Design Patterns from Biology for Distributed Computing

OZALP BABAOGLU and MÁRK JELASITY
University of Bologna
GEOFFREY CANRIGHT and TORE URNES
Telenor R&D
ANDREAS DEUTSCH and NILOY GANGULY
Dresden University of Technology
GIANNI DI CARO, FREDERICK DUCATELLE, LUCA MARIA GAMBARDELLA and ROBERTO MONTEMANNI
Istituto "Dalle Molle" di Studi sull'Intelligenza Artificiale (IDSIA)
and
ALBERTO MONTRESOR
University of Trento

Recent developments in information technology have brought about important changes in distributed computing. New environments such as massively large-scale, wide-area computer networks and mobile ad hoc networks have emerged. Common characteristics of these environments include extreme *dynamicity*, *unreliability* and *large scale*. Traditional approaches to designing distributed applications in these environments based on central control, small scale or strong reliability assumptions are not suitable for exploiting their enormous potential. Based on the observation that living organisms can effectively organize large numbers of unreliable and dynamically-changing components (cells, molecules, individuals, etc.) into robust and adaptive structures, it has long been a research challenge to characterize the key ideas and mechanisms that make biological systems work and to apply them to distributed systems engineering. In this paper we propose a conceptual framework that captures several basic biological processes in the form of a family of design patterns. Examples include plain diffusion, replication, chemotaxis and stigmergy. We show through examples how to implement important functions for distributed computing based on these patterns. Using a common evaluation methodology, we show that our bio-inspired solutions have performance comparable to traditional, state-of-the-art solutions while they inherit desirable properties of biological systems including adaptivity and robustness.

Categories and Subject Descriptors: C.2.1 [**Computer communication networks**]: Network Architecture and Design—*Distributed networks, Wireless Communication*; C.2.2 [**Computer communication networks**]: Network Protocols—*Routing protocols*; C.2.3 [**Computer communication networks**]: Network Operations—*Network Monitoring*; C.2.4 [**Computer communication networks**]: Distributed Systems—*Distributed applications*

General Terms: Algorithms, Design, Performance, Reliability

Authors are to be considered in alphabetical order.

1. INTRODUCTION

Recent developments in information technology have brought about important changes in distributed computing. New environments such as massively large-scale, wide-area computer networks and mobile ad-hoc networks have emerged. These environments represent an enormous potential for future applications: they enable communication, storage and computational services to be built in a bottom-up fashion, often at very low costs.

Yet, these new environments present new challenges because they are extremely *dynamic*, *unreliable* and often *large-scale*. Traditional approaches to distributed system design which assume that the system is composed of reliable components, or that the system scale is modest, are not applicable for these environments. Approaches based on central and explicit control over the system as a whole are not feasible either for the same reasons. In addition, central control introduces a single-point-of-failure which should be avoided whenever possible. It is therefore important to explore approaches that avoid these drawbacks.

Seeking inspiration from the study of biological processes and organisms is one possibility for coping with these problems. It is well known that living organisms can effectively organize large numbers of unreliable and dynamically-changing components (cells, molecules, individuals, etc.) into structures that implement a wide range of functions. In addition, most biological structures (such as organisms) have a number of "nice properties" such as robustness to failures of individual components, adaptivity to changing conditions, and the lack of reliance on explicit central coordination. Consequently, borrowing ideas from nature has long been a fruitful research theme in various fields of computer science. Furthermore, biological inspiration is beginning to make its way into the mainstream of distributed computing after having been a niche topic for a long time [Lodding 2004; Ottino 2004].

In this paper we propose *design patterns* as a conceptual framework for transferring knowledge from biology to distributed computing [Alexander 1977; Gamma et al. 1995]. In its most general sense, a design pattern is a "recurring solution to a standard problem" [Schmidt et al. 1996]. The notion of design patterns is neither novel nor surprising. On the contrary, design patterns emerge from extensive experience and have proven to be successful for solving certain types of problems repeatedly. This explains why the *biological evolution* of organisms must be a rich source of design patterns that work; if a certain species has survived until today, then the solutions that it applies to solve all problems related to survival — from the functioning of a single cell to the cooperation among the members of a population — must be well tested and reliable. Especially if some of these design patterns are observed several times and applied in different contexts, as it often happens in evolution, we can be sure to gain significant knowledge by studying them.

The motivation of the present work is that large-scale and dynamic distributed systems have strong similarities to some of the biological environments. This makes it possible to abstract away design patterns from biological systems and to apply them in distributed systems. In other words, we do not wish to extract design patterns from software engineering practice, as it is normally done. Instead, we wish to extract design patterns from biology and we argue that they can be applied fruitfully in distributed systems.

We identify a number of design patterns common to various biological systems, including plain diffusion, replication, stigmergy and chemotaxis. Design patterns represent a bridge between biological systems and computer systems. The basic idea is to formulate them as local communication strategies over arbitrary (but sparse) communication topologies. We show through examples how to implement practically relevant functions for distributed computing based on these ideas. Using a common evaluation methodology, we show that the resulting functions have state-of-the-art performance while they inherit desirable properties of biological systems including adaptivity and robustness.

The outline of the paper is as follows. In Section 2 we describe the common context of all the design patterns that are identified in the paper. Section 3 presents the design patterns themselves. Section 4 discusses principles of the evaluation methodology of the examples of the design patterns, followed by the actual evaluations; Sections 5 to 7 describe four examples of distributed services in this framework: data aggregation, load balancing and search in overlay networks, and routing in ad hoc networks. Section 9 discusses related work and Section 10 concludes the paper.

2. COMMON CONTEXT OF PATTERNS

In the literature, design patterns (*pattern* for short) appear in many different contexts and are presented in different ways. Most of the attempts follow the principles of Alexander [Alexander 1977], or the same principles adapted in object-oriented design as advocated by Gamma et al. [Gamma et al. 1995]. Based on these works, we will present our patterns by describing the following attributes: *name, context, problem, solution, example*, and finally, *design rationale.*

The meaning of these attributes should be self-explanatory, except perhaps in the case of *context*, which is the subject of this section. The context is defined by the system model: the participants and their capabilities, the constraints on the way they can interact, and, optionally, any services that are available in the system. Most importantly, a significant portion of the context for all patterns we identify is common. In this sense, they form a natural *family* of patterns.

A key feature of the context description is that it is formulated using the *same* system model for distributed systems and biological systems. In other words, the dynamic distributed environments described in the Introduction, in particular, large-scale wide-area networks and mobile ad-hoc networks, and many biological systems we use as inspirations, share the same communication structure. This fact allows us to “import” patterns from biology. The several mappings of this system model onto biology will be explained in the design rationale of each pattern, while the mapping to distributed systems is given in this section.

2.1 System Model

Our basic system abstraction is a network, along which the network nodes communicate via message passing. This abstraction, however, is overly general. To define a meaningful context for the patterns, we need to specify additional key assumptions that define the properties of the components of the network, and the properties of the network as a whole.

The basic components of our system model are *nodes.* The nodes are typically computing devices that can maintain some state, and perform computations. Each node i has a set of *neighbors* defined as the subset of nodes to which i can send messages. We will often call this set of neighbors the *view* of a node. The message passing mechanism is asynchronous in the sense that message delivery is not guaranteed within a fixed time bound. Nodes may fail, can leave or join the system any time. Messages may be lost. The size of the view — the number of neighbors — is typically much smaller than the total number of nodes in the system.

In this model, we can identify the *topology* of a network as a crucial characteristic. The topology is given by the graph defined by the “neighbor” relation defined above. That is, each node has a view, which contains other nodes. If node j is in the view of node i, we say there is a directed edge (i, j) in the topology. Different properties of the topology crucially define the performance of most message passing protocols. For example, the minimal number of steps to reach a node from another node, or the probability that the network becomes partitioned as a result of failures, can all be expressed in graph theoretical terms. Recent advances in the field of complex networks further underline the importance of network topology [Albert and Barabási 2002]. Accordingly, throughout the paper we shall pay special attention to topology, both in terms of design and evaluation.

2.2 Example Networks

As mentioned before, this model serves as a bridge between biological and computer systems. The mapping of this model to several biological systems is delayed until the definition of the patterns. Here we discuss two examples of distributed computer systems, that can be characterized by the model: overlay networks and mobile ad-hoc networks, which are the environments of interest in this paper.

2.2.1 *Overlay Networks.* Recent research in peer-to-peer systems has revealed that one of the most promising paradigms for building applications over large scale wide area networks is through *overlay networks* [Risson and Moors 2004]. Overlay networks are logical structures built on top of a physical network with a routing service. The fact that the physical network is routed means that, in principle, any node can send a message to any other node provided it knows the target node's network address. Despite this possibility, views of nodes do not, and cannot, contain the entire network, since doing so would require each node to keep track of the global network composition. This is simply not feasible under the large scale and extreme dynamism assumptions.

It is not uncommon for overlay networks to be built in environments consisting of millions of nodes, for example in file sharing peer-to-peer networks. The underlying routing service ensures that in principle any pair of nodes can be connected, so there is a large degree of freedom for defining the actual topology. Yet, the fact that views are limited in size implies that actual overlay networks topologies are restricted. This makes topology construction and maintenance a crucial function in overlay networks.

2.2.2 *Mobile Ad Hoc Networks.* In mobile ad-hoc networks (MANETs) [Royer and Toh 1999] a set of wireless mobile devices self-organize into a network without relying on a fixed infrastructure or central control. All nodes are equal, they can join and leave the network at any time, and can serve to route data for each other in a multi-hop fashion.

In MANETs, neighbor relations in the system model depend on the wireless connections between nodes. The set of nodes that some other node can reach is defined by its transmission power and the physical proximity between the nodes. Unlike overlay networks, we cannot take a routing service for granted, and the only means of communication in our model is therefore explicit point-to-point radio transmission. Furthermore, like in overlay networks, topology of MANETs is also restricted. This is in part due to the limited power of the nodes, which means that they are typically not able to cover the entire span of the network. The problem of interference also restricts the transmission range, independent of power constraints. Nodes can transmit only when the frequency is free. If the transmission range is too large, there will be many overlapping transmissions which render the network unusable. In contrast to overlay networks, in MANETs, topology is given by the physical location of the nodes. By changing the transmission power of the nodes (and therefore the range), it is possible to tune the topology, but in a much more limited sense.

3. OVERVIEW OF PROPOSED PATTERNS

As mentioned before, we present our patterns by describing the following attributes: *name, context, problem, solution, example*, and finally, *design rationale*. Out of these attributes, we have already described *context* in Section 2: it is common to all patterns. The *name* attribute is by convention the respective section title. The detailed *examples*, along with a thorough evaluation, are discussed in separate sections for clarity.

One interesting feature of biological systems is that the *problem* that a given mechanism solves is typically not unique. In other words, the same mechanism typically solves many different problems. Accordingly, in the description of the patterns, we list several problems, focusing on the typical cases. However, the *solution* is unique to each pattern. Finally, the attribute *design rationale* explains where the pattern came from and why it works. In our case, the design rationale involves the discussion of the biological manifestations of the pattern, and a brief description of the insight why they function efficiently.

3.1 Plain Diffusion

Problem 1. Assume that all nodes are assigned numeric values, x_i for node i, and the sum of these values is $x = \sum_1^N x_i$, where N is the network size. The problem is to bring the system to a state in which all nodes are assigned the average value x/N.

Problem 2. As before, assume that all nodes are assigned numeric values, x_i for node i. We want to assign a *gradient* to each link at a node that is proportional to the change in values when following the link — positive if the values increase, negative if they decrease.

Solution. Relying only on message passing and the restricted topology inherent in the context, the solution is very simple. For each of its links, each node periodically subtracts a fixed proportion from its current value and sends it along the given link. When a node receives a value in a message, it adds it to its current value. Note that this ensures that the sum of all values in the system remains a constant. This solution solves problem 1 above because very quickly the values at all nodes will approach the average value. Furthermore, during the process, gradients are also naturally generated: if a given link has a net positive flow towards a node, then it must lead to a high value region and vice versa.

Design Rationale. The solution described above is a form of diffusion, a simple yet ubiquitous process that can be observed in a wide range of biological and physical systems [Murray 1990]. Diffusion involves equalizing the concentration of some substance or some abstract quantity, like heat or electrical potential. It is known to be very efficient in both converging to a state when the concentrations are equal (if the system is mass conserving), and creating gradients (if the system is not mass conserving). A possible mapping of the abstract model to a biological process is given for illustration.

node	Nodes are idealized portions of space.
neighbor	Defined by the topology of the space in which diffusion takes place. In biological systems it is often modeled as a 2- or 3-dimensional regular grid.
message	The actual material that is sent to the neighbor. It is typically modeled as a non-negative real number.

Example. Plain diffusion is applied in Section 5 in the context of the averaging problem, and in Sections 8 and 7 in the context of the gradient problem.

3.2 Replication

Problem 1. Assume that a given node receives a novel piece of information (e.g., database update). The problem is to propagate this information to all other nodes.

Problem 2. Assume that all nodes are assigned numeric values, x_i for node i. The problem is to bring the system to a state in which all nodes are assigned the maximal value $\max_i x_i$.

Problem 3. Assume that nodes hold some data that can be a simple ID, or more complex information, such as a document. The problem is to find a node whose document matches a given query (e.g., keywords in a document).

