# Design Patterns from Biology for Distributed Computing*

Ozalp Babaoglu[1], Geoffrey Canright[2], Andreas Deutsch[3], Gianni Di Caro[4], Frederick Ducatelle[4], Luca Gambardella[4], Niloy Ganguly[3], Márk Jelasity[1**], Roberto Montemanni[4], and Alberto Montresor[1]

[1] University of Bologna, Mura Anteo Zamboni 7, I-40126 Bologna, Italy
babaoglu,jelasity,montresor@cs.unibo.it
[2] Telenor R&D, Snarøyveien 30, N-1331 Fornebu, Norway
geoffrey.canright@telenor.com
[3] Center for High Performance Computing, Dresden University of Technology, Dresden, Germany.
deutsch,niloy@zhr.tu-dresden.de
[4] Istituto "Dalle Molle" di Studi sull'Intelligenza Artificiale (IDSIA)
gianni,frederick,luca,roberto@idsia.ch

**Abstract**

Recent developments in information technology have brought about important changes in distributed computing. New environments have emerged such as massively large-scale wide area computer networks and mobile ad hoc networks. These new environments are extremely *dynamic*, *unreliable* and often *large-scale*. Traditional approaches to designing distributed applications based on central control, small scale or strict reliability assumptions are not suitable for exploiting the enormous potential of these environments. Based on the observation that living organisms efficiently organize a large number of unreliable and dynamically changing components (cells, molecules, individuals of a population, etc) it has long been an interesting area of research to try to figure out what are the key ideas that make biological systems work and to apply these ideas in distributed systems engineering. In this paper we propose a conceptual framework that captures a few basic biological processes such as plain diffusion, reaction-diffusion, proliferation, etc. We show through examples how to implement practically relevant functions based on these ideas. Using a common evaluation methodology, we show that these applications have state-of-the-art effectivity and performance while they inherit some nice properties of biological systems, such as adaptivity and robustness to failure.

## 1 Introduction

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

However, these new environments are extremely *dynamic*, *unreliable* and often *large-scale*. Traditional approaches to distributed system design that assume that the system is composed of reliable components, or that the system is of relatively small scale are not applicable in such 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 also 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 a possible way of coping with this problem. It is well-known that living organisms efficiently organize a large number of unreliable and dynamically changing components (cells, molecules, individuals of a population,

*Authors are listed in alphabetical order.
**also with RGAI, MTA SZTE, Szeged, Hungary

etc) into structures that implement a wide diversity of functions. Besides, most of the biological structures (organisms) have a number of "nice properties" which include robustness to the failure 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 a large number of different fields of computer science. Furthermore, having been a niche topic for a long time, biological inspiration is recently beginning to make its way into the mainstream of distributed computing [44, 52].

In this paper we propose a conceptual framework that is based on *design patterns*, not unlike those of object oriented design [25]. We identify design patterns common to a number of biological systems, including plain diffusion, reaction-diffusion, proliferation, stigmergy, etc. The patterns represent a middle layer 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 based on these ideas. Using a common evaluation methodology, we show that these applications have state-of-the-art effectivity and performance while they inherit some nice properties of biological systems, such as adaptivity and robustness to failure.

The outline of the paper is as follows. In Section 2 we describe the conceptual framework and the design patterns used in the rest of the paper. Sections 3 to 5 describe three examples of distributed services in this framework: data aggregation and search in overlay networks and routing in ad hoc networks. Section 6 discusses related work and Section 7 concludes the paper.

# 2 Conceptual Framework

The dynamic distributed environments mentioned in the Introduction, in particular, overlay networks and ad-hoc networks, share a number of important characteristics. This section is devoted to identifying these characteristics and modeling these environments in a common framework that will allow us to apply the same abstract ideas interchangeably in any environment that matches these specifications. After introducing a system model that covers both environments, we introduce a set of primitive building blocks inspired by biological systems. These building blocks operate over the abstraction layer represented by the system model. The applications described in this paper are constructed around one or more of these building blocks.

## 2.1 System Model

The basic components of our system model are *nodes*. The nodes are typically computing devices that can maintain some state, or perform computations. A node has a set of *neighbors*. A node is able to send *messages* to its neighbors only. That is, the set of neighbors of node $i$ is the subset of nodes which $i$ can send messages to. We will often call the set of neighbors the *view* of the node. The message passing mechanism is asynchronous, that is, delivery is not guaranteed in a fixed time window. Nodes can fail, and can leave or join the system any time. Messages can be lost. The size of the view, that is, the number of neighbors is typically much less than the number of nodes in the system.

