussed in more detail in the following.

4.1 An on-line Monte Carlo system with biased exploration

The AntNet routing system can be seen as a collection of mobile agents collecting data about the network status by concurrently performing on-line Monte Carlo simulations (Rubistein, 1981; Streltsov & Vakili, 1996). In Monte Carlo methods, repeated experiments with stochastic transition components are run to collect data about the statistics of interest. Similarly, in AntNet ants explore the network by performing random experiments (i.e., building paths from source to destination nodes using a stochastic policy dependent on the past and current network states), and collect on-line information on the network status. A built-in variance reduction effect is determined (i) by the way ants' destinations are assigned, biased by the most frequently observed data's destinations, and (ii) by the way the ants' policy makes use of current and past traffic information (that is, inspection of the local queues' status and probabilistic routing tables). In this way, the explored paths match the most interesting paths from a data traffic point of view, which results in a very efficient variance reduction effect in the stochastic sampling of the paths. Differently from usual off-line Monte Carlo systems, in AntNet the state space sampling is performed on-line, that is, the sampling of the statistics and the controlling of the non-stationary traffic process are performed concurrently.

This way of exploring the network concurrently with data traffic is very different from what happens in the other algorithms where, either there is no exploration at all (OSPF, SPF and BF), or exploration is both tightly coupled to data traffic and of a local nature (Q-R and PQ-R). Conveniently, as was shown in Section 3.3.5, the extra traffic generated by exploring ants is negligible for a wide range of values, allowing very good performance.

4.2 Information management at each network node

Key characteristics of routing algorithms are the type of information used to build/update routing tables and the way this information is propagated. All the algorithms (except the static OSPF) make use at each node of two main components: a local model $\mathcal{M}$ of some cost measures and a routing table $\mathcal{T}$. SPF and BF use $\mathcal{M}$ to estimate smoothed averages of the local link costs, that is, of the distances to the neighbor nodes. In this case, $\mathcal{M}$ is a local model maintaining estimates of only local components. In Q-R the local model is fictitious because the raw transition time is directly used as a value to update $\mathcal{T}$. PQ-R uses a slightly more sophisticated model with respect to Q-R, storing also a measure of the link utilization. All these algorithms propagate part of their local information to the other nodes, which, in turn, make use of it to update their routing tables and to build a global view of the network. In SPF and BF the content of each $\mathcal{T}$ is updated, at regular intervals, by a "memoryless strategy": the new entries do not depend on the old values, that are discarded. Therefore, the whole adaptive component of the routing system is represented by the model $\mathcal{M}$. Otherwise, in Q-R and PQ-R the adaptive content of $\mathcal{M}$ is almost negligible and the adaptive component of the algorithm is represented by the smoothed average carried out by the Q-learning-like rule. AntNet shows characteristics rather different from its competitors: its model $\mathcal{M}$ contains a memory-based local perspective of the global status of the network. The content of $\mathcal{M}$ allow the reinforcements to be weighted on the basis of a rich statistical description of the network dynamics as seen by the local node. These reinforcements are used to update the routing table, the other adaptive component maintained at the node. The $\mathcal{T}$ updates are carried out in an asynchronous way and as a function of their previous values. Moreover, while $\mathcal{T}$ is used in a straightforward probabilistic way by the data packets, traveling ants select the next node by using both $\mathcal{T}$, that is, an adaptive representation of the past policy, and a model of the current local link queues, that is, an instantaneous representation of the node status. It is evident that AntNet builds and uses more information than its competitors: two different memory-based components and an instantaneous predictor are used and combined at different levels. Moreover, in this way AntNet robustly redistributes among these completely local components the criticality of all the estimates and decisions.

4.3 Robustness to wrong estimates

As remarked above, AntNet, differently from its competitors, does not propagate local estimates to other nodes. Each node routing table is updated independently, by using local information and the ants' experienced trip time. Moreover, (i) each ant experiment affects only one entry in the routing table of the visited nodes, the one relative to the ant's destination, and, (ii) the local information is built from the "global" information collected by traveling ants, implicitly reducing in this way the variance in the estimates. These characteristics make AntNet particularly robust to wrong estimates. On the contrary, in all the other algorithms a locally wrong estimate will be propagated to all other nodes and will be used to compute estimates to many different destinations. How bad this is for the algorithm performance depends on how long the wrong estimate effect lasts. In particular, this will be a function of the time window over which estimates are computed for SPF and BF, and of the learning parameters for Q-R and PQ-R.

4.4 On the probabilistic use of routing tables to route data packets

All the tested algorithms but AntNet use deterministic routing tables.[1] In these algorithms, entries in the routing tables contain distance/time estimates to the destinations. These estimates can provide misleading information if the algorithm is not fast enough to follow the traffic fluctuations, as can be the case under heavy load conditions. Instead, AntNet routing tables have probabilistic entries that, although reflecting the goodness of a particular path choice with respect to the others available, do not force the data packets to choose the perceived best path. This has the positive effect of allowing a better balancing of the traffic load on different paths, with a resulting better utilization of the resources (as was shown in particular in the experiments with the SimpleNet). As remarked at the end of Section 2.1.1,

[1]Singh, Jaakkola, and Jordan (1994) showed that stochastic policies can yield higher performance than deterministic policies in the case of an incomplete access to the state information of the environment. In (Jaakkola, Singh, & Jordan, 1995), the same authors developed a Monte-Carlo-based stochastic policy evaluation algorithm, confirming the usefulness of the Monte-Carlo approach, used in AntNet too, to deal with incomplete information problems.

the intrinsic probabilistic structure of the routing tables and the way they are updated allow AntNet to exploit the ant's arrival rate as a way to assign implicit (cumulative) reinforcements to discovered paths. It is not obvious how the same effect could be obtained by using routing tables containing distance/time estimates and using this estimates in a probabilistic way. In fact, in this case each new trip time sample would modify the statistical estimate that would simply oscillate around its expected value without inducing an arrival-dependent cumulative effect.
Probabilistic routing tables provide some remarkable additional benefits: (a) they give to the ants a built-in exploration method in discovering new, possibly better, paths, and (b) since ants and data routing are independent in AntNet, the exploration of new routes can continue while, at the same time, data packets can exploit previously learned, reliable information.
It is interesting to note that the use of probabilistic routing tables whose entries are learned in an adaptive way by changing on positive feedback and ignoring negative feedback, is reminiscent of older automata approaches to routing in telecommunications networks. In these approaches, a learning automaton is usually placed on each network node. An automaton is defined by a set of possible actions and a vector of associated probabilities, a continuous set of inputs and a learning algorithm to learn input-output associations. Automata are connected in a feedback configuration with the environment (the whole network), and a set of penalty signals from the environment to the actions is defined. Routing choices and modifications to the learning strategy are carried out in a probabilistic way and according to the network conditions (see for example (Nedzelnitsky & Narendra, 1987; Narendra & Thathachar, 1980)). The main difference lies in the fact that in AntNet the ants are part of the environment itself, and they actively direct the learning process towards the most interesting regions of the search space. That is, the whole environment plays a key, active role in learning good state-action pairs.