Solution. A possible solution to these problems is based on replication. In its abstract form, the nodes receive messages from their neighbors, and they forward (that is, replicate) some of the messages they received according to application-specific rules. In the information propagation problem, the nodes simply copy all new pieces of information they receive to all neighbors. This strategy is called flooding. However, more efficient variants exist where the nodes apply a more clever rule for forwarding, taking into account elapsed time, the number of times they received the same information, etc. In the case of maximum finding, the messages are candidates for the maximum value, and nodes keep, and forward the maximal value they have received locally. Finally, in the case of search, the pattern is applied to search queries, that are replicated and forwarded until a match is found. Again, there is lots of room for optimizing the actual strategy according to which the query is replicated, for example, based on information about the topology or characteristics of the data being stored at the nodes.

Design Rationale. Efficient and successful replication-based processes are commonplace in nature. Examples include growth processes, signal propagation is certain neural networks [Arbib et al. 1997], epidemic spreading [Bailey 1975], or proliferation processes in the immune system [Janeway et al. 2001]. As an example, we present the mapping of the abstract model to epidemics.

node	Potential hosts of a virus.
neighbor	Physical proximity, sexual contact, social relationships, etc.
message	The message is the infective agent (e.g., virus). Typically it is transmitted unchanged. It can also mutate in the host and be transmitted in its mutated form.

Example. The pattern has been successfully used for information propagation in the past [Demers et al. 1987]. In this paper, replication is applied in Section 5 in the context of the maximum finding problem, and in Sections 6 and 7 in the context of the search problem (in different, customized forms).

3.3 Stigmergy

Problem 1. Assume that the links between nodes are assigned weights, and we fix two nodes, i and j. The problem is to find the shortest path between i and j.

Problem 2. Each network node holds a number of different items, each with a certain attribute. The objective is to redistribute the items over a small number of nodes (proportional to the number of different attributes) such that items with similar attributed are held at the same node.

Solution. A possible solution is based on a generic mechanism called *stigmergy* [Theraulaz and Bonabeau 1999]. Each node contains a set of variables, called *stigmergic variables.* Nodes generate messages, and send them to neighbors, according to an application dependent policy, that is a function of the stigmergic variables. The reception of a message at a node triggers an action, the nature of which is defined by the information in the message and the stigmergic variables of the node. The action typically consists of updating the stigmergic variables of the node, as well as the information in the message, and forwarding the message until it meets an application-specific objective. Since changes in the stigmergic variables are persistent, the change triggered by a message will influence the way subsequent messages are dealt with and the way their objectives are realized. The stigmergic variables represent the local parameters of the *decision policy* at the nodes. The repeated updating of these parameters in the direction of locally *reinforcing* the decisions which led to a good realization of message objectives gives rise to a *distributed reinforcement learning process* (e.g., [Sutton and Barto 1998]).

In the shortest path problem, node i repeatedly sends messages with the objective to find node j. The path followed by the message is influenced by stigmergic variables at intermediate nodes, and these stigmergic variables are in turn updated to reflect an estimate of the cost to reach j, using information stored in the messages. In the clustering problem, the stigmergic variables are the currently stored items and their properties, and messages contain items as well. These items in turn influence the probability that a given other item in an arriving message stays at a given node or is forwarded on to a neighbor.

It is worth pointing out that in the literature stigmergy is usually described in terms of mobile agents moving through a passive environment, communicating indirectly via modifications they make to stigmergic variables distributed in their environment [Theraulaz and Bonabeau 1999; Keil and Goldin 2005]. While this is often a more natural way of describing stigmergic processes in biology, it turns out that their engineered counterparts are usually implemented with active environment nodes communicating through passive messages, as described above.

Design Rationale. Stigmergic processes can account for a variety of distributed self-organizing behaviors, across diverse social systems, from insects (e.g., nest building, labor division, path finding) to humans [Fewell 2003; Camazine et al. 2001]. As an example, we present the mapping of the abstract network model to the shortest path finding mechanism of an ant colony (see Section 7 for a detailed description of this behavior).

node	Nodes are idealized portions of space. Stigmergic variables are levels of pheromone intensity left by ants while moving in their environment.
neighbor	The neighbor relation between nodes is defined by the physical possibility of ants to move between the locations corresponding to the nodes.
message	Messages are the ants themselves.

Example. Stigmergy is applied in Section 7 to find shortest paths, and so to help route data packets, in mobile ad hoc networks.

3.4 Composite Design Patterns

Patterns are normally combined when used to implement applications. For example, the example presented in Section 7, routing in mobile ad hoc networks, relies on all the patterns described so far.

However, in some cases, there are recurring combinations of certain patterns that can themselves be considered as a composite pattern. In this section we describe two of these: *chemotaxis* and *reaction-diffusion.*

3.4.1 *Chemotaxis.*

Context. In this case the context (described in Section 2) is extended by the presence of *plain diffusion.* In other words, to apply chemotaxis, we need to have some sort of diffusion present in the system, that generates gradients, as described in Section 3.1.

Problem. Find a short path from a given node to regions of the network where the concentration of a diffusive substance is maximal.

Solution. The solution is simply to follow the maximal gradient. That is, starting from the given node, we select the link with the highest gradient, and we repeat this procedure until we find a local maximum concentration.

Design Rationale. When cells or other organisms direct their movements according to the concentration gradients of one or more chemicals (*signals*) in the environment, we talk about chemotaxis. Chemotaxis is responsible for a number of processes that include certain phases of the development of multicellular organisms and pattern formation. Note that the *time scales* of signal diffusion (*chemo*) and cell motion following the gradient (*taxis*) are usually different: signal diffusion needs to be faster to provide useful guidance even in regions that remain distant from the maximal concentration.

Example. Section 8 compares techniques for load balancing based on chemotaxis with simpler techniques based on plain diffusion.

3.4.2 *Reaction-Diffusion.* We do not present examples for reaction-diffusion in this paper, but due to its importance, we briefly mention it here. Reaction-diffusion is not a pattern, but a general framework covering a large number of patterns. Indeed, reaction-diffusion is powerful enough to support a standalone computing paradigm, reaction-diffusion computers [Adamatzky et al. 2005]. Therefore it does not make sense to try to define what kind of specific problems are being solved.

Still, reaction-diffusion can be considered a powerful generalization of the plain diffusion pattern, involving the simultaneous diffusion of one or more materials and allowing for addition or removal of these materials, potentially as a function of the actual concentration of each material. The name "reaction" refers to this potential interaction between the materials present in the system. Reaction-diffusion models have been applied successfully to explain a wide range of phenomena such as pattern formation and developmental processes [Murray 1990].

4. EVALUATION METHODOLOGY

An important motivation for the study of bio-inspired methods is something that we called the "nice properties" of living systems in the Introduction. That is, we observe that living systems are self-repairing, self-organizing, adaptive, intelligent, etc. We can in fact encapsulate most of what we mean by nice properties in a single word: *insensitivity.* Let us now clarify what we mean by that. First, engineered systems are evaluated according to human norms, according to what is good and what is not. If we quantify such evaluation, in a general way, we would call the result a "figure of merit". The measured value of a figure of merit is of course dependent on many things, which we loosely break down into two categories: the system (protocol, algorithm) that is being evaluated, and the "environment", which may be described quantitatively in terms of environmental variables. Obvious examples of the latter include network topology, the load or stress, failures, fluctuations, etc. An insensitive system will then show little variation in the set of figure of merits describing its performance, as the environment is varied.

Now we comment on a few, more familiar, words that are viewed by many as nice properties. First we mention *scalability.* Here we interpret the environmental variable to be the system size (as measured by some parameter such as number of nodes N). Note that in general it is not realistic to require that a figure of merit be totally insensitive to system size (although in Section 5 we will see an example). Next we address the term *robustness.* We also view robustness as a type of insensitivity. Here the environmental variable is a quantitative measure of *damage* to the system whose performance is being evaluated. Finally we define *adaptivity* as insensitivity for all environmental variables other than system size and damage.

These definitions are very schematic; but they lend themselves readily to being rendered quantitative. Here we offer them not as final answers to the problem of relating living systems, engineered systems, and nice properties, but rather to stimulate further thought and discussion.

Finally, we note that our schematic definitions allow for very many quantitative realizations — there are many environmental variables to be varied, and many choices of where and how to measure insensitivity. We do not however view this as a drawback. In fact, we find the general unifying notion of insensitivity to be appealing. In this sense, nice properties are not more difficult to define for engineered systems than for living systems: the latter must simply persist, survive, and reproduce, in the face of the fluctuating environment, while the former must maintain their own corresponding figures of merit.

5. PLAIN DIFFUSION PATTERN EXAMPLE: DATA AGGREGATION

As described in Section 3.1, the plain diffusion pattern is suitable, among other things, for calculating the *average* of some quantity. In other words, plain diffusion allows us to implement protocols that inform all participating nodes about the average of the values of some attributes of the nodes.

The averaging problem, and in general, the problem of calculating global functions over the set of locally known quantities is known as the *distributed aggregation problem* [van Renesse 2003]. The calculated aggregates serve to simplify the task of controlling, monitoring and optimizing distributed applications. Additional aggregation functions include finding extremal values of some property, computing the sum, the variance, etc. Applications include calculating the network size, total free storage, maximum load, average uptime, location and intensity of hotspots, etc. Furthermore, simple aggregation functions can be used as building blocks to support more complex protocols. For example, the knowledge of average load in a system can be exploited to implement near-optimal load-balancing schemes [Jelasity et al. 2004].

This section presents a detailed example, which illustrates how to apply the plain diffusion pattern to calculate averages, how to calculate more complicated functions based only on the average of certain quantities, and finally, evaluates the resulting protocol's efficiency and robustness.

5.1 The Algorithm

Our basic aggregation protocol is shown in Figure 1. Each node p executes two different threads. The *active* thread periodically initiates an *information exchange* with a peer node q selected randomly among its neighbors, by sending q a message containing the local state s_p and waiting for a response with the remote state s_q. The *passive* thread waits for messages sent by an initiator and replies with the local state. The term push-pull refers to the fact that each information exchange is performed in a symmetric manner: both peers send and receive their states. Even though the system is not synchronous, we find it convenient to describe the protocol execution in terms of consecutive real time intervals of length δ called *cycles* that are enumerated starting from some convenient point.

Method UPDATE builds a new local state based on the previous one and the

```
do forever
  wait($\delta$ time units)
  $q \leftarrow$ GETNEIGHBOR()
  send $s_p$ to $q$
  $s_q \leftarrow$ receive($q$)
  $s_p \leftarrow$ UPDATE($s_p, s_q$)
```

(a) active thread

```
do forever
  $s_q \leftarrow$ receive(*)
  send $s_p$ to sender($s_q$)
  $s_p \leftarrow$ UPDATE($s_p, s_q$)
```

(b) passive thread

Fig. 1. Protocol executed by node p.

remote state received during the information exchange. The output of UPDATE depends on the specific function being implemented by the protocol. For example, to calculate the average, each node stores a single numeric value representing the current estimate of the aggregation output. Each node initializes the estimate with the local value it holds. Method UPDATE(s_p, s_q), where s_p and s_q are the estimates exchanged by p and q, returns $(s_p + s_q)/2$. After one exchange, the sum of the two local estimates remains unchanged since method UPDATE simply distributes the initial sum equally among the two peers. So, the operation does not change the global average either; it only decreases the variance over all the estimates. With this implementation of UPDATE, the protocol represents an instantiation of the plain diffusion pattern.

We note here however, that aggregates other than the average can also be computed. For example, for calculating the maximum, UPDATE returns the maximum of its parameters. As a result, the maximal value will be broadcast to all nodes in an epidemic fashion. Other aggregates are described in [Jelasity et al. 2005]. For example, to calculate the variance, one needs the average and the average of the squares; both obtainable through an instance of the averaging protocol. Other means can be calculated as well. For example, the geometric mean (N-th root of the product) is the exponential of the average of the logarithms. From now on we restrict our discussion to the diffusion pattern (that is, average calculation).

It is easy to see that the value at each node will converge to the true global average, as long as the underlying overlay network remains connected. In our previous work [Jelasity et al. 2005], we presented analytical results for the convergence speed of the averaging protocol. Let σ_i^2 be the empirical variance of the local estimates at cycle i. The *convergence factor* ρ_i, with $i \geq 1$, characterizes the speed of convergence for the aggregation protocol and is defined as $\rho_i = E(\sigma_i^2)/E(\sigma_{i-1}^2)$. In other words, it describes how fast the expected variance of the estimates decreases. If the (connected) overlay network topology is sufficiently random, it is possible to show that for $i \geq 1$, $\rho_i \approx 1/(2\sqrt{e})$. In other words, each cycle of the protocol reduces the expected variance of the local estimates by a factor $2\sqrt{e}$. From this result, it is clear that the protocol converges exponentially and very high precision estimates of the true average can be achieved in only a few cycles, irrespective of the network size, confirming the extreme scalability of our protocol. In other words, we can say that the convergence factor is completely insensitive to network size.

5.2 Simulation Model

The simulation experiments were run using PeerSim [PeerSim], a simulator developed at the University of Bologna. We experimented with the COUNT protocol, that computes the number of nodes present in the system. The COUNT protocol is average calculation over a special starting set of numbers: if the initial distribution of local values is such that exactly one node has the value 1 and all the others have 0, then running the averaging protocol we obtain $1/N$; the network size, N, can be easily deduced from it. COUNT is sensitive to failures due to the highly unbalanced initial distribution and thus represents a worst case. During the first few cycles, when only a few nodes have a local estimate other than 0, their removal from the network due to failures can cause the final result of COUNT to diverge significantly from the actual network size.