In this framework, we can identify the *topology* of the network as a crucial aspect. The topology is given by the graph defined by the neighborhood 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 gets partitioned as a result of the failure of one or more nodes can be expressed in graph theoretical terms. Recent advances in the field of complex networks further

underline the importance of network topology [2]. Accordingly, throughout the paper we shall pay special attention to topology, both in terms of design and evaluation.

To summarize, the framework involves three concepts: nodes, neighbors and messages, with the properties defined above. When instantiating this framework, we need to map these concepts onto the components of the system in question.

## 2.2 Example Networks

Having introduced the basic abstract concepts, let us explain how this system model can be applied to overlay networks and mobile ad-hoc networks, the environments discussed in this paper.

### 2.2.1 Overlay Networks

In overlay networks, the nodes are connected by a physical network and we also assume the existence of a routing service. In other words, any node can send a message to any other node, provided that the target network address is known. That is, a node can actually send a message only if it knows the address of the target node. This means that the view of a node does not and cannot contain the entire network. The reason is that—under the assumption of a dynamic environment and a large number of nodes—it is not feasible to keep track of the entire network in real time. In overlay networks there are typically millions of nodes, and keeping an up-to-date list of the nodes that are online is not realistic. Therefore the topology will not be the complete graph, but a graph in which nodes have a limited degree, that is, they have a limited number of neighbors.

Another characteristic of overlay networks is that the topology can be chosen almost arbitrarily. The underlying routing service ensures that in principle any pairs of nodes can be connected, so there is a large degree of freedom for defining the actual topology. 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) [57] 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 neighborhood relations in the system model depend on the wireless connections between nodes. The set of nodes a given node can access is therefore defined by its transmission range and the physical proximity between the nodes. As opposed to 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. Most notably, like in overlay networks, the topology is also restricted. One one hand, this is due to the limited power of the nodes, which means they are typically not able to cover the entire span of the network. Besides, apart from the power constraints, the problem of interference also restricts the transmission range. Nodes can transmit only when the frequency is free. If the range is too large, there are too many overlapping transmissions which render the network unusable. The difference from overlay networks is that topology is given by the physical location of the nodes. By changing the transmission power (and therefore the range) of the nodes, it is possible to tune the topology, but in a much more limited sense.

## 2.3 Primitive Functions: Design Patterns

In the following we describe a number of biological processes, cast in the system model defined above. They all share the assumptions of the model: they are defined over systems that are large scale, dynamic, unreliable, and can be described as a message passing process of a set of nodes organized in a *restricted* communication topology. We will demonstrate in the remaining part of the paper

that these processes can provide us with a useful set of building blocks to solve practically relevant problems.

### 2.3.1 Plain Diffusion

Diffusion is a simple yet ubiquitous process that can be observed in a wide range of biological and physical systems [51]. Diffusion involves equalizing the concentration of some material or some abstract quantity, like heat or electric potential. Instead of a rigorous mathematical approach, here we define diffusion to include processes that rely on the passive equalization of concentrations.

| | |
|---|---|
| node | Nodes are idealized parts of space. The function these "nodes" perform is that they receive the diffusive material from neighboring nodes, and (according to a diffusion coefficient) emit a part of their material to their neighbors. Plain diffusion is weight conserving, which means material is not created nor does it disappear. Nodes have *state* in that they remember the amount of material they hold, modeled by a non-negative real number. |
| neighbor | The neighborhood relation is defined by the topology of the space in which diffusion takes place. In biological systems it is often modeled by a 2- or 3-dimensional regular grid. |
| message | The message is the actual material that is sent to the neighbor. It is typically modeled by a non-negative real number. |

### 2.3.2 Replication

Replication-based processes are commonplace in nature. Examples include growth processes, signal propagation is certain neural networks [3], epidemic spreading [4], or proliferation processes in the immune system [34]. From now on, we will use the epidemics metaphor: this is the closest analogy with the applications we discuss.

| | |
|---|---|
| node | In the epidemics metaphor, nodes are for example potential hosts of a virus. The nodes have a state, which we define as the virus itself, if the node is infected or "not-infected" otherwise. Other definitions are of course possible. |
| neighbor | The neighborhood relation can be defined by physical proximity, sexual contact or social relationships. |
| message | The message is the virus. Typically it is transmitted unchanged. It can also mutate in the host and get transmitted in its mutated form. |