As a last note for supporting and justifying the utilization of a probabilistic decision policy at the level of data packets, we mention the recent work (Rusmevichientong & Roy, 2000). In a different and more general context the authors show how in a distributed decision system, as a network can be considered, global performance can get as much as close to the equivalent, but ideal case of system with centralized control, if (i) the decision policy used by each independent decision node is stochastic, (ii) only local information concerning the internal status of the other nodes is used to locally select the actions. The paper also shows that the amount of information about the other nodes, which is locally necessary to reach centralized-level performance is independent of the total number of nodes if the policy is stochastic, while it grows linearly with this number if the policy is deterministic. Of course these results must be taken *cum grano salis* considered the differences between the assumed contexts. Nevertheless, they provide an encouraging theoretical substrate to our probabilistic choice.

4.5 Robustness to routing table update frequency

In BF and SPF the broadcast frequency of routing information plays a critical role, particularly so for BF, which has only a local representation of the network status. This frequency is unfortunately problem dependent, and there is no easy way to make it adaptive, while, at the same time, avoiding large oscillations. In Q-R and PQ-R, routing tables updating is data driven: only those Q-values belonging to pairs (i, j) of neighbor nodes visited by packets are updated. Although this is a reasonable strategy given that the exploration of new routes could cause undesired delays to data packets, it causes delays in discovering new good routes, and is a great handicap in a domain where good routes could change all the time. In OSPF, in which routing tables are not updated, we set static link costs on the basis of their physical characteristics. This lack of an adaptive metric is the main reason of the poor performance of OSPF (as remarked in Section 3.1, we slightly penalized OSPF with respect to its real implementations, where additional heuristic knowledge about traffic patterns is used by network administrators to set link costs). In AntNet, we experimentally observed the robustness to changes in the ants' generation rate: for a wide range of generation rates, rather independent of the network size, the algorithm performance is very good and the routing overhead is negligible (see Section 3.3.5).

4.6 Relationships with reinforcement learning

The characteristics of the routing problem allow one to interpret it as a distributed, stochastic time-varying RL problem. This fact, as well as the structure of AntNet, make it natural to draw some parallels between AntNet and classical RL approaches. It is worth remarking that those RL problems that have been most studied, and for which algorithms have been developed, are problems where, unlike routing, assumptions like Markovianity or stationarity of the process considered are satisfied. The characteristics of the adaptive routing problem make it very difficult and not well suited to be solved with usual RL algorithms. This fact, as we explain below, determines a departure of AntNet from classical RL algorithms.

A first way to relate the structure of AntNet to that of a (general) RL algorithm is connected to the way the outcomes of the experiments, the trip times $T_{k \to d}$, are processed. The transformation from the raw values $T_{k \to d}$ to the more refined reinforcements r are reminiscent of what happens in Actor-Critic systems (Barto, Sutton, & Anderson, 1983): the raw reinforcement signal is processed by a critic module, which is learning a model (the node's component $\mathcal{M}$) of the underlying process, and then is fed to the learning system (the routing table $\mathcal{T}$) transformed into an evaluation of the policy followed by the ants. In our case, the critic is both adaptive, to take into account the variability of the traffic process, and rather simple, to meet computational requirements.

Another way of seeing AntNet as a classical RL system is related to its interpretation as a parallel replicated Monte Carlo (MC) system. As was shown by Singh and Sutton (1996), a first-visit MC (only the first visit to a state is used to estimate its value during a trial) simulation system is equivalent to a batch temporal difference (TD) method with replacing traces and decay parameter λ=1. Although AntNet is a first-visit MC simulation system, there are some important differences with the type of MC used by Singh and Sutton (and in other RL works), mainly due to the differences in the considered class of problems. In AntNet, outcomes of experiments are both used to update local models able to capture the variability of the whole network status (only partially observable) and to generate a sequence of stochastic policies. On the contrary, in the MC system considered by Singh and Sutton, outcomes of the experiments are used to compute (reduced) maximum-likelihood estimates of the expected mean and variance of the states' returns (i.e., the total reward following a visit of a state) of a Markov chain. In spite of these differences, the weak parallel with TD(λ) methods is rather interesting, and allows to highlight an important difference between AntNet and its competitors (and general TD methods): in AntNet, following the generation of a stochastic transition chain by the forward ant, there is no back-chaining of the information from one state (i.e., a triple {current node, destination node, next hop node}) to its predecessors. Each state is rewarded only on the basis of the ant's trip time information strictly relevant to it. This approach is completely different from that followed by (TD methods) Q-R, PQ-R, BF and, in a different perspective, by SPF. In fact, these algorithms build the distance estimates at each node by using the predictions made at other nodes. In particular, Q-R and PQ-R, which propagate the estimation information only one step back, are precisely distributed versions of the TD(0) class of algorithms. They could be transformed into generic TD(λ), $0 < \lambda \leq 1$, by transmitting backward to all the previously visited nodes the information collected by the routing packet generated after each data hop. Of course, this would greatly increase the routing traffic generated, because it has to be done after each hop of each data packet, making the approach at least very costly, if feasible at all.

In general, using temporal differences methods in the context of routing presents an important problem: the key condition of the method, the self-consistency between the estimates of successive states[2] may not be strictly satisfied in the general case. This is due to the fact that (i) the dynamics at each node are related in a highly non-linear way to the dynamics of all its neighbors, (ii) the traffic process evolves concurrently over all the nodes, and (iii) there is a recursive interaction between the traffic patterns and the control actions (that is, the modifications of the routing tables). This aspect can explain in part the poor performance of the pure TD(0) algorithms Q-R and PQ-R.

[2]For instance, the prediction made at node k about the time to-go to the destination node d should be additively related to the prediction for the same destination from each one of k's neighbors, being each neighbor one of the ways to go to d.