The goal of the experiments is to examine the scalability and robustness of the algorithm. To this end, we have run two sets of experiments. The first includes networks of different sizes up to 10^6 nodes and a wide range of different communication topologies. In the second set, the network size is fixed to be 10^5 and the underlying overlay network used for communication is based on NEWSCAST, an epidemic protocol for maintaining random connected topologies [Jelasity et al. 2004].

In all figures, 50 individual experiments were performed for all parameter settings. When the result of each experiment is shown in a figure (e.g., as a dot) to illustrate the entire distribution, the x-coordinates are shifted by a small random value so as to separate results having similar y-coordinates. The size estimates and the convergence factor plotted in the figures are those obtained after 30 cycles.

5.3 Results

To test scalability, we have run COUNT in networks whose size range from 10^3 to 10^6 nodes. Several different underlying topologies have been considered, including the complete graph, random network, scale-free topology, newscast, and several Watts-Strogatz small-world networks with different rewiring probability β. With parameter $\beta = 1$ the Watts-Strogatz model generates a random network, while $\beta = 0$ results in a regular ring lattice. We refer to [Albert and Barabási 2002] for a detailed description of these topologies.

The results are shown in Figure 2. In the case of the topologies that allow for a sufficiently random sampling of neighbors from the entire network, the convergence factor is independent of the network size, and approximates the $1/(2\sqrt{e})$ value, as predicted by the analysis. That is, the protocol is insensitive to the choice of underlying topology, as long as the topology allows for a sufficiently random selection of communication partners from the entire network.

In the second set of experiments we tested robustness to crash failures. The crash of a node may have several possible effects. If the crashed node had a value smaller than the actual global average, the estimated average (which should be $1/N$) will increase and consequently the reported size of the network N will decrease. If the crashed node has a value larger than the average, the estimated average will decrease and consequently the reported size of the network N will increase.

The effects of a crash are potentially more damaging in the latter case. The larger the removed value, the larger the estimated size. At the beginning of an execution,

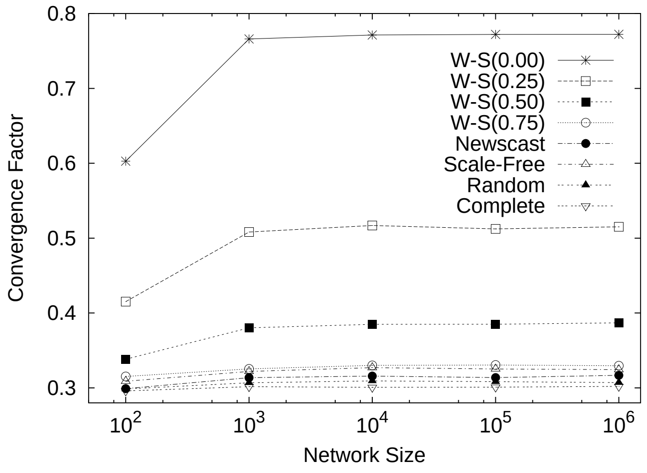
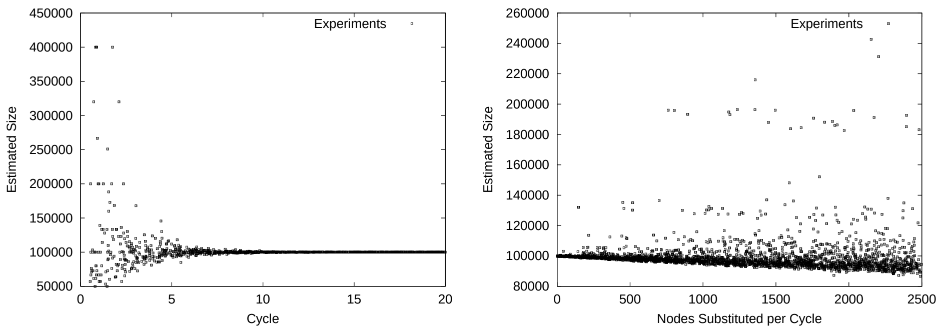


Fig. 2. Average convergence factor computed over a period of 20 cycles in networks of varying size. Each curve corresponds to a different topology where W-S(β) stands for the Watts-Strogatz model with parameter β.



(a) Network size estimation with protocol COUNT where 50% of the nodes crash suddenly. The y-axis represents the cycle at which the "sudden death" occurs.

(b) Network size estimation with protocol COUNT in a network of constant size subject to a continuous flux of nodes joining and crashing. At each cycle, a variable number of nodes crash and are substituted by the same number of new nodes.

Fig. 3. Effects of node crashes on the COUNT protocol in a NEWSCAST network.

relatively large values are present, obtained from the first exchanges originated by the initial value 1. These observations are confirmed by Figure 3(a), that shows the effect of the "sudden death" of 50% of the nodes in a network of 10^5 nodes at different cycles. Note that in the first cycles, the effect of crashing may be very harsh: the estimate can even become infinite (not shown in the figure), if all nodes having a value different from 0 crash. However, around the tenth cycle the variance is already so small that the damaging effect of node crashes is practically negligible.

A more realistic scenario is a network subject to churn. Figure 3(b) illustrates the behavior of aggregation in such a network. Churn is modeled by removing a number of nodes from the network and substituting them with new nodes at each cycle. In other words, the size of the network is constant, while its composition is dynamic.

The plotted dots correspond to the average estimate computed over all nodes that still participate in the protocol after 30 cycles, that is, that were originally part of the system at the beginning. Note that although the average estimate is plotted over all nodes, in cycle 30 the estimates are practically identical. Also note that 2,500 nodes crashing in a cycle means that 75% of the nodes $((30 \times 2500)/10^5)$ are substituted during an execution, leaving 25% of the nodes that make it until the end.

The figure demonstrates that (even when a large number of nodes are substituted during an execution) most of the estimates are included in a reasonable range. The above experiment can be considered as a worst case analysis, since the level of churn was much higher than could be expected in a realistic scenario.

The simulation results presented in this section have been confermed by a real implementation of the protocol run on more than 400 machines on PlanetLab, each of them was executing up to 10 aggregation nodes. Results are presented in Jelasity et al. [2005].

5.4 Discussion

The diffusion design pattern has proven to be an efficient and robust solution for the aggregation problem in overlay networks. We have seen that the protocol is *insensitive* to network size and the communication topology, as long as it is possible to select sufficiently random neighbors at each communication step.

We have seen also that the convergence of the protocol is exponential, in the sense that the variance of the estimates decreases exponentially fast. Exponential behavior has been observed in the context of other applications as well [Parunak et al. 2005]. However, instead of an approximative mapping of a highly simplified model onto our system (as it is done in Parunak et al. [2005]), we were able to characterize convergence quantitatively with a very high precision (see Jelasity et al. [2005] for more details).

Finally, the aggregation problem has been addressed by a number of proposals. There are a number of general purpose systems, the best known of which is Astrolabe [van Renesse et al. 2003]. In these systems, a hierarchical architecture is deployed which reduces the cost of finding the aggregates and enables the execution of complex database queries. However, maintenance of the hierarchical topology introduces additional overhead, which can be significant if the environment is very dynamic. Kempe et al. propose an aggregation protocol similar to ours, tailored to work on random topologies [Kempe et al. 2003]. The main difference is that their discussion is limited to theoretical analysis, while we consider the practical details needed for a real implementation and evaluate our protocol in unreliable and dynamic environments.

6. REPLICATION PATTERN EXAMPLE: SEARCHING

In Section 3.2 we have seen that replication is a pattern that can be observed in a wide range of biological functions. When applied to the distributed search problem, the replication pattern is used to spread queries, by the nodes making "clones" of the queries they receive according to some strategy. The production of clones necessarily incurs some overhead. Hence, to effectively use this design pattern, two opposing objectives need to be fulfilled: higher efficiency and lower overhead. The replication strategy used by the search algorithm discussed in this section is aimed at achieving these objectives.

Our search algorithms are designed for *unstructured* overlay networks — those where there is no relation between the information stored at a node and its position in the overlay topology. This is in contrast with other structures like *Distributed Hash Tables* where the position of a node in the topology determines exactly which data it may store. Unstructured overlay networks are attractive for a number of reasons. They are extremely easy to maintain, and they are highly robust to failures and other sources of dynamism (churn). Furthermore, search algorithms implemented over unstructured networks can support arbitrary *keyword based* searches [Chawathe et al. 2003].

As mentioned before, the replication pattern can support a number of different strategies. For example, *flooding* (*unbridled replication*) techniques have generally been used to implement search in unstructured networks. Although flooding fulfills the criterion of robustness, and also gives very fast results, it produces a huge number of query messages which ultimately overwhelm the entire system. This is a well known problem with the first generation Gnutella networks. The alternative slower-but-efficient method is to perform the search operation using k-random walkers (*no replication*) [Lv et al. 2002]. In this section, we report search algorithms based on *proliferation* — a specific replication strategy inspired by the immune system. We will show that our proliferation algorithm (*controlled replication*), when constrained to produce a number of messages comparable to the k-random walker algorithm, is significantly faster in finding the desired items.

Our algorithm has been inspired by the simple mechanism of the *humoral immune system*, where B cells, upon stimulation by a foreign agent (antigen) undergo proliferation generating antibodies [Janeway et al. 2001]. In our terminology, this mechanism represents an instance of the replication pattern. Proliferation helps in increasing the number of antibodies that can then efficiently track down the antigens (foreign bodies). In our problem, the query message is conceived as an antibody which is generated by the node initiating a search, whereas antigens are the searched items hosted by other nodes of the overlay network. As in the natural immune system, the messages undergo proliferation based on the affinity measure between the message and the contents of the node visited, which results in an efficient search mechanism. Additional details have been reported in various conference proceedings [Ganguly et al. 2005; Ganguly et al. 2004a; 2004b; Ganguly and Deutsch 2004b; 2004a].

6.1 Algorithms

In this section, we introduce two proliferation-based search algorithms. All nodes in the network run exactly the same algorithm. The search can be initiated from any node in the network. The initiating node sends $k \geq 1$ identical query messages to k of its neighbors. When a node receives a query Q, it first calculates the number of local hits generated by Q. Subsequently, the node processing the query forwards the same query to some of its neighbors. The exact way in which the forwarding is implemented differs for the various algorithm variants:

Random walk (RW). The received query is forwarded to a random neighbor.

Proliferation (P). The query possibly undergoes proliferation at each node it visits, in which case it is forwarded to several neighbors. The node first calculates the number of messages it needs to forward (η_p) using a *proliferation controlling function.* The proliferation controlling function is defined based upon the model we take into consideration; however, the essence of the function is that proliferation increases as the similarity between the query message and the contents of the node increases.

All forwarding approaches have a corresponding *restricted* version. Restricted forwarding means that copy of a query is sent to a *free* neighbor — one that has not been visited previously by the same query. The idea behind this restriction is that this way we can minimize redundant network utilization. If the number of free neighbors is less than the number of query-copies, then only the free neighbors will receive a copy. However, if there is no free neighbor at all, one copy of the query is forwarded to a single random neighbor. The restricted versions of the above protocols will be called *restricted random walk* (RRW) and *restricted proliferation* (RP).

6.2 Simulation Model

In order to test the efficiency of the proposed algorithm, we build a simple model of a peer-to-peer network. In the model we focus on the two most important aspects of a peer-to-peer system: network topology, and query/data distributions. For simplicity, we assume that the topology and the distributions do not change during the simulation of our algorithms. For the purpose of our study, if one assumes that the time to complete a search is short compared to changes in network topology and query distribution, results obtained from the stationary settings are indicative of performance in real systems.

Network topology.. We consider random graphs generated by the Erdős-Rényi model, in which each possible edge is included with some fixed probability p. The average node degree is therefore Np where N is the total number of nodes, and the node degree follows a Poisson distribution with a very small variance. Overlay networks that approximate this topology can be maintained through simple distributed protocols [Jelasity et al. 2004]. In the rest of this section we fix the network size to be $N = 10000$, and the average node degree to be $Np = 4$.

Data distribution.. Files are modeled as collections of keywords [Lee et al. 1997]. Hence the data distribution is represented in terms of keywords. We assume that there are 2000 different keywords in the system. Each node stores some number of keywords. The number of keywords (not necessarily unique) at each node follows a Poisson distribution with mean 1000. The data profile of a node is denoted $D = \{(\delta_1, n_1), (\delta_2, n_2), \cdots\}$ where δ_i are unique keywords and n_i are their respective frequencies at the node. The 2000 possible keywords are distributed over the nodes in the system such that the resulting global frequency of keywords follows Zipf's distribution[Zipf 1935].

Query distribution.. A query is a set of keywords $Q = \{q_1, q_2, \cdots\}$. Queries are generated according to the following model: 95% of them contain 5 or fewer keywords, while the remaining 5% contain 6 to 10 keywords. In both cases, the actual number of keywords contained in a query is selected randomly uniform over the respective length interval. The actual keywords contained in a query are selected from the same (Zipf's) distribution as in the data model.

Based upon the above models for data and queries, the *number of hits* as well as the *proliferating controlling function* is defined below.