### 2.3.3 Chemotaxis

When cells or other organisms direct their movements according to the concentration gradients of some chemicals in the environment, we talk about chemotaxis. Chemotaxis is not a strictly "primitive" function, as it is defined on top of a diffusion process of the chemical in question. 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 scale* of diffusion and chemotaxis is usually different.

| | |
|---|---|
| node | Nodes are idealized parts of space. Diffusion takes place as in plain diffusion, but apart from that, a node can host one or more cells as well. The node "sends" the cells in the direction of the gradient of the diffusive material (looking at concentrations at neighboring nodes). |
| neighbor | The neighborhood relation is defined by the topology of the space in which diffusion takes place. |
| message | The message can include the diffusive material, and one or more cells as well. |

### 2.3.4 Reaction-Diffusion

Reaction-diffusion is a generalization of diffusion, 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 [51].

| | |
|---|---|
| node | Nodes are idealized parts of space. The function they perform is that they receive the diffusive materials from neighboring nodes, and update the local concentration of each material, potentially generating new material or removing some of it. The system is therefore not weight conserving. Nodes have *state* in that they remember the amount of material they hold, modeled by a non-negative real number. |
| neighbor | Like with plain diffusion, the neighborhood relation is defined by the topology of the space in which diffusion takes place. |
| message | The message is the actual material that is sent to the neighbor. |

### 2.3.5 Stigmergy and Learning by Reinforcements

*Stigmergy* is a form of distributed control based on indirect communication among agents. Each agent reacts to the actual state of the environment, and locally modifies this state. If this interaction between the agents and the environment is properly "designed", it leads to global coordination of the agent actions [61]. The local variables of the environment whose values determine the response of the agent, are called *stigmergic variables*. More specifically, they represent the local parameters of the agent *decision policy*. The repeated updating of these parameters in the direction of locally *reinforcing* the decisions which led to globally good behaviors gives raise to a *distributed reinforcement learning process* (e.g., [60]). 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 [8, 24]. Learning by reinforcements is at very basis of the ontogenetic development of living beings (e.g., [11]).

| | |
|---|---|
| node | Nodes are idealized parts of space. A node has stigmergic variables (e.g., pheromone level). These variables are read and written by agents, and can also be modified as a function of time (e.g., evaporation of pheromone). |
| neighbor | The neighborhood relation between nodes is defined by the physical possibility of agents (e.g., insects) to move between the locations corresponding to the nodes. |
| message | The messages are the agents themselves which are passed between the nodes of the environment. The behavior of the agents is defined by local decisions based on the stigmergic variables, while these variables are in turn updated according to the received and passed agents. |

### 2.4 Evaluation Methodology

An important motivation for the study of bio-inspired methods is something that we call the "nice properties" of living systems. 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" or FOM. The measured value of a FOM is of course dependent on many things, which we loosely break down into two categories: the system (protocol, algorithm) being evaluated, and the "environment", which may be described quantitatively in terms of environmental variables EV. Obvious examples of the latter include the given topology, the load or stress, fluctuations, etc.

An insensitive system will then show little variation in the set of FOMs describing its performance, as the environment is varied. This may be expressed quantitatively as $((\Delta FOM)/(\Delta EV))^{-1}$. That is, $(\Delta FOM)/(\Delta EV)$ expresses *sensitivity*; and we take insensitivity to be its inverse. Thus our generic definition of insensitivity has the dimensions of an inverse derivative or finite difference.

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 EV 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 FOM is totally insensitive to system size (although in Section 3 we will see an example). Next we address the term *robustness*. We also view robustness as a type of insensitivity. Here the EV 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. We offer them here, 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 the finite difference/derivative. 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, reproduce, in the face of the fluctuating environment, while the former must maintain their own corresponding figures of merit.

## 3 Data Aggregation

Aggregation is an important service in many distributed systems [62]. It is a common name for a set of functions that provide a summary of some global system property. In other words, they allow local access to global information in order to simplify the task of controlling, monitoring and optimization in distributed applications. Possible aggregation functions include finding extremal values of some property, computing averages and sums, etc. For example, they can be used to obtain information such as 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 [38].