Chapter 5

Related work

As already pointed out in the Introduction and further discussed in Section 2.3.1, all the algorithms presented in this thesis, (with special emphasis for AntNet and AntNet-FA), find they roots in the framework of ant colony optimization, briefly summarized in Section 1.1. In this respect, there are strong relationships between the algorithms described in this thesis and all the other algorithms developed in the general ACO framework. Actually, in (Dorigo et al., 1999; Dorigo & Di Caro, 1999b, 1999a) ACO is introduced as an *a posteriori* general framework that accounts for all the features common to a wide set of algorithms, mainly developed over the last ten years, that have been inspired by the foraging behavior of colonies of real ants. Therefore, it is quite natural to consider any specific implementation of ACO as related to the algorithms presented in this thesis. In spite of these general affinities, there are several differences between the characteristics of the problems of combinatorial optimization, focus of most of the ACO implementations, and the characteristics of typical network, and adaptive routing in particular, problems. Differences that are accordingly reflected in different salient characteristics of the different algorithms. In Section 2.3.1 these questions have already been addressed in relationship to the issue of adaptive components. In general, network problems have characteristics of spatial distribution, stochasticity, non-stationarity, real-time, that do not usually found their counterpart in combinatorial optimization problems. Therefore, here we only consider as related work the applications of ACO that are specific to online network problems. The reader interested in the more general relationships of the ACO framework with other computational paradigms can consult for example the "Related work" section in (Dorigo et al., 1999).

In addition to the work in the context of ACO, also the work in the field of multi-agent systems and in particular of the application of multi-agent systems for network management and control *de facto* should be considered as related work. Anyway, the "problem" is that it would be necessary a whole thesis to adequately account for it. It is a field in a continual expansion, receiving a growing interest from many different scientific communities. Here we choose to not report about this aspect, being the object of some ongoing and future work. For sake of completeness we just provide here some bibliographic references where overviews on multi-agent systems and on application of the agent technology in telecommunication systems are provided: (Weiss, 1999; Stone & Veloso, 2000; Hayzelden & Bigham, 1999; Kotz & Gray, 1999; Gray et al., 2001; Tintin & Lee, 1999; Küpper & Park, 1998; Vigna, 1998)

Schoonderwoerd, Holland, Bruten and Rothkrantz (1996, 1997), were the firsts to consider routing as a possible application domain for algorithms inspired by the behavior of ant colonies. Their approach, which is applied to routing in telephone networks, differs from AntNet in many respects that we discuss below with some detail to acknowledge the more than "historical" relevance of ABC to the development of AntNet algorithms. The main differences are a direct consequence of the different network model they considered, which has the following characteristics (see Figure 5.1): (i) connection links potentially carry an infinite number of full-duplex, fixed bandwidth channels, and (ii) transmission nodes are crossbar switches with limited connectivity (that is, there is no necessity for queue management in the nodes). In such a model, bottlenecks are put on the nodes, and the congestion degree of a network can be expressed in terms of connections still available at each switch. As a result, the network is cost-symmetric: the congestion status over available paths is completely bi-directional. The path $n_0, n_1, n_2, \ldots, n_k$ connecting n_0 and n_k will exhibit the same level of congestion in both directions because the congestion depends only on the state of the nodes in the path. Moreover, dealing with telephone networks, each call occupies exactly one physical channel across the path. Therefore, "calls" are not multiplexed over the links, but they can be accepted or refused, depending on the possibility of reserving a physical circuit connecting the caller and the receiver. All of these modeling assumptions make the problem of Schoonderwoerd et al. rather different from the cost-asymmetric routing problems for data networks we presented in this thesis. This difference is reflected in many algorithmic differences between ABC and AntNet, the most important of which is that in ABC ants update pheromone trails after each step, without waiting for the completion of an experiment as done in AntNet.[1] In ABC ants move over a control network isomorphic to the one were calls are really established. In the used model the system evolves synchronously according to a discrete clock. At each node an ant ages of t virtual steps computed as a fixed function of the current node spare capacity, the number of still available channels. Let s be the source and d the destination node of a traveling ant. After crossing the control link connecting node i with node j, the routing table on node j is updated by using the ant age T: the probability for subsequent ants to choose node i when their destination node is s, is increased proportionally to the current value of T according to an updating/normalizing rule similar to rules 2.6 and 2.5. In this way is the routing table in opposite direction of the ant motion which is modified. This strategy is reasonably sound only in the case of an (at least) approximately cost-symmetric network, where where the congestion status is direction-independent. In particular, the case considered by the authors fits this model, being the network completely cost-symmetric by design.

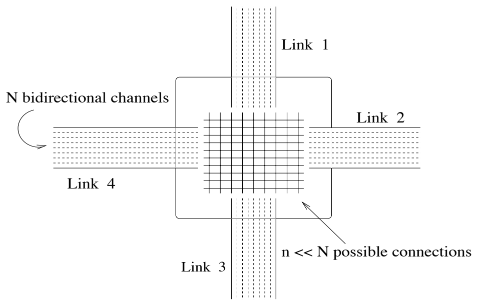


Figure 5.1: Network node in the telecommunications network model of Schoonderwoerd et al. (1996).

Other important differences can be found in the fact that (i) ABC does not use local models to score the ants trip times: T values are used as they are, without considering their relativity with respect to different network states (see Section 2.1.2), (ii) nor local heuristic information (equation 2.3) or ant-private memory are used to improve the ants decision policies. Moreover, ABC's ants do not recover from cycles and do not use the information contained in all the ant sub-paths (Section 2.1.1).

ABC has been tested on a model of the British Telecom (BT) telephone network (30 nodes) using a sequential discrete time simulator and has been compared, in terms of percentage of accepted calls, to an agent-based algorithm developed by BT researchers. Results were encouraging:[2] ABC always performed significantly better than its competitor on a variety of different traffic situations.

Subramanian, Druschel and Chen (1997) have proposed an ant-based algorithm for packet-switched networks. Their algorithm is a straightforward extension of Schoonderwo-

[1]This choice, justified by the cost-symmetry assumption, is reminiscent of the pheromone trail updating strategy implemented in ant-density, another of the first ant colony based algorithms (Dorigo et al., 1991; Dorigo, 1992; Colorni, Dorigo, & Maniezzo, 1991), and makes ABC behavior closer to those of real ants.

[2]Here and in the following we use the word "encouraging" whenever interesting results were obtained but the algorithm was compared only on simple problems, or with no state-of-the-art algorithms or under limited experimental conditions.

erd et al. system by adding so-called *uniform ants*, a very simple (and of quite questionable efficacy) exploration mechanism that should avoid a rapid sub-optimal convergence of the algorithm. A limitation of Subramanian et al. work is that, although the algorithm they propose is based on the same cost-symmetry hypothesis as ABC, they apply it to packet-switched networks where this requirement is very often not met.