Number of hits.. When a node with data profile D receives a query Q, it generates the number of local hit (S_l) as follow:

$$S_l = \sum_{i=1}^{K} \sum_{j=1}^{\|Q\|} (q_j \oplus \delta_i) n_i \tag{1}$$

where $q_j \oplus \delta_i = 1$ if $q_j = \delta_i$, otherwise 0, the total number of (not necessarily unique) keywords in D is $K = \sum_i n_i$. The number of successful matches calculated this way is then recorded to calculate search statistics.

Proliferation Controlling Function.. As has been stated in Section 6.1, the number of copies to be forwarded to the neighboring nodes, η_p, is determined through the proliferating controlling function. The proliferation of queries at a node is heavily dependent on the similarity between the query and the data profile of the node in question. We define the measure of similarity between the data profile D of the node and a query Q as S_l/K where S_l is as defined in Equation (1). Note that $0 \leq S_l/K \leq 1$. The number of copies to be forwarded is defined as

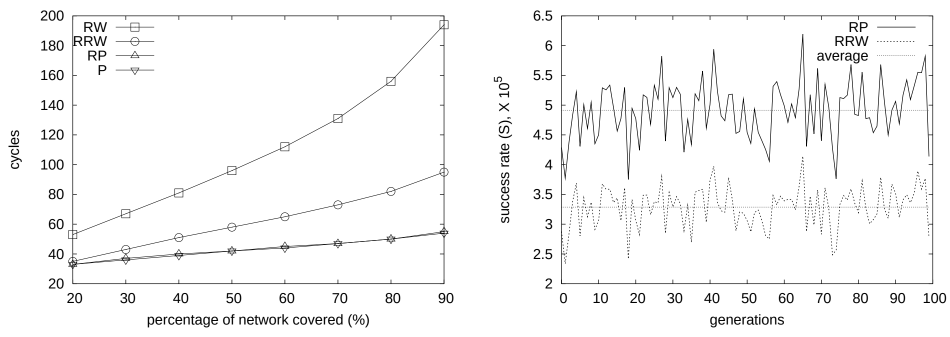
$$\eta_p = 1 + \frac{S_l}{K}(\eta - 1)\rho \tag{2}$$

where η represents the number of neighbors the particular node has, and $\rho \leq 1$ is the proliferation constant ($\rho = 0.5$ in all our experiments). The above formula ensures that $1 < \eta_p \leq \eta$.

6.3 Experimental Results

In this section we compare random walk and restricted proliferation. The overlay network and the query and data distributions are as described in Section 6.2. The experiments focus on efficiency aspects of the algorithms, and use the following simple metrics that reflect the fundamental properties of the algorithms:

Network Coverage. The amount of time required to cover (visit) a given percentage of the network.



(a) Network coverage of all the protocol variants.

(b) Search efficiency of RP and RRW

Fig. 4. Experimental results on network coverage (a) and search efficiency (b).

Search Efficiency. The number of similar items found by the query messages within a given time period.

Both proliferation and random walk are distributed algorithms and the nodes perform the task independently of the others. However, to assess the speed and efficiency of the algorithm, we have to ensure some sort of synchronous operation among the peers. To this end, we require all nodes to execute the algorithm exactly once in a fixed time interval thereby defining *cycles* of the system as a whole. That is, if a node has some messages in its message queue, it will process one message within one cycle, which includes calculating the number of hits and forwarding the copies of the query. The interpretation of cycle is very similar to the other applications presented throughout the paper. Nodes are shuffled at each cycle, to guarantee an arbitrary order of execution. The length of the message queue is assumed to be unbounded.

To ensure fair comparison among all the processes, we must ensure that each protocol is assigned the same "power". To provide fairness for comparison of the proliferation algorithms with random walk, we ensure that the total number of transmitted query messages is the same in all the cases (apart from integer rounding). Query transmissions determine the cost of the search; too many messages cause network congestion, bringing down the efficiency of the system as a whole. It can be seen that the number of transmitted messages increases in the proliferation algorithms over time, while it remains constant in the case of random walk algorithms. Therefore, while performing a particular experiment, the initial number of messages k in all the protocols is chosen in a fashion so that the aggregate number of message transmissions used by both random walk and proliferation is the same. Parameter k is set to be the out-degree of the initiating node for proliferation, and for the rest of the algorithms it is calculated as discussed earlier. To ensure fairness in "power" between the two proliferation algorithms P and RP, we keep the proliferation constant ρ and the value of k the same for both algorithms.

6.3.1 *Network Coverage.* Here we are interested in how rapidly the protocols reach a given proportion of the network. We ran all the protocols 1000 times from randomly selected starting nodes, and for all percentage values shown in Figure 4(a) we calculated the average number of cycles needed to visit that percentage of the nodes. The fairness criterion was applied as follows: first proliferation is run with k being set to the out-degree of the initiating node, until it covers the network (say, in n_c cycles) and the overall number of messages transferred is calculated (say, n_m). Parameter k for the random walker is then initialized to be $k = n_m/n_c$. The random walker is then run until it covers the network. Note that it typically needs more cycles than proliferation, so in fact we have a slight bias in favor of the random walker, because (especially in the initial phase) it is allowed to transfer much more messages than proliferation.

In figure 4(a) it is seen that P and RP need an almost identical number of cycles to cover the network. This time, however, is much smaller than that needed by RRW and RW. Algorithm RRW is much more efficient than RW. Simple proliferation (run with the same proliferation constant ρ as RP), produces many more messages than RP (not shown). So, although P and RP produce similar results in terms of coverage times, we can conclude that the restricted versions of both the random walk and proliferation algorithms are more efficient.

6.3.2 *Search Efficiency.* Since we have seen that in both cases the restricted versions are more efficient, we focus only on the restricted variants: RRW and RP. To compare the search efficiency of RP and RRW, we performed 100 individual searches for both protocols to collect statistics. We repeated this 100 times, resulting in 100000 searches performed in total. In each experiment a search is started from a random node and run for 50 cycles. Apart from a different k parameter (chosen based on the fairness criterion described above), the two protocols are run over the same system, starting from the same node with the same query.

We call one set of 100 experiments (used to calculate statistics) a *generation.* That is, each generation consists of 100 searches. In each search, we collect all the hits in the system, summing up the number of local hits (S_l) at all the nodes (calculated according to (1)) over the 50 cycles. The value of the success rate, S, is the average of the number of hits over the 100 searches in a generation.

Figure 4(b) shows S for all generations for RP and RRW. In this figure, we see that the search results for both RP and RRW show fluctuations. The fluctuations occur due to the difference in the availability of the searched items selected at each generation. However, we see that on the average, search efficiency of RP is almost 50% higher than that of RRW. (For RP, the number of hits is approximately 5×10^5, while it is 3.2×10^5 for RRW.)

6.4 Discussion

In this section, we have presented experimental results showing that a replication pattern, the simple immune-system inspired concept of restricted proliferation, can be used to search more effectively than random walk. The main reason for this is that proliferation is a more cost-effective way of covering large portions of the network. This feature also makes us believe that the approach can be successfully applied for not only search but also application-level *broadcasting* and *multicasting.*

In [Ganguly et al. 2005] we have also derived a theoretical explanation of the performance of the proliferation algorithm. The theoretical work is still ongoing. We believe the next challenge is to more systematically define an efficient flooding mechanism, a mechanism which will not generate a huge number of messages like traditional flooding but will be just as fast. Speaking more quantitatively, it can be shown that a (multiple or single) random walk requires $O(t^d)$ time to cover a d-dimensional grid network if flooding takes $O(t)$ time [Yuste and Acedo 2000]. Our goal is to design proliferation schemes (*controlled replication*) that will take only $O(t^2)$ time, yet will use a much lower number of message packets than flooding.

7. STIGMERGY PATTERN EXAMPLE: ROUTING IN MOBILE AD HOC NETWORKS

Routing is the task of directing data flows from sources to destinations while maximizing network performance. This is particularly difficult in MANETs due to the constant changes in network topology and the fact that the shared wireless medium is unreliable and provides limited bandwidth. These challenges mean that MANET routing algorithms should be highly adaptive and robust and work in a distributed way, while at the same time, they should be efficient with respect to bandwidth use. Such properties can be expected to result from the implementation of the patterns described in Section 3. In particular we describe a MANET routing algorithm called *AntHocNet* [Di Caro et al. 2005a; Ducatelle et al. 2005b], which uses stigmergy as the main driving mechanism to adaptively learn routing tables. Stigmergic learning is supported by a simple diffusion process. Finally, the replication pattern is also applied in the form of flooding, in certain phases of the protocol. We can therefore say that this protocol takes advantage of three different design patterns.

In what follows we first elaborate in the specific stigmergic and diffusion processes that formed the inspiration of our work, then we give a detailed overview of the algorithm, and finally we show the validity of our approach in a set of experiments.

7.1 Stigmergy and Diffusion for Learning Shortest Paths

We take inspiration from the foraging behavior of ants which allows the colony to find the shortest path between the nest and a food source [Camazine et al. 2001]. The main catalyst of this behavior is the use of a volatile chemical substance called *pheromone*, which acts as a stigmergic variable: ants moving between their nest and a food source deposit pheromone, and preferentially move towards areas of higher pheromone intensity. Shorter paths can be completed quicker and more frequently by the ants, and are therefore marked with higher pheromone intensity. These paths then attract more ants, which in turn increases the pheromone level, finally allowing the colony as a whole to converge onto the *shortest path*. The ant colony foraging behavior has attracted attention as a framework for (distributed) optimization, and has been reverse-engineered in the context of *Ant Colony Optimization* [Dorigo et al. 1999]. In particular, it was the inspiration for a number of adaptive routing algorithms for wired communications networks, such as *AntNet* [Di Caro and Dorigo 1998] (see [Di Caro 2004] for an overview).

Our algorithm is in the first place based on the stigmergic learning process described above. Additionally, we use a diffusion process. We explicitly model the fact that pheromone released by the ants is volatile and spreads around the original path followed by the ant [Mankin et al. 1999]. While in a pure stigmergic model, the stigmergic variables are kept only locally in the nodes, the combination with diffusion allows for them to be spread out, in order to make the learning process more efficient and/or effective.

7.2 The AntHocNet Algorithm

AntHocNet is a *hybrid algorithm*, in the sense that it contains both *proactive* and *reactive* components. The distinction between proactivity and reactivity is important in the MANET community, where routing algorithms are usually classified as being proactive (e.g., OLSR [Clausen et al. 2001]), reactive (e.g., AODV [Perkins and Royer 1999]) or hybrid (e.g., ZRP [Haas 1997]). AntHocNet is reactive in the sense that nodes only gather routing information for destinations which they are currently communicating with, while it is proactive because nodes try to maintain and improve routing information for current communication sessions. We therefore make a distinction between the path setup, which is the reactive mechanism to obtain initial routing information about a destination, and path maintenance and improvement, which is the normal mode of operation during the course of a session and serves to proactively adapt to network changes. The hybrid architecture is needed to improve efficiency, which is crucial in MANETs. The main mechanism to obtain and maintain routing information is a *stigmergic learning* process: mimicking path sampling by ants in biological processes, nodes independently send out messages (referred to as ants in the following) to *sample* and *reinforce* good paths to a specific destination. Routing information is kept in arrays of stigmergic variables, called *pheromone tables*, which are followed and updated by the ants. This mechanism is further supported by the *diffusion process*: the routing information obtained via stigmergic learning is spread between the nodes of the MANET to provide secondary guidance for the learning agents. Data packets are routed stochastically according to the learned pheromone tables. Link failures are dealt with using a local path repair process or via notification messages. In the following we provide a concise description of each of the algorithm's components (however, for lack of space, we will not discuss the rather technical component which deals with link failures). A detailed description and evaluation of AntHocNet can be found in Di Caro et al. [2004; Ducatelle et al. [2005a; Di Caro et al. [2005a; Ducatelle et al. [2005b; Di Caro et al. [2005b].

7.2.1 *Routing Tables as Stigmergic Variables.* We adopt the datagram model of IP networks, where paths are expressed in the form of routing tables kept locally at each node. In AntHocNet, a routing table $\mathcal{T}^i$ at node i is a matrix, where each entry $\mathcal{T}^i_{nd} \in \mathbb{R}$ of the table is a value indicating the *estimated goodness* of going from i over neighbor n to reach destination d. Goodness is a combined measure of path end-to-end delay, number of hops, and radio signal quality, measured via the signal-to-noise ratio. These values play the role of stigmergic variables in the distributed reinforcement learning process: they are followed by ants which sample paths to a given destination, and are in turn updated by ants according to the estimated goodness of the sampled paths (see 7.2.2). The routing tables are therefore termed *pheromone tables.* The learned pheromone tables are used to route data packets in a stochastic forwarding process (see 7.2.4).