In this section we introduce a scalable, robust and adaptive protocol for calculating aggregates. The core of the protocol is a simple generic scheme [37,50] in which aggregation is performed through periodic local message exchanges between nodes. As we will see later, if the aggregate we would like to compute is the *average* then the protocol takes the form of a kind of *diffusion*. If the aggregate we are interested in is an extremal value (maximum, minimum) then the protocol takes the form of an

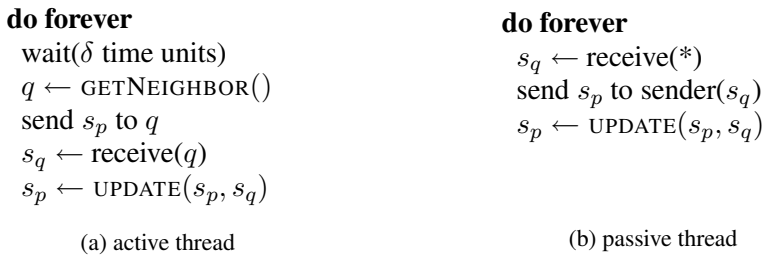



Figure 1: Protocol executed by node $p$.

*epidemic* algorithm. In sum, the primitive building blocks we will apply from Section 2.3 are diffusion and replication. We will mainly focus on average calculation in the present discussion.

There are a number of general purpose systems for aggregation, the best known of which is Astrolabe [63]. 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. [40] propose an aggregation protocol similar to ours, based on gossiping and tailored to work on random topologies. Apart from some minor technical details, that allow our system to work in networks characterized by weak connectivity, the main difference is the 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.

## 3.1 The Basic Protocol

Each node in the network holds a numeric value. In a practical setting, this value can characterize any (dynamic) aspect of the node or its environment (e.g., the load at the node, temperature monitored by a sensor network). The task of a proactive protocol is to continously provide all nodes with an up-to-date estimate of an aggregate function, computed over the values held by the nodes.

Our basic aggregation protocol is based on a push-pull gossiping scheme 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 $\delta$ called *cycles* that are enumerated starting from some convenient point.

Method UPDATE builds a new local state based on the previous one and the remote state received during the information exchange. The output of UPDATE depends on the specific function being implemented by the protocol. For example, to implement 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.

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

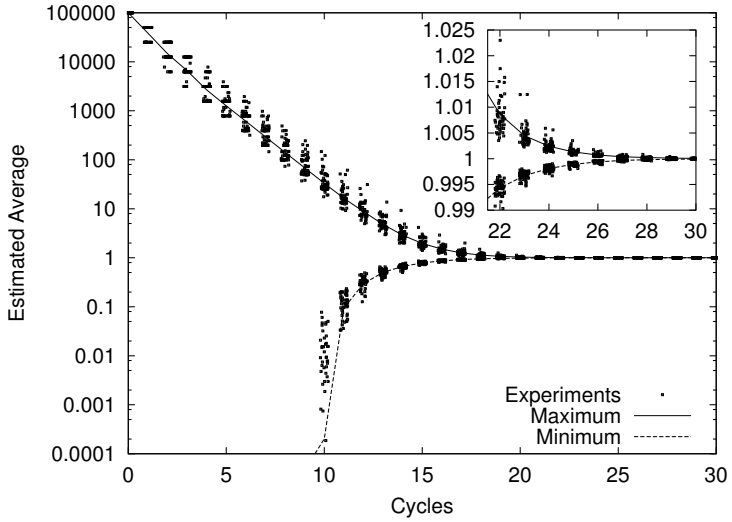



Figure 2: Behavior of protocol AVERAGE.

to all nodes in an epidemic fashion. Other aggregates are described in [37, 50]. From now on we restrict our discussion to 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 [37], we presented analytical results for the convergence speed of the averaging protocol. Let $\sigma_i^2$ be the empirical variance of the local estimates at cycle $i$. The *convergence factor* $\rho_i$, 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 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.

Figure 2 illustrates the behavior of the protocol. The AVERAGE protocol was run on a simulated network composed of $10^5$ nodes connected through a regular random overlay network, where each node knows exactly 20 neighbors. Initially, a single node has the value $10^5$, while all others have zero as their local value (so that the global average is 1). We are interested in this *peak distribution* because it constitutes a worst-case scenario for averaging, in particular for testing robustness. In the Figure, results for the first 30 cycles of the protocol are shown. The two curves represent the minimum and the maximum estimates of the average over all the nodes at the completion of each cycle. The curves correspond to averages over 50 independent experiments, whose results are shown as individual points in the figure.