Bonabeau, Henaux, Guérin, Snyers, Kuntz and Théraulaz (1998) improved the ABC algorithm by the introduction of a dynamic programming mechanism. They update the pheromone trail values of all the links on an ant path not only with respect to the ant's origin node, but also with respect to all the intermediate nodes on the sub-path between the origin and the ant current node. A similar mechanism, described in Section 2.1.1, was earlier implemented in AntNet since its first reported version (Di Caro & Dorigo, 1997a). The difference between the AntNet's and Bonabeau's et al. updating strategy lies in the fact that AntNet refuses the general validity of the Bellman's principle of optimality, at the base of dynamic programming, for the considered network situations. Therefore, AntNet does not necessarily updates all the sub-paths of a path, but only those that appear to carry reliable or competitive information. The fact that a straight application of the Bellman's principle should be criticized find its rationale in the fact that the traffic dynamics is intrinsically non-stationary and, at the same time, only local information can be used to take routing decisions. If the traffic dynamics is quasi-stationary in the portion of the network interested by the current routing decisions then it could be reasonable to use the properties of consistency that advocate dynamic programming techniques.

Van der Put and Rothkrantz (1998, 1999) designed ABC-backward, an extension of the ABC algorithm based to the AntNet behavior. In this way ABC-backward is applicable to cost-asymmetric networks, but ultimately the algorithm results to be identical to AntNet with some, rather unjustified, simplifications inherited from the ABC settings. The authors use the same forward-backward ant mechanism as in AntNet: Forward ants, while moving from the source to the destination node, collect information on the status of the network, and backward ants use this information to update the routing table of the visited links during their journey back from the destination to the source node. In ABC-backward, backward ants update the routing tables using an updating formula identical to that used in ABC, except for the fact that the ants' age is replaced by the trip times experienced by the ants in their forward journey. Van der Put and Rothkrantz have shown experimentally that ABC-backward has a better performance than ABC on both cost-symmetric, because backward ants can avoid depositing information on cycles, and cost-asymmetric networks. They apply ABC-backward to a fax distribution problem proposed by the Dutch largest telephone company (KPN Telecom).

White, Pagurek and Oppacher (1998) use an ACO algorithm for routing in connection-oriented point-to-point and point-to-multipoint networks. The algorithm, called ASGA, follows a scheme very similar to that of Ant System, the first published ACO algorithm (Dorigo et al., 1991; Dorigo, 1992). Ant System was designed to solve instances of the traveling salesman problem (TSP), where an Hamiltonian path of minimum cost must be found. If the Hamiltonian constraint is relaxed, then the problem becomes basically equivalent to a minimum cost path problem, which is equivalent to a "one-dimensional" routing problem (in routing, minimum cost paths must be found, possibly at the same time, for all the pairs of nodes in the network). In this perspective, a not-so-*ad-hoc* solution strategy for TSPs, such that used in Ant System, can be quite straightforwardly applied to routing problems. This is what the authors did, by adding additional components to account for the fact that it was the case of a routing not a TSP problem. In particular, they used the same forward-backward mechanism of AntNet, combined with a mechanism very similar to that used by AntNet++ for supporting the setup phase of a QoS or connection-oriented application (see Section 2.3.3). The basic difference with AntNet lies in the fact that the authors apply the same selection and updating formulae used in Ant System. Basically, the formula 2.3 is replaced by one where the terms are multiplied instead of being summed and the term l is replaced by a an unspecified link cost, while the formulae 2.6 and 2.5 are replaced by

exponential averages. From the source node of each incoming connection, a group of ants is launched to search for a path. At the beginning of the trip each ant k sets a private path cost variable C_k to 0, and after each link crossing the path cost is incremented by the link cost l_{ij}: $C_k \leftarrow C_k + l_{ij}$. When arrived at destination, ants move backward to their source node and at each node they use a simple additive rule, similar to that of AntNet, to compute the equivalent of the value T in AntNet: T is equal to C_k, which is the sum of all the previously encountered costs. When all the spooled ants come back at the source node, a simple local daemon algorithm decides whether to allocate a path, based on the percentage of ants that followed a same path. Moreover, during all the connection lifetime, the local daemon launches and coordinates exploring ants to re-route the connection paths in case of network congestion or failures. A genetic algorithm (Goldberg, 1989; Holland, 1975) is used online to evolve two critical parameters (the equivalent of α in eq. 2.3) that regulate the behavior of the transition rule formula (because of this mechanism the algorithm is called ASGA, ant-system plus genetic algorithm). Some preliminary results were obtained testing the algorithm on several networks and using several link cost functions. Results are promising: the algorithm is able to compute shortest paths and the genetic adaptation of the rule parameters considerably improves the algorithm's performance.
Actually the same authors published several other short conference papers on ant, swarm, multi-agent systems for network management and control. Most of them mix the above mechanisms and concepts with other similar flavors, therefore we do not further mention them here.

Heusse, Snyers, Guérin and Kuntz (1998) developed a new algorithm for general cost-asymmetric networks, called Co-operative Asymmetric Forward (CAF). CAF, even if still grounded on the ACO framework, introduces some new ideas to exploit the advantages of an online step-by-step updating scheme, as that used by ABC, versus the forward-backward model of AntNet. In ABC such a strategy was made possible by the assumptions of cost-symmetry, in the CAF this assumption is relaxed, but it is still possible to use it. In fact, each data packet, after going from node i to node j, releases on node j the information c_{ij} about the sum of the experienced waiting and crossing times from node i. This information is used as an estimate of the time distance to go from i to j, and is read by the ants traveling in the opposite direction to perform online step-by-step updating of the local routing table. Of course, to work properly the system requires that an "updating" ant would arrive in coincidence or with a negligible time shit with respect to the moment the "information" ant deposited the information c. The algorithm's authors tested CAF under some static and dynamic conditions, using the average number of packets waiting in the queues and the average packet delay as performance measures. They compared CAF to an algorithm very similar to an earlier version of AntNet. Results were encouraging, under all the test situations CAF performed better than its competitors.

The two-ways mechanism of Heusse et al. is quite general and could be incorporated in AntNet++ straightforwardly, as we plan to do. Actually Doi and Yamamura (2000, 2001) used also a similar strategy in their BntNetL algorithm. BntNetL has been designed as a supposed improvement over AntNet. Supposed because the authors misunderstood some of the behaviors of AntNet and then tried to devise some additional heuristics to "correct" them. For example, they asserted that AntNet's selection rule could get "locked" if one entry in the routing table would reach the limit value of 1, but actually this is not the case, because AntNet makes use of the selection rule 2.3 that weights both the entries in the routing table and the current status of the link queues. Anyway, the minor modifications brought to AntNet-FA (the Heusse et al. mechanism being the most significant one) have been tested and BntNetL has been compared to AntNet-FA on a restricted set of simulated situations. Results essentially appear to be very similar, so of not so much interest (also considered that their version of AntNet-FA contains, again, some incorrectly interpreted parts).