7.2.2 *Reactive Path Setup.* When a source node s starts a communication session with a destination node d, and it does not have routing information for d available, it broadcasts a *reactive forward ant.* The objective of the forward ant is to find a path to d. At each node, the ant is either unicast or broadcast, according to whether or not the current node has routing information for d. If pheromone information is available, the ant is sent to next hop n with the probability P_{nd} which depends on the relative goodness of n as a next hop, expressed in the pheromone variable $\mathcal{T}^i_{nd}$

$$P_{nd} = \frac{(\mathcal{T}^i_{nd})^\beta}{\sum_{j \in \mathcal{N}^i_d} (\mathcal{T}^i_{jd})^\beta}, \quad \beta \geq 1, \tag{3}$$

where $\mathcal{N}^i_d$ is the set of neighbors of i over which a path to d is known, and β is a parameter value which controls the exploratory behavior of the ants. If no pheromone information is available, the ant is broadcast. Due to subsequent broadcasts, many duplicate copies of the same ant travel to the destination. A node which receives multiple copies of the same ant only accepts the first and discards the other. This way, only one path is set up initially. Later, during the course of the communication session, more paths are added via the proactive path exploration and maintenance mechanism to provide a *mesh of multiple paths* for data forwarding.

Each forward ant contains a list $\mathcal{P} = [1, 2, \ldots, d]$ of the nodes it has visited. Upon arrival at the destination d, it is converted into a *backward ant*, and sent back to its source over $\mathcal{P}$. The objective of the backward ant is to gather information about the goodness of the path sampled by the forward ant, which allows nodes along the path to update their pheromone table entries towards d. As mentioned in 7.2.1, goodness of a path is expressed in terms of delay, number of hops and radio signal quality. An ant arriving in a node i ($i \in \mathcal{P}$ and $i < d$) from a neighbor node n ($n = i + 1$, the next hop from i in the direction of d on $\mathcal{P}$) contains a value τ^n_d, which is a measure of the goodness of its path from n to d. On reception of the ant, node i will in the first place update τ^n_d to τ^i_d by including the goodness τ^i_n of the hop from i to n. Then i will use this information to update the value $\mathcal{T}^i_{nd}$ in its pheromone table, which is its own estimate of the goodness of going from i over n to d. Finally, i will further forward the ant over $\mathcal{P}$.

To calculate the one hop goodness τ^i_n of going from i to n, i uses in the first place the time it would take a data packet to travel from i to n. This time is given by the estimate $\hat{T}^i_n$, which is maintained locally in i, based on MAC layer measurements. This time is then combined with a fixed cost given to one hop, to calculate the cost c^i_n of going from i to n, as follows:

$$c^i_n = \frac{\hat{T}^i_n + T_{hop}}{2}, \tag{4}$$

where T_{hop} is a fixed value representing the time to take one hop in unloaded conditions. Calculating c^i_n like this is a way to avoid possibly large oscillations in the time estimates gathered by the ants (e.g., due to local bursts of traffic) and to take into account both end-to-end delay and number of hops. Next, c^i_n is modified according to the signal-to-noise ratio snr^i_n of the radio connection between i and n. If snr^i_n is higher than a fixed cutoff value, c^i_n is multiplied by a penalty factor. Finally, to obtain the goodness value τ^i_n, the cost value c^i_n is inverted.

τ^i_n is then combined with τ^n_d to obtain the new goodness value τ^i_d carried by the ant. The pheromone value $\mathcal{T}^i_{nd}$ in i's pheromone table is updated as follows:

$$\mathcal{T}^i_{nd} = \gamma \mathcal{T}^i_{nd} + (1-\gamma)\tau^i_d, \quad \gamma \in [0,1]. \tag{5}$$

Once the backward ant makes it back to the source, a full path is set up and the source can start sending data. If the backward ant for some reason does not arrive, a timer will run out at the source, and the whole process is started again.

7.2.3 *Proactive Path Maintenance and Exploration.* During the course of a communication session, source nodes send out *proactive forward ants* to update the information about currently used paths and to try to find new and better paths. They follow pheromone and update routing tables in the same way as reactive forward ants. Such continuous sampling of paths and pheromone updating by ants is the typical mode of operation in ant inspired routing algorithms [Di Caro 2004]. However, in MANET environments, characterized by constant changes, the needed ant sending frequency is quite high, so that the process gets in conflict with the typically limited bandwidth in such networks. Moreover, to find entirely new paths, too much blind exploration through random walks or broadcasts would be needed, again leading to excessive bandwidth consumption. Therefore, we introduce at this point a supporting *diffusion function* which allows to spread pheromone information over the network. This process provides a second way of updating pheromone information about existing paths, and can give information to guide exploratory behavior.

The pheromone diffusion function is implemented using short messages, passed periodically and asynchronously by the nodes to all their neighbors via a broadcast. In these messages, the sending node n places a list of destinations it has information about, including for each of these destinations d the best pheromone value $\mathcal{T}^n_{m^*d}, m^* \in \mathcal{N}^n_d$, which n has available for d. A node i receiving the message from n first of all updates its view, indicating that n is its neighbor. Then, for each destination d listed in the message, it can derive an estimate of the goodness of going from i to d over n, combining the cost of hopping from i to n with the reported pheromone value $\mathcal{T}^n_{m^*d}$. We call the obtained estimate the *bootstrapped pheromone* variable $\mathcal{B}^i_{nd}$, since it is built up using an estimate which is non-local to i. This bootstrapped pheromone variable can in turn be forwarded in the next message sent out by n, giving rise to a bootstrapped pheromone field over the MANET. This sort of process is typical for Bellman-Ford routing algorithms, which are based on dynamic programming approaches [Bertsekas and Gallager 1992].

Bootstrapped pheromone is used directly for the *maintenance* of existing paths. If i already has a pheromone entry $\mathcal{T}^i_{nd}$ in its routing table for destination d going over neighbor n, $\mathcal{B}^i_{nd}$ is treated as an update of the goodness estimate of this path, and is used directly to replace $\mathcal{T}^i_{nd}$. Due to the slow multi-step forwarding of bootstrapped pheromone, this information does not provide the most accurate view of the current situation. However, it is obtained via a lightweight, efficient process, and is complemented by the explicit path updating done by the ants. In this way we have two updating frequencies in the path maintenance process.

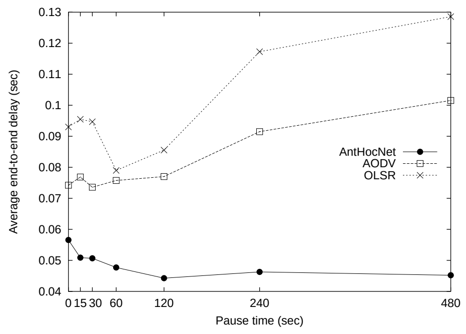
For path *exploration*, bootstrapped pheromone is used indirectly. If i does not yet have a value for $\mathcal{T}^i_{nd}$ in its routing table, $\mathcal{B}^i_{nd}$ could indicate a possible new path from i to d over n. However, this path has never been sampled explicitly by an ant, and due to the slow multi-step pheromone bootstrapping process it could contain undetected loops or dangling links. It is therefore not used directly for data forwarding. It is seen as a sort of virtual pheromone, which needs to be tested. Proactive forward ants will use both the regular and the virtual pheromone on their way to the destination, so that they can test the proposed new paths. This way, promising virtual pheromone is investigated, and if the investigation is successful it is turned into a regular path which can be used for data. This increases the number of paths available for data routing, which grows to a full mesh, and allows the algorithm to exploit new routing opportunities in the ever changing topology.

7.2.4 *Stochastic Data Routing.* Data are forwarded according to the values of the pheromone entries. Nodes in AntHocNet *forward data stochastically.* When a node has multiple next hops for the destination d of the data, it randomly selects one of them, with probability P_{nd}. P_{nd} is calculated in the same way as for reactive forward ants, using equation (3). However, a higher value for the exponent β is used in order to be greedy with respect to the better paths. According to this strategy, we do not have to choose a priori how many paths to use: their number will be automatically selected in function of their quality. The probabilistic routing strategy leads to data load spreading according to the estimated quality of the paths. If estimates are kept up-to-date, this leads to *automatic load balancing.* When a path is clearly worse than others, it will be avoided, and its congestion will be relieved. Other paths will get more traffic, leading to higher congestion, which will make their end-to-end delay increase. By adapting the data traffic, the nodes spread the data load evenly over the network.

7.3 Simulation Model

AntHocNet's performance was evaluated in an extensive set of simulation tests using QualNet [Scalable Network Technologies, Inc. 2003], a widely used commercial simulation packet. We studied the behavior of the algorithm under different conditions for network size, connectivity and change rate, radio channel capacity, data traffic patterns, and node mobility. Performance was measured in terms of data delivery ratio, end-to-end packet delay and delay jitter as *measures of effectiveness*, and routing overhead in number of control packets per successfully delivered data packet as *measure of efficiency.* In addition to these traditional evaluation metrics we also measured other important properties such as scalability, adaptivity and robustness. We present a representative subset of the results of these simulation tests. For the complete set of results we refer to other publications about the algorithm [Di Caro et al. 2004; Ducatelle et al. 2005a; Di Caro et al. 2005a; Ducatelle et al. 2005b; Di Caro et al. 2005b].

The MANET scenarios used in the tests reported on here were all derived from the same base scenario. In this scenario, 100 nodes are randomly placed in an area of $2400 \times 800\ m^2$. Each experiment is run for 900 seconds. Data traffic is generated by 20 constant bit rate (CBR) sources sending four 64-byte packet per second. Each source starts sending at a random time between 0 and 180 seconds after the start of the simulation, and keeps sending until the end. A two-ray path loss model is used in the radio propagation model. The radio range of the nodes is 250 meters,



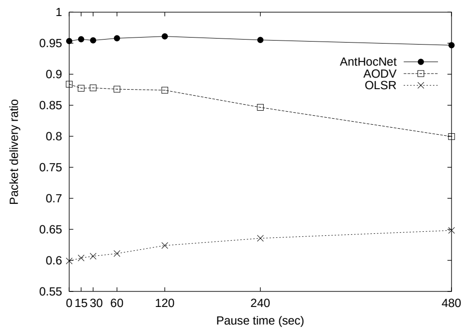


Fig. 5. Average delay and delivery ratio for increasing pause times

and the data rate is 2 Mbit/s. At the MAC layer we use the IEEE 802.11b DCF protocol as is common practice in MANET research. The nodes move according to the *random waypoint* (RWP) mobility model [Johnson and Maltz 1996]: they choose a random destination point and a random speed, move to the chosen point with the chosen speed, and rest there for a fixed amount of pause time before they choose a new destination and speed. The speed is chosen between 0 and 10 m/s, and the pause time is 30 seconds. To assess the performance of our algorithm relative to the state-of-the-art in the field, we compare each time to *Ad-hoc On-demand Distance Vector routing (AODV)* [Perkins and Royer 1999], and *Optimized Link State Routing (OLSR)* [Clausen et al. 2001], two important benchmark algorithms in this area. We have also carried out experiments with other algorithms, such as Dynamic Source Routing (DSR) and Bellman-Ford, but since those algorithms gave much worse results, they were not included here.

7.4 Results

In a first set of experiments we vary the pause time between 0 and 480 seconds. Higher pause time means lower mobility, but also lower connectivity (due to specific properties of RWP mobility, see Bettstetter et al. [2003]). Therefore, the performance of different algorithms can show different trends, and performance behavior can be non-monotonic. The results of the tests are presented in figures 5 (average delay and delivery ratio) and 6 (average jitter and overhead). AntHocNet shows much better effectiveness than AODV and OLSR, in terms of average delay, delivery ratio and jitter. Also in terms of efficiency, AntHocNet outperforms the two other algorithms.

In a second set of experiments, we increase the number of nodes, from 100 to 800 nodes. The MANET area was increased accordingly, to keep the node density constant. The results are presented in Figures 7 and 8. For OLSR we report only results up to 500 nodes, as simulation run times became prohibitively large beyond that, and performance very low. We can see that AntHocNet's advantage over both other algorithms grows for all measures of effectiveness for larger networks. This is an indication that it is a scalable algorithm. Also in terms of efficiency, AntHocNet seems to be scalable: while its overhead is comparable to that of the other algorithms for small networks, it increases less quickly and is much lower for

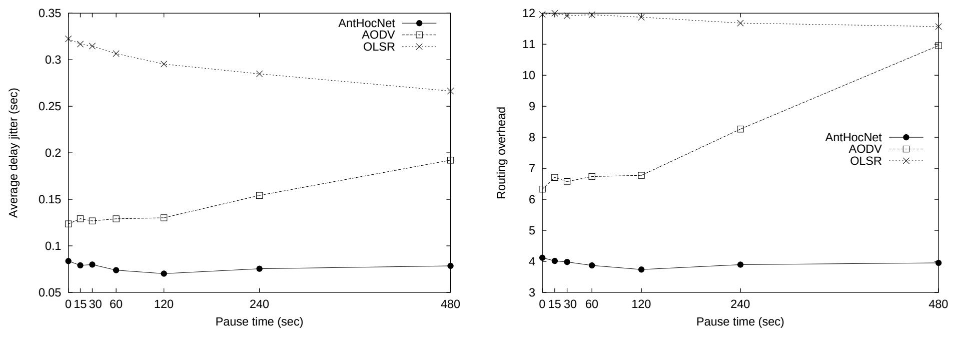


Fig. 6. Average jitter and overhead for increasing pause times

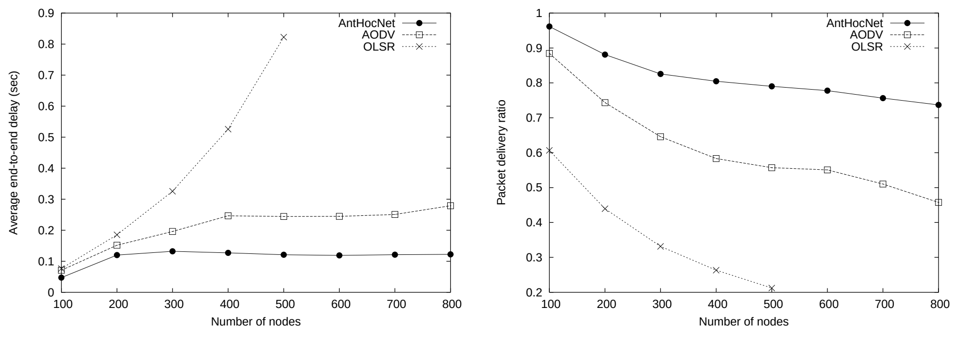


Fig. 7. Average delay (left) and delivery ratio (right) for an increasing number of nodes