On a similar stream of misleading interpretations of AntNet is the work of Oida and Kataoka (1999) . These authors, in spite of journal published versions of AntNet, decided to work on improving the very first version of AntNet (Di Caro & Dorigo, 1997a), where the status of the data link queues was not used and then the above mentioned “lock” mechanism could actually happen. To avoid this fact, Oida and Kataoka added some simple adaptive mechanisms to the routing table updating rule. Their algorithms, DCY-AntNet and NFB-Ants, once compared to the early version of AntNet (Di Caro & Dorigo, 1997a) performed much better under the limit situations the algorithms have been thought to address. The used mechanism could actually be straightforwardly added to AntNet-FA and AntNet++. Their efficacy should anyway be tested.

The same Oida but this time in collaboration with Sekido (1999) also added some simple functionalities to the basic behavior of AntNet to make it suitable to work in a QoS environment with constraints on the bandwidth and the number of hops. In particular they make the forward ants moving in a “virtually constrained network”. In the sense that the probability of a forward ant of choosing a link is made depending not only on the values in the routing table but also on the level of already reserved bandwidth. If almost all bandwidth had already been reserved then the probability is accordingly made very low. Similarly, if the number of executed hops is already too big and/or none of the outgoing links has much free bandwidth, then the forward ant terminate itself. Actually, these heuristics introduced by the authors can be readily seen as the direct transposition of some basic AntNet's mechanisms in a QoS context. The heuristic taking into account also the available bandwidth to assign the link probability is equivalent to the AntNet's heuristic that takes into account the current number of bits waiting on the link queue. While the mechanisms for the ant self-termination had already been implemented in AntNet to eliminate those ants that are hopelessly trying to build a very bad path.

Other minor implementations of both the ACO and AntNet paradigms are the works of: Gallego-Schmid (1999) , on a minor modification of AntNet in the general perspective of network management; Sigel, Denby and Le Heǵarat-Mascle (2000) , who applied a straight modification of Ant-System to the problem of adaptive routing in satellite networks; Sandalidis, Mavromoustakis and Stavroulakis (2001) , who add a probabilistic component at the routing decision policy in ABC; Michalareas and Sacks (2001) who implemented a trivial hybrid between AntNet and ABC for the management of the routing in multi-constrained or QoS networks. We label these implementations as “minor” because they have just been submitted for publication but do not have been accepted yet (or have been accepted in an restricted context as is the case of Gallego-Schmid).

Using the same concepts (stigmergy, ants, pheromone, multi-agency) underlying ACO but not being explicitly inspired by ACO, Fennet and Hassas (2000) developed a preliminary model of a multi-agent system for multiple criteria balancing on a network of processors. It has been tested on a some rather simple simulated situations. In their system ants move among the machines, perceiving local artificial pheromone fields, that are emitted by at the local nodes and that encode some useful information about the criteria to optimize. While the description the authors give of the actions and tasks is quite vague and not not well defined, what is a bit interesting to mention here is the fact that they use a terminology that reminds that of AntNet++: a distinction is made between static and mobile agents interacting together, while the mobile agents are also seen as sensory and effector agents for user applications.

Chapter 6

Conclusions and Future work

In this thesis, we have introduced AntNet, a novel distributed approach to routing in packet-switched communications networks. We compared AntNet with 6 state-of-the-art routing algorithms on a variety of realistic testbeds. AntNet showed superior performance and robustness to internal parameter settings for almost all the experiments. AntNet's most innovative aspect is the use of stigmergetic communication to coordinate the actions of a set of agents that cooperate to build adaptive routing tables. Although this is not the first application of stigmergy-related concepts to optimization problems (e.g., Dorigo et al., 1991; Dorigo, 1992; Dorigo et al., 1996; Bonabeau et al., 1999), the application presented here is unique in many respects. First, in AntNet, stigmergy-based control is coupled to a model-building activity: information collected by ants is used not only to modify routing tables, but also to build local models of the network status to be used to better direct the routing table modifications. Second, this is the first attempt to evaluate stigmergy-based control on a realistic simulator of communications networks: the used simulator retains many of the basic components of a real routing system. An interesting step forward, in the direction of testing the applicability of the idea presented to real networks, would be to rerun the experiments presented here using a complete Internet simulator. Third, this is also the first attempt to evaluate stigmergy-based control by comparing a stigmergetic algorithm to state-of-the-art algorithms on a realistic set of benchmark problems. It is very promising that AntNet turned out to be the best performing in all the tested conditions.

There are obviously a number of directions in which the current work could be extended, which are listed below.

1) A first, natural, extension of the current work would consider the inclusion in the simulator of flow and congestion control components (with re-transmissions and error management). This inclusion will require a paired tuning of the routing and flow-congestion components, to select the best matching between their dynamics.

2) In AntNet, each forward ant makes a random experiment: it builds a path from a source node s to a destination node d. The path is built exploiting the information contained in the probabilistic routing tables and the status of the queues of the visited nodes. While building the path, the ant collects information on the status of the network. This is done by sharing link queues with data packets, and by measuring waiting times of queues and traversal times that will be used as raw reinforcements by backward ants. Since forward ants share queues with data packets, the time required to run an experiment depends on the network load, and is approximately the same as the time $T_{s \rightarrow d}$ required for a packet to go from the same source node s to the same destination node d. This delays the moment the information collected by forward ants can be distributed by backward ants, and makes it less up-to-date than it could be. A possible improvement in this schema would be to add a model of link-queue depletion to nodes, and to let forward ants use high priority queues to reach their destinations without storing crossing times (for a first step in this direction see Di Caro & Dorigo, 1998f). Backward ants would then make the same path, in the

opposite direction, as forward ants, but use the queue local models they find on their way to estimate local “virtual” queueing and crossing times. Raw reinforcements, used to update the routing tables, are then computed using these estimates. Clearly, here there is a trade-off between delayed but real information and more recent but estimated information. It will be interesting to see which scheme works better, although we are confident that the local queue models should allow the backward ants to build estimates accurate enough to make the improved system more effective than the current AntNet, at a cost of a little increase in computational complexity at the nodes.

3) As we discussed in Section 4, AntNet is missing one of the main components of classical RL/TD algorithms: there is no back-chaining of information from a state to previous ones, each node policy is learned by using a complete local perspective. An obvious extension of our work would therefore be to study versions of AntNet closer to TD(λ) algorithms. In this case each node should maintain Q-values expressing the estimate of the distance to each destination via each neighbor. These estimates should be updated by using both the ant trip time outcome and the estimates coming from successive nodes (closer to the destination node) that could be also carried by the backward ant.

4) In this thesis we applied AntNet to routing in datagram communications networks. It is reasonable to think that AntNet could be easily adapted to be used for the generation of real-time car route guidance in Dynamic Traffic Assignment (DTA) systems (see for example Yang, 1997). DTA systems exploit currently available and emerging computer, communication, and vehicle sensing technologies to monitor, manage and control the transportation system (the attention is now focused mainly on highway systems) and to provide various levels of information and advice to system users so that they can make timely and informed travel decisions. Therefore, adaptive routing of vehicle traffic presents very similar features to the routing of data packets in communications networks. Moreover, vehicle traffic control systems have the interesting property of a very simplified “transport” layer. In fact, many activities that interfere with routing and that are implemented in the transport layer of communications networks do not exist, or exist only to a limited extent, in vehicles traffic control algorithms. For example, typical transport layer activities like data acknowledgement and retransmission cannot be implemented with real vehicles. Other activities, like flow control, have strong constraints (e.g., people would not be happy to be forbidden to leave their offices for, say, one hour on the grounds that there are already too many cars on the streets!). This makes AntNet still more interesting since it can express its full potential as a routing algorithm.

5) In AntNet, whenever an ant uses a link its desirability (probability) is incremented. Although this strategy, which finds its roots in the ant colony biological metaphor that inspired our work, allowed us to obtain excellent results, it would be interesting to investigate the use of negative reinforcements, even if it can potentially lead to stability problems, as observed by people working on older automata systems. As discussed before, AntNet differs from automata systems because of the active role played by the ants. Therefore, the use of negative reinforcements could show itself to be effective, for example, in reducing the probability of choosing a given link if the ant that used it performed very badly.

Appendix A

Optimal and shortest path routing

In this Appendix, the characteristics of the two most used routing paradigms, optimal and shortest path routing (introduced in Section 1.2.1) are summarized:

A.1 Optimal routing

Optimal routing (Bertsekas & Gallager, 1992) has a network-wide perspective and its objective is to optimize a function of all individual link flows.
Optimal routing models are also 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, and 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.

A.2 Shortest path routing

Shortest path routing (Wang & Crowcroft, 1992) has a source-destination pair perspective. As opposed to optimal routing, there is no global cost function to be optimized. Instead, the route between each node pair is considered by itself and no *a priori* knowledge about the traffic process is required (although of course such knowledge could be fruitfully used).
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; Wang & Crowcroft, 1992). Some very popular cost metrics take into account queuing and transmission delays, link usage, link capacity and various combination of these measures. The way costs are updated usually involves attempting to reduce big variations considering both long-term and short-term

statistics of link congestion states (Khanna & Zinky, 1989; Shankar, Alaettinoğlu, Dussa-Zieger, & Matta, 1992b).
On the other hand, if the costs are *static*, they will reflect both some measure of the expected/wished traffic load over the links and their transmission capacity. Of course, serious loss of efficiency could arise in case of non-stationary conditions or when the a priori assumptions about the traffic patterns are strongly violated in practice.
Considering the different content stored in each routing table, shortest path algorithms can be further subdivided in two classes called *distance-vector* and *link-state* (Steenstrup, 1995; Shankar et al., 1992b). 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.
4. 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 which uses all the information stored in the table to choose some randomized or alternative path.

In the following, the main features specific to each class are described.

A.2.1 Distance-vector

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.[1] 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 the number of nodes in the network and n is the average connectivity degree (i.e., the average number of neighbor 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 overwriting the entry corresponding to node j with the received values.

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

$$arg \min_{j \in \mathcal{N}_i} \{d_{ij} + D_j\}$$

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 wider class of distance-vector algorithms (Malkin

[1] In some cases, only the best estimates are kept at nodes. Therefore, the above triples are defined for all the destinations only.

& Steenstrup, 1995) developed with the aim of reducing the risk of circular loops and of accelerating the convergence in case of rapid changes in link costs.

A.2.2 Link-state

Link-state algorithms make use of routing tables containing much more information than that 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 whole network, describing the details of all its components and their current interconnections. Using this database as input, each node calculates its best paths using an appropriate algorithm like Dijkstra's (1959) algorithm (a wide variety of alternative efficient algorithms are available, as described for example in Cherkassky, Goldberg, & Radzik, 1994). The memory requirements for each node in this case are $O(N^2)$.

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 mechanism (Bertsekas & Gallager, 1992) 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 have been implemented to make the algorithm behavior more robust and efficient (Moy, 1998).

Appendix B

An historical glance at the routing on the Internet

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

The first routing algorithm on ARPANET, implemented in 1969, was a distributed adaptive asynchronous Bellman-Ford algorithm (Bertsekas & Gallager, 1992). 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 et al., 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 behavior than distance-vector algorithms, but in this case too, under 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, 1998) 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 highlights the many 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.

Bibliography

Alaettinoğlu, C., Shankar, A. U., Dussa-Zieger, K., & Matta, I. (1992). Design and implementation of MaRS: A routing testbed. Tech. rep. UMIACS-TR-92-103, CS-TR-2964, Institute for Advanced Computer Studies and Department of Computer Science, University of Maryland, College Park (MD).

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.

Birattari, M., Di Caro, G., & Dorigo, M. (2000). For a formal foundation of the Ant Programming approach to combinatorial optimization. Part 1: The problem, the representation, and the general solution strategy. Tech. rep. TR-H-301, ATR Human Information Processing Research Laboratories, Kyoto, Japan.

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.

Bonabeau, E., Dorigo, M., & Théraulaz, G. (1999). *From Natural to Artificial Swarm Intelligence.* Oxford University Press.

Bonabeau, E., Dorigo, M., & Theraulaz, G. (2000). Inspiration for optimization from social insect behaviour. *Nature*, pp. 39–42.

Bonabeau, E., Henaux, F., Guérin, S., Snyers, D., Kuntz, P., & Théraulaz, G. (1998). Routing in telecommunication networks with "Smart" ant-like agents. In *Proceedings of IATA'98, Second Int. Workshop on Intelligent Agents for Telecommunication Applications*, Vol. 1437 of *Lecture Notes in Aritficial Intelligence.* Springer Verlag.

Boyan, J., & Littman, M. (1994). Packet routing in dinamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems 6 (NIPS6)*, pp. 671–678. San Francisco, CA:Morgan Kaufmann.

Brakmo, L. S., O'Malley, S. W., & Peterson, L. L. (1994). TCP vegas: New techniques for congestion detection and avoidance. *ACM Computer Communication Review (SIGCOMM'94)*, *24*(4).

Cheng, C., Riley, R., Kumar, S. P. R., & Garcia-Luna-Aceves, J. J. (1989). A loop-free extended bellman-ford routing protocol without bouncing effect. *ACM Computer Communication Review (SIGCOMM '89)*, *18*(4), 224–236.

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.

Choi, S., & Yeung, D.-Y. (1996). Predictive Q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In *Advances in Neural Information Processing Systems 8 (NIPS8)*, pp. 945–951. MIT Press.