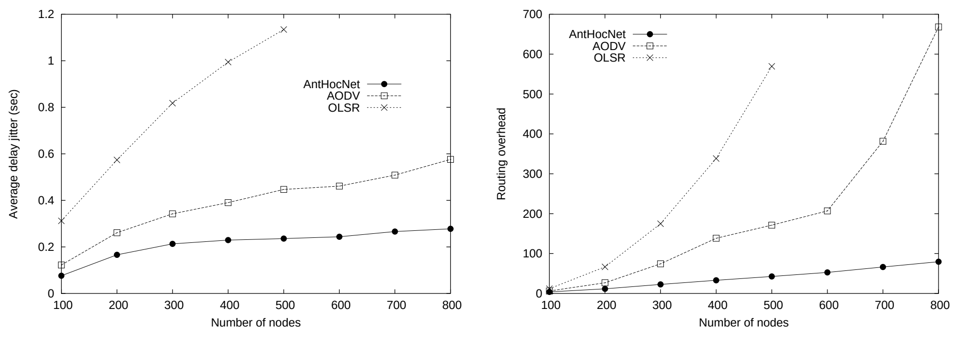


Fig. 8. Average jitter (left) and routing overhead (right) for an increasing number of nodes

the larger networks.

7.5 Discussion

In this section, we have described AntHocNet, a new routing algorithm for MANETs which is designed using the stigmergy and diffusion patterns. In a set of simulation experiments, we show that AntHocNet can outperform important reference algorithms in a wide range of different environments. In particular, the algorithm shows very good scalability, and has robust behavior over the different environments.

The results obtained with the AntHocNet routing algorithm show that the applied design patterns from biology can be useful components to build powerful algorithms for distributed computing. However, we would like to point out that in order to obtain state-of-the-art performance, a significant engineering effort is required in order to adapt the patterns to the specific task. In MANETs, the mobility of the nodes, the unreliability and low bandwidth of the wireless communication, etc. pose specific problems which need to be addressed. For example, we experienced that it was very important to construct a good composite stigmergic variable (the pheromone in this case) which truly grasps the features of good paths that need to be reinforced. That is why we combined the use of number of hops, end-to-end delay and radio signal quality. Furthermore, the use of a stochastic component is important. While stochasticity is not an intrinsic component of the stigmergic pattern as we described it in 3.3, most biological processes combine stigmergy with a stochastic component in order to explore the environment and learn about it. Especially in a dynamic environment like MANETs, exploration is important, and a good balance of the level of stochasticity is important to maintain efficiency and effectiveness. Finally, we realized that using a stigmergic process designed after ant colony behavior had its limitations due to efficiency problems caused by the continuous path sampling. We solved this by combining it with a diffusion pattern, which allows us to get good effectiveness and efficiency at the same time, as shown by the scalability results. This is an example of how a good understanding of the properties of different patterns can allow to compose them into a strong engineering solution.

We would also like to point out that this is not the only attempt at using stigmergy in MANETs and more in general in networking. Starting from early work on AntNet [Di Caro and Dorigo 1998] and ABC [Schoonderwoerd et al. 1996], a number of algorithms have been proposed in wired networks (for an overview, see Di Caro [2004]). For MANETs, examples of existing stigmergy based algorithms for routing are ARA [Günes et al. 2003], PERA [Baras and Mehta 2003] and ANSI [Shen et al. 2004]. For an example of the application of stigmergy to other problems in MANETs, see H. Van Dyke Parunak [2004]. The interested reader can find in these other works additional examples of how stigmergy can be engineered to provide good network algorithms.

8. CHEMOTAXIS PATTERN EXAMPLE: LOAD BALANCING

As described in Section 3.4.1, the chemotaxis design pattern is motivated by a common biological process where diffusing signal molecules (chemo) emitted by cells control the movement (taxis) of those cells. Cells are able to detect the concentration of signal and to move in directions of increasing or decreasing concentrations (positive and negative chemotaxis, respectively).

The chemotaxis design pattern was formulated as a composite pattern consisting of two components. The first component employs the plain diffusion design pattern to propagate signal system-wide. The second component utilizes the propagating signal to achieve a global data movement objective more efficiently. Improved efficiency is possible when signal carries information about the presence of data at remote locations; this information enables better local movement decisions to be made by the nodes that implement the second component. Chemotaxis assumes that the two components operate at different time scales, i.e., that signal propagates faster than the speed at which data can be moved.

In this section we apply the chemotaxis design pattern to the problem of load balancing. Our idea is to use a diffusing signal to guide the diffusion of load more efficiently towards the balanced distribution. We define signal as a load indicator that requires only a few bytes and therefore can propagate quickly. The load to be evenly distributed among nodes is assumed to consist of large amounts of data and therefore moves slowly in comparison.

Note that plain diffusion can also be applied to implement load balancing, for example, following the averaging algorithm presented in Section 5, provided the movement of load is cheap. Our goal here is to demonstrate the efficacy of chemotaxis as a mechanism for efficient load balancing by comparing it to load balancing via plain diffusion. This comparison will illustrate clearly the gains that may be achieved by using signal to guide the diffusive movement of data. We note that using diffusion to accomplish load balancing dates back at least to the work of Cybenko [1989] and Boillat [1990]. A recent survey of diffusive load balancing may be found in Elsässer and Monien [2003].

8.1 Chemotaxis

Let us elaborate on the properties of the chemotaxis design pattern and introduce the formalism we will use to describe the algorithm.

Plain diffusion is a simple concept. Basically, nodes that have more load than capacity send a fraction of their excess load to their neighbors. In the simplest case, a node i with load ϕ_i and capacity C_i will send a small fraction c of its excess load $(\phi_i - C_i)$ to each of its neighbors independent of node, of neighbor, and of time. Each transfer of load to a neighbor node j can be captured by the following equation:

$$\Delta\phi_{i\to j} = c(\phi_i - C_i). \tag{6}$$

With plain diffusion, load is moved in all directions without taking into account load already present in different regions of the network. Thus there is the risk of moving too much load to overloaded regions and too little to underloaded regions. The result is an inefficient load balancing mechanism.

It is well known from biology that certain cells are able to move autonomously. Such motile cells make decisions about when to move and in what direction to move based on the presence of certain chemicals in the immediate environment. The process of cell motility in response to concentration gradients of chemicals is called chemotaxis. Some chemicals (e.g., nutrients) may cause a cell to move in the direction of increasing concentration of the sensed chemical, other chemicals (e.g., poison) act as repellents and cause a negative chemotactic response.

The phenomenon of chemotaxis has inspired us to investigate a new load balancing mechanism for systems that restrict load to move slowly. The mechanism is based on a simulated negative chemotactic response; we use the term *signal* to denote the sensed repellent chemical. Chemotaxis allows us to make the movement of load less blind, by giving the load a local signal which can guide it away from overloaded regions of the network.

Chemotactic load balancing is based on the idea that each node continuously emits a signal proportional to its excess load. The signal emitted at node i at each time is:

$$\Delta S_i^{emit} = c_2(\phi_i - C_i). \tag{7}$$

Signal propagates through the network away from the emitting node using a fast diffusive mechanism. Signal, in contrast to load, is simply a single numerical value which can be encoded as a few bytes. We therefore assume that the restrictions on load movement speed do not apply to signal. At each time, signal accumulated at a node is diffused to its neighbors. The following equation expresses the simple diffusion of signal from a node i to its neighbor j:

$$\Delta S_{i\to j} = c_4 S_i. \tag{8}$$

Now, the slowly diffusing load can be guided by gradients of signal as follows:

$$\Delta \phi_{i\to j} = c_3(S_i - S_j). \tag{9}$$

Note that our new signal-aided diffusion mechanism consists of two components: a load diffusion component and a signal diffusion component. Also note that the two components are independent in the sense that they operate on different time scales.

8.2 Algorithms

To implement the simple equations for plain and signal-aided diffusion on a given overlay network topology we must develop corresponding algorithms. Our equation for plain diffusion, Eq. (6), exhibits two questionable features: negative load is sent whenever a node's load is less than capacity, and a node's load may become negative. Each of these features is either unrealistic or meaningless; hence we introduce simple modifications to Eq. (6) to address this. To remove the possibility of sending negative load we find the net difference $(\phi_i - C_i) - (\phi_j - C_j)$ for each node-neighbor link ij. Then the node with the largest (most positive) difference between load and capacity is chosen as the sending node and only the net, positive quantity of $c|(\phi_i - C_i) - (\phi_j - C_j)|$ is sent. To prevent a node's load from becoming negative we must ensure that no node sends more load than it has. If a node i has k_i neighbors then the total load sent in one time step can be as much as $ck_i(\phi_i - C_i)$. Hence, if c is chosen to be less than $1/k_i$ for all nodes i, then loads will always remain positive. (A similar constraint was imposed by Cybenko [1989].)

As pointed out in the previous section, signal and signal diffusion are not restricted in the same ways as load. Specifically, we assume that signal can move quickly, and that signal can take on negative values. After considerable experimentation, exploring a range of algorithms, we came up with two candidate algorithms for fast signal diffusion. The two algorithms are, for historical reasons, termed "version-6" and "version-10".

The version-6 algorithm is based on the algorithm for plain diffusion presented above. Though employed by us as a signal diffusion algorithm, the plain diffusion pattern is also suitable for diffusing load in systems that do not have restrictions on how quickly load can move (similarly to the averaging protocol in Section 5). The diffusion constant c is assigned a "default" value c_{default}. Any node i which discovers that $c_{\text{default}} > 1/k_i$ (where k_i is its degree), will adjust its own c value to be precisely $1/k_i$ in order to avoid negative load values. Hence, two neighbors i and j who have both adjusted their values by this rule will have diffusion constants $1/k_i$ and $1/k_j$, respectively. This gives an asymmetry which can seemingly violate load conservation. The problem is solved by picking a sending node, and only transferring the net positive difference — as in the algorithm for plain diffusion. The point is that the sending node can impose its choice of c on both ends of a given link, thus making the version-6 algorithm load conserving. An interesting feature of the version-6 algorithm, that we exploit in our experiments below, is that its speed can be continuously tuned: maximum speed is obtained by setting $c_{\text{default}} = 1$, while decreasing values reduce speed correspondingly.

The version-10 algorithm is only suitable for signal diffusion, because it does not maintain a strictly positive "load" (in our case, signal). There are similarities between the version-6 and version-10 algorithms, but contrary to version-6, version-10 has each sending node i always choosing $1/k_i$ as its diffusion constant. Also, the definition of sending node is modified, to allow for the fact that the sent quantity (signal) can be negative.

We know that signal can take both positive and negative values. (In fact, to avoid steady divergence of signal values towards plus or minus infinity, we set the total load equal to total capacity — thus enforcing an average value of zero for signal. Biological systems have *sinks* for signals; but our model does not.) Hence the definition of "sending node" requires generalization from the one-component case with purely positive load. We choose the sending node of two nodes i and j to be that node which has its signal value farthest from zero, i.e., farthest from the signal value corresponding to the uniform fixed point distribution. A sending node i sends its neighbor node j the amount of signal equal to $(S_i - S_j)/k_i$. Experiments have shown that the version-10 algorithm is significantly faster than the version-6 algorithm.

Early experiments with signal-aided diffusion showed that when diffusion of load responds to signal gradients according to Eq. (9), instabilities often resulted. A closer inspection of those early results, in addition to insights offered by the work of Cybenko [1989], led us to believe that chemotaxis can be made less prone to instabilities if nodes that contain less load are more constrained in their response to signal gradients. Our algorithm for Eq. (9) therefore incorporates the following two constraints. First, only nodes with more load than capacity are allowed to send to neighbors. Secondly, the total load sent must be less or equal to the difference between load and capacity of the sending node. The effect of these constraints is that once a node receives more load that capacity, it will maintain a load of at least capacity.

8.3 Simulation Model

We have conducted tests of our chemotaxis-inspired load balancing method using the PeerSim simulator [PeerSim]. This section briefly describes the simulation model used while the next section reports on our results.

Both plain diffusion and chemotactic diffusion converge only asymptotically to a uniform distribution. We therefore need to propose a definition of convergence: if $(max - min) < threshold$, where min is the smallest and max is the largest load values in the network, for a sufficient number n of consecutive simulation cycles then we have convergence. We use the values $n = 100$ and $threshold = 0.1$ unless otherwise specified.