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 183–188.

Colorni, A., Dorigo, M., & Maniezzo, V. (1991). Distributed optimization by ant colonies. In *Proceedings of the European Conference on Artificial Life (ECAL 91)*, pp. 134–142. Elsevier.

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.

Di Caro, G., & Dorigo, M. (1997a). Distributed reinforcement agents for adaptive routing in communication networks. *Third European Workshop on Reinforcement Learning (EWRL-3)*, Rennes, France.

Di Caro, G., & Dorigo, M. (1997b). AntNet: A mobile agents approach to adaptive routing in communication networks. *Ninth Dutch Conference on Artificial Intelligence (NAIC '97)*, Antwerpen, Belgium.

Di Caro, G., & Dorigo, M. (1997c). Adaptive learning of routing tables in communication networks. In *Proceedings of the Italian Workshop on Machine Learning* Torino, Italy.

Di Caro, G., & Dorigo, M. (1998a). Mobile agents for adaptive routing. In *Proceedings of the 31st International Conference on System Sciences (HICSS-31)*, Vol. 7, pp. 74–83. IEEE Computer Society Press.

Di Caro, G., & Dorigo, M. (1998b). AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research (JAIR)*, *9*, 317–365.

Di Caro, G., & Dorigo, M. (1998c). Ant colonies for adaptive routing in packet-switched communications networks. In Eiben, A. E., Back, T., Schoenauer, M., & Schwefel, H.-P. (Eds.), *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, pp. 673–682. Springer-Verlag.

Di Caro, G., & Dorigo, M. (1998d). An adaptive multi-agent routing algorithm inspired by ants behavior. In *Proceedings of PART98 - 5th Annual Australasian Conference on Parallel and Real-Time Systems*, pp. 261–272. Springer-Verlag.

Di Caro, G., & Dorigo, M. (1998e). Extending AntNet for best-effort Quality-of-Service routing. *ANTS'98 - From Ant Colonies to Artificial Ants: First International Workshop on Ant Colony Optimization*, Brussels (Belgium).

Di Caro, G., & Dorigo, M. (1998f). Two ant colony algorithms for best-effort routing in datagram networks. In *Proceedings of the Tenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'98)*, pp. 541–546. IASTED/ACTA Press.

Di Caro, G., & Dorigo, M. (1997a). AntNet: A mobile agents approach to adaptive routing. Tech. rep. 97-12, IRIDIA, Université Libre de Bruxelles, Brussels (Belgium).

Di Caro, G., & Dorigo, M. (1997b). A study of distributed stigmergetic control for packet-switched communications networks. Tech. rep. 97-18, IRIDIA, Université Libre de Bruxelles, Brussels (Belgium).

Di Caro, G., & Vasilakos, T. (2000). Ant-SELA: Ant-agents and stochastic automata learn adaptive routing tables for QoS routing in ATM networks. *ANTS'2000 - From Ant Colonies to Artificial Ants: Second International Workshop on Ant Colony Optimization*, Brussels (Belgium).

Dijkstra, E. W. (1959). A note on two problems in connection with graphs. *Numer. Math.*, *1*, 269–271.

Doi, S., & Yamamura, M. (2000). BntNetL: Evaluation of its performace under congestion. *Journal of IEICE B* (in Japanese), 1702–1711.

Doi, S., & Yamamura, M. (2001). Robust algorithm for network routing using bidirectional traffic statistics collection. Submitted to *IEEE Transactions on Evolutionary Computation*.

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

Dorigo, M., & Di Caro, G. (1999a). Ant colony optimization: A new meta-heuristic. In *Proceedings of CEC99 - Congress on Evolutionary Computation* Washington DC.

Dorigo, M., & Di Caro, G. (1999b). The ant colony optimization meta-heuristic. In Corne, D., Dorigo, M., & Glover, F. (Eds.), *New Ideas in Optimization*. McGraw-Hill.

Dorigo, M., Di Caro, G., & (Editors), T. S. (2000). Ant algorithms. Special Issue on *Future Generation Computer Systems (FGCS)*, *16*(8).

Dorigo, M., Di Caro, G., & Gambardella, L. M. (1999). Ant algorithms for distributed discrete optimization. *Artificial Life*, *5*(2).

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). Positive feedback as a search strategy. Tech. rep. 91-016, Dipartimento di Elettronica, Politecnico di Milano, IT.

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

Fenet, S., & Hassas, S. (2000). A.N.T.: a distributed network control framework based on mobile agents. In *Proceedings of the International ICSC Congress on Intelligent Systems And Applications*.

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

Gallego-Schmid, M. (1999). Modified AntNet: Software application in the evaluation and management of a telecommunication network. Genetic and Evolutionary Computation Conference (GECCO-99), Student Workshop.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley.

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.

Gray, R. S., Cybenko, G., Kotz, D., & Rus, D. (2001). Mobile agents: Motivations and state of the art. In Bradshaw, J. (Ed.), *Handbook of Agent Technology.* AAAI/MIT Press.

Hauskrecht, M. (1997). *Planning and control in stochastic domains with imperfect information.* Ph.D. thesis, Department of Electrical Engineering and Computer Science, MIT.

Hayzelden, A. L. G., & Bigham, J. (1999). Agent Technology in Communications Systems: An Overview. *Knowledge Engineering Review.*

Heusse, M., Guérin, S., Snyers, D., & Kuntz, P. (1998). Adaptive agent-driven routing and load balancing in communication networks. Tech. rep. RR-98001-IASC, Départment Intelligence Artificielle et Sciences Cognitives, ENST Bretagne. Accepted for publication in the *Journal of Complex Systems.*

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

Jaakkola, T., Singh, S. P., & Jordan, M. I. (1995). Reinforcement learning algorithms for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems 7*, pp. 345–352. MIT Press.

Kaebling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stohastic domains. *Artificial Intelligence*, *101*(1-2), 99–134.

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. *ACM SIGCOMM Computer Communication Review*, *19*(4), 45–56.

Kotz, D., & Gray, R. S. (1999). Mobile agents and the future of the Internet. *ACM Operating Systems Review*, *33*(3), 7–13.

Küpper, A., & Park, A. S. (1998). Stationary vs. mobile user agents in future mobile telecommunication networks. In *Proceedings if Mobile Agents '98*, Vol. 1477 of *Lecture Notes in Computer Science*, pp. 112–124. Springer-Verlag: Heidelberg, Germany.

Ma, Q. (1998). *Quality of Service routing in integrated services networks.* Ph.D. thesis, Department of Computer Science, Carnegie Mellon University.

Ma, Q., Steenkiste, P., & Zhang, H. (1996). Routing in high-bandwidth traffic in max-min fair share networks. *ACM Computer Communication Review (SIGCOMM'96)*, *26*(4), 206–217.