The main focus of our simulation experiments is to compare plain diffusion with signal-aided diffusion (i.e., chemotaxis). It is therefore important to ensure that comparisons between the two be fair. It can be proven [Deutsch et al. 2003] that fairness says that the diffusion coefficient for load must be the same for both plain and signal-aided diffusion, i.e., we choose the same value for c and c_3. Hence, c and c_3 must be network-wide constants. We also impose the constraint, to ensure stable convergence of plain diffusion, that c must be smaller than the inverse degree of the most connected node in the topology.

Important simulation model parameters are the choice of overlay network topology and start distribution for load on nodes. We will focus here on what we feel is the most realistic choice for each: a power-law network topology and a random start distribution. Our power-law topology consists of 10.000 nodes, with the most connected node having 2200 neighbors. To generate a random start distribution, we divide the total load into 10.000 units, and place one unit at a time on a randomly selected node until all units have been placed.

8.4 Results

In this section we report on results from simulations using the simulation model defined in the previous section. We start by presenting our results regarding time to reach convergence, then we look at the quantities of load that were moved between nodes.

Our convergence time experiments aim at both comparing plain diffusion with signal-aided diffusion, and exploring the effect of different signal speeds on signal-aided diffusion performance. Recall that the Version-6 signal diffusion algorithm allows its speed to be altered by varying the value for the diffusion constant c_{default}. The fastest signal diffusion speed is always obtained by our Version-10 algorithm. Version-6 with $c_{\text{default}} = 1$ gives the second fastest signal speed. Progressively slower signal speeds are then obtained by halving the value of c_{default}. We chose Version-6 with $c_{\text{default}} = 1/2048$ as our slowest signal diffusion algorithm.

Figure 9 plots the time to reach convergence for plain diffusion runs and signal-aided diffusion runs with different signal speeds. Each of the three graphs in Figure 9 represents a different instance of the random start distribution. Signal speed increases along the horizontal axis. Convergence times for plain diffusion are shown as the left-most plot of each graph.

As can be seen from Figure 9, convergence to a balanced load was achieved for all runs, even when signal diffused very quickly compared to the load. It is also

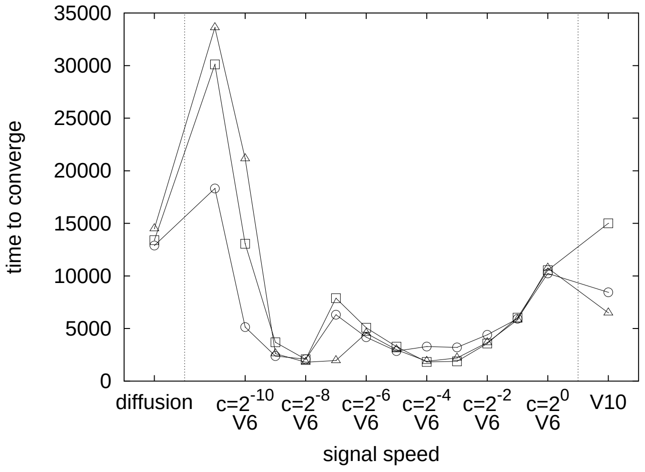


Fig. 9. The effect of increasing signal diffusion speed on time to reach convergence for signal-aided load diffusion. Each of the three graphs corresponds to a different instance of our random start distribution. The convergence times for plain diffusion are plotted to the far left. All other plots show convergence times for signal-aided diffusion.

evident that signal-aided diffusion exhibited shorter time to reach convergence than did plain diffusion for most simulation runs. Several signal speeds produced reductions in convergence times of about 80%[1]. Interestingly, the shortest convergence times were obtained when signal diffused at medium speeds. Signal-aided diffusion performed worse than plain diffusion when using our slowest signal speed (version-6 with $c_{\text{default}} = 1/2048$).

When signal diffused using the very fast version-10 algorithm and the version-6 algorithms with $c_{\text{default}} < 1/64$, an increased sensitivity to variations in start distribution was observed. Space does not permit us to include figures that show very similar behavior (i.e., increased sensitivity to variation at the very fast and slow ends of the signal speed spectrum) when introducing variations in topology, or when varying the start distribution more drastically than as in Figure 9. Numerous simulation runs that use a variety of topologies, start distributions, and convergence criterion thresholds have so far shown remarkable consistency in the time to reach convergence for signal-aided diffusion, when using medium (version-6) signal speeds.

Chemotaxis-inspired load balancing addresses systems where the ability to move load is limited. Therefore, a key metric is the amount of load that is moved during a short interval, e.g., during a single cycle. The largest load amount moved during a single cycle by plain diffusion for the runs whose convergence times are shown in Figure 9 was 0.002. For signal-aided diffusion, results varied widely with regards to largest load amount moved during a single cycle. The smallest values (around 0.014) were achieved when using the version-6 algorithm with $c_{\text{default}} > 1/8$ for signal diffusion. Version-6 results for slower signal speed were as much as seven

[1]The shortest time to reach convergence for signal-aided diffusion was 1795 cycles; plain diffusion took about 13.000 cycles to reach convergence.

times higher. Finally, when the version-10 algorithm was used to diffuse signal, the maximum load amount moved in a single cycle was about fifty times higher than for the best version-6 cases. Thus, although our results for maximum load moved are not as clear as for convergence time, our tests suggest that performance gains are possible while still restricting load movement speed.

8.5 Discussion

Our results show clearly that diffusion guided by chemotactic signal can give large improvements in speed of convergence over plain diffusion. A basic premise of our study is that the load itself is constrained to be slow. Otherwise, one might simply try using one-component fast diffusion for load, with the version-6 diffusion algorithm. Our results thus suggest that, in cases where load itself is slow (e.g., very large files) — while at the same time a centralized mechanism is undesirable — guided diffusion of load, via the chemotaxis mechanism, is a good candidate for efficient load balancing.

Figure 9 also gives a hint of the *sensitivity* of the three studied algorithms (plain diffusion, and chemotaxis with version-6 and version-10) to start distribution (and to signal speed for version-6). We see that version-10, and version-6 with very low speed, are rather sensitive to start distribution, while both diffusion and the best range of version-6 are not. Other studies show that plain diffusion can be highly sensitive to start distribution when the latter is more highly skewed, while version-6 retains its insensitivity. The same conclusion holds for variations in network topology. Hence we find version-6 chemotaxis to be outstandingly *insensitive* to variation of a range of environmental parameters. This kind of insensitivity (or adaptivity) is a property that we hope to find in decentralized, distributed mechanisms.

We describe here only a summary of an ongoing effort to thoroughly evaluate this new mechanism. We report elsewhere [Canright et al. 2005] on this work in full. In particular, we wish to evaluate more thoroughly the properties of adaptivity and robustness, both for the three algorithms described here, and for other, existing methods. We hope to show that our biology-inspired approaches can be both efficient and resilient, to a degree that competes successfully with any other approach.

Finally, we comment on the mathematical structure and detailed behavior of our two-component system. Clearly, and by choice, we have coupled a slow system to a fast system; hence we have created the discrete analog of "stiff" differential equations. This has two, not unexpected, consequences. The first is that the coupled systems show often a highly complex behavior over time, on the way to convergence. That is, they converge (fast), but the approach to convergence can be far from smooth! Secondly, the convergence rate is not optimal for fastest signal speed (where the disparity in speed between the two components is greatest) — see again Figure 9. Instead we find fastest convergence for moderate signal speed. Hence we need a *decentralized* mechanism for tuning the fast component speed to this moderate value. This may be done by, for example: (i) using our distributed aggregation protocol to find the maximum node degree $k_{\max}$ (see Section 5); (ii) setting the load speed coefficient c_3 (globally) to be somewhat less than $1/k_{\max}$; (iii) choosing (again globally) the signal speed c_4 according to $c_4 = R'c_3$, where R' is a global parameter chosen to give good performance. Out present results indicate that an R' in the range of 10–500 should work well.

9. RELATED WORK

Since work related to the individual applications presented in the paper was discussed in the corresponding sections, here we focus on the role of biological inspiration in computer science in general.

Seeking inspiration from nature for solving problems from computer science has a long history. There are several established and active research communities organized around powerful metaphors. Examples include evolutionary computing [Eiben and Smith 2003], ant colony optimization [Dorigo and Stützle 2004], artificial life [Langton 1997], artificial immune systems [de Castro and Timmis 2002], artificial neural networks [Haykin 1998], cellular automata [Ilachinski 2001], DNA computing [Păun et al. 2005] and membrane computing [Păun 2002]. These communities are focused on both understanding the corresponding metaphor, motivated by their "nice properties", and on applications of the metaphor to solve computational problems.

On the other hand, the distributed systems engineering community has turned to biological analogies recently from the opposite direction, starting from problems and identifying a range of natural systems as possible sources for solutions. IBM's autonomic computing initiative [Kephart and Chess 2003] has found many followers. The basic idea is to use the autonomic nervous system as a metaphor. Most importantly, the autonomic nervous system regulates many functions of the body without conscious intervention, hiding the details from the "user", that is, our conscious self. Furthermore, the idea of building computer systems using ideas from emergence and self-organization is also gaining momentum [Di Marzo Serugendo et al. 2004]. Other lines of research similar to our approach include amorphous computing [Abelson et al. 2000] and several ideas based on various complex adaptive systems [Staab et al. 2003].

10. DISCUSSION AND CONCLUSIONS

In this paper we have proposed a family of design patterns that facilitate the adoption of biology-inspired ideas in distributed systems engineering. The patterns capture primitive communication strategies of biological systems by expressing ideas in terms of a restricted *communication topology* of a large set of simple components, along with the definition of the *local* communication scheme on top of the topology, the function of the components, examples from biology and the expected global outcome (or function). The patterns allow the translation of ideas from a large number of seemingly different biological systems to the same language, and they allow for the specialization and customization of these ideas so that they can be applied to distributed systems. The design patterns can be considered as a *middle layer* of abstraction between biological systems and computer systems. Ideas expressed in this layer are more abstract than actual biological systems. They typically generalize some common "ideas" of a diverse set of biological and sometimes even social systems. These abstract ideas can in turn be specialized again for application in distributed systems, typically combined with other design patterns.

We have described a number of design patterns such as diffusion, replication, chemotaxis, stigmergy and reaction-diffusion. We have described in detail four applications that are based on these design patterns: aggregation (based on diffusion), load balancing (based on chemotaxis), search (based on replication) and routing (based on stigmergy, diffusion and replication). We have developed several other applications as well. For lack of space, we mention them only briefly:

Power optimization in MANETs.. We have proposed distributed protocols for the problem of assigning transmission powers to the nodes of a wireless network in such a way that all the nodes are connected through bidirectional links, and the total power consumption is minimized. This problem is important since nodes are usually equipped with batteries with a very limited lifetime. The new distributed protocols [Montemanni and Gambardella 2005b; pear] implement state-of-the-art centralized techniques for power minimization [Montemanni and Gambardella 2005a; Montemanni et al. pear] in a local, distributed fashion. Use of these distributed protocols lead to a system where optimization of the global network emerges from the behavior of local nodes, each carrying out a myopic, local optimization and exchanging information with other nodes through a reaction-diffusion mechanism.

Unstructured overlay topology management.. We have proposed protocols that can construct and maintain a random network in extreme environments with catastrophic failures and extremely high rates of churn [Jelasity et al. 2004]. These random networks can be used as a basis for many other protocols, in particular, protocols that need to communicate regularly with random peers. Examples include protocols for aggregation, load balancing and search presented in this paper. The underlying pattern here is replication.

Structured overlay topology management.. Most peer-to-peer applications require some special overlay topology, such as semantic or geographic proximity, or sorting according to some property of the nodes or according to abstract keys. The T-MAN protocol [Jelasity and Babaoglu 2005] offers a solution to this problem, based on the *cell adhesion* pattern from developmental biology. The basic idea of adhesion is that cells preferentially select some other cells to be their neighbors, based on some markers (molecules). Combined with a stochastic "cooling" process, the cell adhesion model can explain pattern formation [Glazier and Graner 1993].

The goal of identifying and applying design patterns from biology is the possibility that we can match the scalability, robustness and adaptivity of biological systems in technological networks. Having evaluated carefully the performance of our proposed applications, we can conclude that they indeed inherit some of these "nice properties" present in the underlying abstract ideas they are based on.

However, a large number of open questions remain. In particular, during the specialization process, when we apply a design pattern in a specific network topology, and possibly under specific constraints, we need to better understand how the performance of the given idea depends on these environment variables. The identification of simple patterns makes it possible to analyze these ideas at a high-enough level of abstraction, opening up many promising new research directions.

ACKNOWLEDGMENT

This work was partially supported by the Future and Emerging Technologies unit of the European Commission through Project BISON (IST-2001-38923). We would like to thank Poul Heegaard, Gian Paolo Jesi, Vittorio Maniezzo, Luciano Margara, Kenth Engø-Monsen and Andrea Rizzoli for their valuable contribution to the ideas presented in the paper.

REFERENCES

Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Thomas F. Knight, J., Nagpal, R., Rauch, E., Sussman, G. J., and Weiss, R. 2000. Amorphous computing. *Communications of the ACM 43,* 5 (May).

Adamatzky, A., De Lacy Costello, B., and Asai, T. 2005. *Reaction-diffusion Computers.* Elsevier.

Albert, R. and Barabási, A.-L. 2002. Statistical mechanics of complex networks. *Reviews of Modern Physics 74,* 1 (Jan.), 47–97.

Alexander, C. 1977. *A Pattern Language: Towns, Buildings, Construction.* Center for Environmental Structure Series. Oxford University Press.

Arbib, M. A., Érdi, P., and Szentágothai, J. 1997. *Neural Organization: Structure, Function and Dynamics.* MIT Press.

Bailey, N. T. J. 1975. *The mathematical theory of infectious diseases and its applications*, second ed. Griffin, London.

Baras, J. S. and Mehta, H. 2003. A probabilistic emergent routing algorithm for mobile ad hoc networks. In *WiOpt03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks.*

Bertsekas, D. and Gallager, R. 1992. *Data Networks.* Prentice–Hall, Englewood Cliffs, NJ, USA.

Bettstetter, C., Resta, G., and Santi, P. 2003. The node distribution of the random waypoint mobility model for wireless ad hoc networks. *IEEE Transactions on Mobile Computing 2,* 3, 257–269.

Boillat, J. 1990. Load balancing and poisson equation on a graph. *Concurrency: Practice and Experience 2*, 280–313.

Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraulaz, G., and Bonabeau, E. 2001. *Self-Organization in Biological Systems.* Princeton University Press.

Canright, G., Deutsch, A., and Urnes, T. 2005. Chemotaxis-inspired load balancing. In *Proceedings of the European Conference on Complex Systems* (ECCS 05).

Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., and Shenker, S. 2003. Making gnutella-like p2p systems scalable. In *Proceedings of ACM SIGCOMM 2003.* ACM Press, 407–418.

Clausen, T., Jacquet, P., Laouiti, A., Muhlethaler, P., Qayyum, A., and Viennot, L. 2001. Optimized link state routing protocol. In *Proceedings of IEEE INMIC.*

Cybenko, G. 1989. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing 7*, 279–301.

de Castro, L. N. and Timmis, J. 2002. *Artificial Immune Systems.* Springer.

Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. 1987. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87).* ACM Press, Vancouver, British Columbia, Canada, 1–12.

Deutsch, A., Ganguly, N., Canright, G., Jelasity, M., and Engø-Monsen, K. 2003. Models for advanced services in AHN, P2P Networks. Bison Deliverable, www.cs.unibo.it/bison/deliverables/D08.pdf.

Di Caro, G. 2004. Ant colony optimization and its application to adaptive routing in telecommunication networks. Ph.D. thesis, Faculté des Sciences Appliquées, Université Libre de Bruxelles, Brussels, Belgium.

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

Di Caro, G., Ducatelle, F., and Gambardella, L. 2004. AntHocNet: an ant-based hybrid routing algorithm for mobile ad hoc networks. In *Proceedings of Parallel Problem Solving from Nature (PPSN) VIII.* Lecture Notes in Computer Science, vol. 3242. Springer-Verlag, 461–470. (Conference best paper award).

Di Caro, G., Ducatelle, F., and Gambardella, L. 2005a. AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Transactions on Telecommunications,* Special Issue on Self Organization in Mobile Networking *16,* 5 (September–October), 443–455.

Di Caro, G., Ducatelle, F., and Gambardella, L. 2005b. Swarm intelligence for routing in mobile ad hoc networks. In *Proceedings of the 2005 IEEE Swarm Intelligence Symposium (SIS).*

Di Marzo Serugendo, G., Karageorgos, A., Rana, O. F., and Zambonelli, F., Eds. 2004. *Engineering Self-Organising Systems.* Lecture Notes in Artificial Intelligence, vol. 2977. Springer.

Dorigo, M., Di Caro, G., and Gambardella, L. M. 1999. Ant algorithms for distributed discrete optimization. *Artificial Life 5,* 2, 137–172.

Dorigo, M. and Stützle, T. 2004. *Ant Colony Optimization.* MIT Press, Cambridge, MA.

Ducatelle, F., Di Caro, G., and Gambardella, L. 2005a. Ant agents for hybrid multipath routing in mobile ad hoc networks. In *Proceedings of the Second Annual Conference on Wireless On demand Network Systems and Services (WONS).* St. Moritz, Switzerland.

Ducatelle, F., Di Caro, G., and Gambardella, L. 2005b. Using ant agents to combine reactive and proactive strategies for routing in mobile ad hoc networks. *International Journal of Computational Intelligence and Applications (IJCIA),* Special Issue on Nature-Inspired Approaches to Networks and Telecommunications *5,* 2 (June), 169–184.

Eiben, A. E. and Smith, J. E. 2003. *Introduction to Evolutionary Computing.* Springer.

Elsässer, R. and Monien, B. 2003. Diffusion load balancing in static and dynamic networks. In *Proc. Internat. Workshop on Ambient Intelligence Computing.* 49–62.

Fewell, J. H. 2003. Social insect networks. *Science 301,* 26 (September), 1867–1869.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1995. *Design Patterns.* Addison-Wesley.

Ganguly, N., Brusch, L., and Deutsch, A. 2005. Design and analysis of a bio-inspired search algorithm for peer to peer networks. In *Self-Star Properties in Complex Information Systems.* Lecture Notes in Computer Science, Hot Topics, vol. 3460. Springer-Verlag.

Ganguly, N., Canright, G., and Deutsch, A. 2004a. Design of a Robust Search Algorithm for P2P Networks. In 11th *International Conference on High Performance Computing.*

Ganguly, N., Canright, G., and Deutsch, A. 2004b. Design Of An Efficient Search Algorithm For P2P Networks Using Concepts From Natural Immune Systems. In 8th *International Conference on Parallel Problem Solving from Nature.*

Ganguly, N. and Deutsch, A. 2004a. A Cellular Automata Model for Immune Based Search Algorithm. In 6th *International conference on Cellular Automata for Research and Industry.*

Ganguly, N. and Deutsch, A. 2004b. Developing Efficient Search Algorithms for P2P Networks Using Proliferation and Mutation. In 3rd *International Conference on Artificial Immune Systems.*

Glazier, J. A. and Graner, F. 1993. Simulation of the differential adhesion driven rearrangement of biological cells. *Phys. Rev. E 47,* 3, 2128–2154.

Günes, M., Kähmer, M., and Bouazizi, I. 2003. Ant-routing-algorithm (ARA) for mobile multi-hop ad-hoc networks - new features and results. In *Proceedings of the 2nd Mediterranean Workshop on Ad-Hoc Networks (Med-Hoc-Net'03).* Mahdia, Tunisia.

H. Van Dyke Parunak, S. B. 2004. Stigmergic learning for self-organizing mobile ad-hoc networks. In *Proceedings of AAMAS.*

Haas, Z. J. 1997. A new routing protocol for the reconfigurable wireless networks. In *Proceedings of the IEEE International Conference on Universal Personal Communications.*

Haykin, S. 1998. *Neural Networks: A Comprehensive Foundation*, 2nd ed. Prentice Hall.

ILACHINSKI, A. 2001. *Cellular Automata: A Discrete Universe.* World Scientific.

JANEWAY, C. A., TRAVERS, P., WALPORT, M., AND SHLOMCHIK, M. 2001. *Immuno Biology: The Immune System in Health and Disease*, 5th ed. Garland Publisher.

JELASITY, M. AND BABAOGLU, O. 2005. T-Man: Gossip-based Overlay Topology Management. In *3rd Int. Workshop on Engineering Self-Organising Applications (ESOA'05).*

JELASITY, M., GUERRAOUI, R., KERMARREC, A.-M., AND VAN STEEN, M. 2004. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Middleware 2004*, H.-A. Jacobsen, Ed. Lecture Notes in Computer Science, vol. 3231. Springer-Verlag, 79–98.

JELASITY, M., MONTRESOR, A., AND BABAOGLU, O. 2004. A modular paradigm for building self-organizing peer-to-peer applications. In *Engineering Self-Organising Systems*, G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, Eds. Lecture Notes in Artificial Intelligence, vol. 2977. Springer, 265–282.

JELASITY, M., MONTRESOR, A., AND BABAOGLU, O. 2005. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems 23,* 3 (Aug.), 219–252.

JOHNSON, D. AND MALTZ, D. 1996. *Mobile Computing.* Kluwer, Chapter Dynamic Source Routing in Ad Hoc Wireless Networks, 153–181.

KEIL, D. AND GOLDIN, D. 2005. Adaptation and evolution in dynamic persistent environments. In *Proceedings of the Workshop on the Foundations of Interactive Computation (FInCo2005).* To be published in Electronic Notes in Theoretical Computer Science.

KEMPE, D., DOBRA, A., AND GEHRKE, J. 2003. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03).* IEEE Computer Society, 482–491.

KEPHART, J. O. AND CHESS, D. M. 2003. The vision of autonomic computing. *IEEE Computer 36,* 1 (Jan.), 41–50.

LANGTON, C. G., Ed. 1997. *Artificial Life: An Overview.* MIT Press.

LEE, D. L., CHUANG, H., AND SEAMONS, K. 1997. Document ranking and the vector-space model. *IEEE Softw. 14,* 2, 67–75.

LODDING, K. N. 2004. The hitchhiker's guide to biomorphic software. *ACM Queue 2,* 4, 66–75.

LV, Q., CAO, P., COHEN, E., AND SHENKER, S. 2002. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of the* 16^{th} *ACM International Conference on Supercomputing.*

MANKIN, R., ARBOGAST, R., KENDRA, P., AND WEAVER, D. 1999. Active spaces of pheromone traps for *Plodia interpunctella* in enclosed environments. *Environmental Entomology 28,* 4, 557–565.

MONTEMANNI, R. AND GAMBARDELLA, L. 2005a. Exact algorithms for the minimum power symmetric connectivity problem in wireless networks. *Computers and Operations Research 32,* 11 (November), 2891–2904.

MONTEMANNI, R. AND GAMBARDELLA, L. 2005b. Power-aware distributed protocol for a connectivity problem in wireless sensor networks. In *Self-Star Properties in Complex Information Systems.* Lecture Notes in Computer Science, Hot Topics, vol. 3460. Springer-Verlag.

MONTEMANNI, R. AND GAMBARDELLA, L. to appear. Swarm approach for a connectivity problem in wireless networkws. In *Proceedings of the IEEE Swarm Intelligence Symphosium (SIS 2005).*

MONTEMANNI, R., GAMBARDELLA, L., AND DAS, A. to appear. Models and algorithms for the MPSCP: an overview. In *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks* (J. Wu ed.). CRC Press.

MURRAY, J. D. 1990. *Mathematical Biology.* Springer-Verlag.

OTTINO, J. M. 2004. Engineering complex systems. *Nature 427*, 399.

PARUNAK, H. V. D., BRUECKNER, S. A., SAUTER, J. A., AND MATTHEWS, R. 2005. Global convergence of local agent behaviors. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 05).* 305–312.

PEERSIM. http://peersim.sourceforge.net/.

Perkins, C. and Royer, E. 1999. Ad-hoc on-demand distance vector routing. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications.*

Păun, G. 2002. *Computing with Membranes: an Introduction.* Springer.

Păun, G., Rozenberg, G., and Salomaa, A. 2005. *DNA Computing.* Springer.

Risson, J. and Moors, T. 2004. Survey of Research towards Robust Peer-to-Peer Networks: Search Methods. Tech. Rep. UNSW-EE-P2P-1-1, University of New South Wales, Sydney, Australia. Sept.

Royer, E. and Toh, C.-K. 1999. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*.

Scalable Network Technologies, Inc. 2003. *QualNet Simulator, Version 3.6.* Scalable Network Technologies, Inc., Culver City, CA, USA. `http://www.scalable-networks.com`.

Schmidt, D. C., Johnson, R. E., and Fayad, M. 1996. Guest editorial for the special issue on patterns and pattern languages. *Communications of the ACM 39,* 10 (Oct.).

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

Shen, C.-C., Jaikaeo, C., Srisathapornphat, C., Huang, Z., and Rajagopalan, S. 2004. Ad hoc networking with swarm intelligence. In *Ants Algorithms - Proceedings of ANTS 2004, Fourth International Workshop on Ant Algorithms.* LNCS. Springer-Verlag.

Staab, S., Heylighen, F., Gershenson, C., Flake, G. W., Pennock, D. M., Fain, D. C., De Roure, D., Aberer, K., Shen, W.-M., Dousse, O., and Thiran, P. 2003. Neurons, viscose fluids, freshwater polyp hydra—and self-organizing information systems. *IEEE Intelligent Systems 18,* 4, 72–86.

Sutton, R. and Barto, A. 1998. *Reinforcement Learning: An Introduction.* MIT Press.

Theraulaz, G. and Bonabeau, E. 1999. A brief history of stigmergy. *Artificial Life,* Special Issue on Stigmergy *5*, 97–116.

van Renesse, R. 2003. The importance of aggregation. In *Future Directions in Distributed Computing*, A. Schiper, A. A. Shvartsman, H. Weatherspoon, and B. Y. Zhao, Eds. Number 2584 in Lecture Notes in Computer Science. Springer, 87–92.

van Renesse, R., Birman, K. P., and Vogels, W. 2003. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems 21,* 2 (May), 164–206.

Yuste, S. B. and Acedo, L. 2000. Number of distinct sites visited by N random walkers on a Euclidean lattice. *Physical Review E 61*, 6327–34.

Zipf, G. K. 1935. *Psycho-Biology of Languages.* Houghton-Mifflin.

Received Month 2005; revised Month 2006; accepted Month 2006