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.

McCallum, A. K. (1995). *Reinforcement learning with selective perception and hidden state.* Ph.D. thesis, Department of Computer Science, University of Rochester, Rochester (NY).

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

Merlin, P., & Segall, A. (1979). A failsafe distributed routing protocol. *IEEE Transactions on Communications*, *COM-27*(9), 1280–1287.

Michalareas, T., & Sacks, L. (2001). Swarm intelligence techniques for solving multi-constrained routing in internet networks. Submitted.

Moulin, B. (1998). The social dimension of interactions in multi-agent systems. In Wobcke, W., Pagnucco, M., & Zhang, C. (Eds.), *Agents and Multi-Agent Systems, Formalisms, Methodologies and Applications*, Vol. 1441 of *Lecture Notes in Artificial Intelligence*, pp. 109–122. Springer-Verlag: Heidelberg, Germany.

Moy, J. T. (1998). *OSPF Anatomy of an Internet Routing Protocol.* Addison-Wesley.

Narendra, K. S., & Thathachar, M. A. (1980). On the behavior of a learning automaton in a changing environment with application to telephone traffic routing. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-10*(5), 262–269.

Nedzelnitsky, O. V., & Narendra, K. S. (1987). Nonstationary models of learning automata routing in data communication networks. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-17*, 1004–1015.

Oida, K., & Kataoka, A. (1999). Lock-free AntNet and its evaluation for adaptiveness. *Journal of IEICE B* (in Japanese), *J82-B*(7), 1309–1319.

Oida, K., & Sekido, M. (1999). An agent-based routing system for QoS guarantees. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pp. 833–838.

Papoulis, A. (1991). *Probability, Random Variables and Stochastic Process* (Third edition). McGraw-Hill.

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

Rubistein, R. Y. (1981). *Simulation and the Monte Carlo Method.* John Wiley & Sons.

Rusmevichientong, P., & Roy, B. V. (2000). Decentralized decision-making in a large team with local information. Tech. rep., Department of Management Science and Engineering, Stanford University. Submitted to Games and Economic Behavior.

Sandalidis, H., Mavromoustakis, K., & Stavroulakis, P. (2001). Ant-based probabilistic routing with pheromone and antipheromone mechanisms. Submitted.

Sandick, H., & Crawley, E. (1997). QoS routing (qosr) working group report. Internet Draft, Internet Engineering Task Force (IEFT).

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

Schoonderwoerd, R., Holland, O., Bruten, J., & Rothkrantz, L. (1996). Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, *5*(2), 169–207.

Shankar, A. U., Alaettinoğlu, C., Dussa-Zieger, K., & Matta, I. (1992a). Performance comparison of routing protocols under dynamic and static file transfer connections. *ACM Computer Communication Review*, *22*(5), 39–52.

Shankar, A. U., Alaettinoğlu, C., Dussa-Zieger, K., & Matta, I. (1992b). Transient and steady-state performance of routing protocols: Distance-vector versus link-state. Tech. rep. UMIACS-TR-92-87, CS-TR-2940, Institute for Advanced Computer Studies and Department of Computer Science, University of Maryland, College Park (MD).

Shoham, Y., & Tennenholtz, M. (1995). On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, *73*(1–2), 231–252.

Sigel, E., Denby, B., & Heárat-Mascle, S. L. (2000). Application of ant colony optimization to adaptive routing in a Leo telecommunications satellite network. Submitted.

Singh, S. P., & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, *22*, 123–158.

Singh, S. P., Jaakkola, T., & Jordan, M. I. (1994). Learning without state estimation in partially observable Markovian decision processes. In *Proceedings of the Eleventh Machine Learning Conference*, pp. 284–292. New Brunswick, NJ: Morgan Kaufmann.

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

Stone, P., & Veloso, M. M. (2000). Multiagent systems: A survey from a machine learning persective. *Autonomous Robots*, *8*(3).

Streltsov, S., & Vakili, P. (1996). Variance reduction algorithms for parallel replicated simulation of uniformized Markov chains. *Discrete Event Dynamic Systems: Theory and Applications*, *6*, 159–180.

Subramanian, D., Druschel, P., & Chen, J. (1997). Ants and reinforcement learning: A case study in routing in dynamic networks. In *Proceedings of IJCAI-97, International Joint Conference on Artificial Intelligence*, pp. 832–838. Morgan Kaufmann.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press.

Tennenhouse, D. L., Smith, J. M., Sincoskie, W. D., Wetherall, D. J., & Minden, G. J. (1997). A survey of active network research. *IEEE Communications Magazine*, *35*(1), 80–86.

The ATM Forum (1996). *Private Network-Network Interface Specification: Version 1.0.*

Tintin, R. A., & Lee, D. I. (1999). Intelligent and mobile agents over legacy, present and future telecommunication networks. In Karmouch, A., & Impley, R. (Eds.), *First International Workshop on Mobile Agents for Telecommunication Applications (MATA'99)*, pp. 109–126. World Scientific Publishing Ltd.

van der Put, R. (1998). Routing in the faxfactory using mobile agents. Tech. rep. R&D-SV-98-276, KPN Research.

van der Put, R., & Rothkrantz, L. (1999). Routing in packet switched networks using agents. Submitted to *Simulation Practice and Theory.*

Vasilakos, A., & Papadimitriou, G. (1995). A new approach to the design of reinforcement scheme for learning automata: Stochastic Estimator Learning Algorithms. *Neurocomputing*, *7*(275).

Vigna, G. (Ed.). (1998). *Mobile Agents and Security*, Vol. 1419 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany.

Walrand, J., & Varaiya, P. (1996). *High-performance Communication Networks*. Morgan Kaufmann.

Wang, Z., & Crowcroft, J. (1992). Analysis of shortest-path routing algorithms in a dynamic network environment. *ACM Computer Communication Review*, *22*(2).

Weiss, G. (Ed.). (1999). *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*. The MIT Press.

White, T., Pagurek, B., & Oppacher, F. (1998a). Asga: improving the ant system by integration with genetic algorithms. In *Proceedings of the Third Genetic Programming Conference*, pp. 610–617.

White, T., Pagurek, B., & Oppacher, F. (1998b). Connection management using adaptive mobile agents. In Arabnia, H. (Ed.), *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, pp. 802–809. CSREA Press.

Yang, Q. (1997). *A Simulation Laboratory for Evaluation of Dynamic Traffic Management Systems*. Ph.D. thesis, Department of Civil and Environmental Engineering, Massachusetts Institute of Technology (MIT).

Zinky, J., Vichniac, G., & Khanna, A. (1989). Performance of the revised routing metric in the ARPANET and MILNET. In *MILCOM 89*.