## 3.2 Experimental Results

To complement the theoretical analysis, we have performed numerous experiments based on simulation. Our goal is to show the scalability and robustness of our approach. Instead of the simple AVERAGE protocol, we examine the COUNT protocol, that computes the number of nodes present in the system. The COUNT protocol is in fact 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

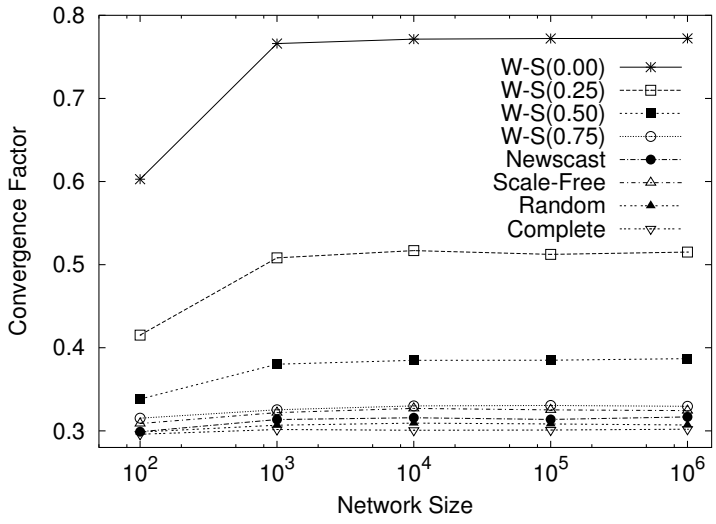



Figure 3: Average convergence factor computed over a period of 20 cycles in networks of varying size. Each curve corresponds to a different topology where W-S($\beta$) stands for the Watts-Strogatz model with parameter $\beta$.

have 0, then running AVERAGE we obtain $1/N$; the network size, $N$, can be easily deduced from it. COUNT is more 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.

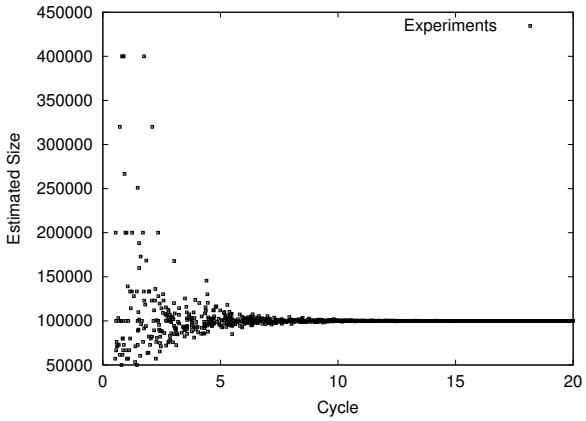
Unless stated otherwise, all simulations are performed on networks composed of $10^5$ nodes. The size estimates and the convergence factor plotted in the figures are those obtained after 30 cycles. The underlying overlay network used for communication is based on NEWSCAST, a gossip-based protocol for maintaining random connected topologies. 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.

### 3.2.1 Scalability and Insensitivity to Underlying Topology

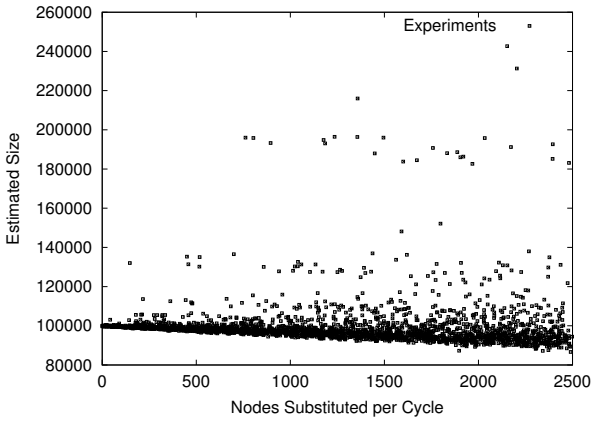
Regarding scalability, we have run COUNT in networks whose size range from $10^3$ to $10^6$ nodes. Several different underlying topologies have been considered, including the complete graph, random network, scale-free topology, newscast, and several small-world networks with varying $\beta$ parameter. The results are shown in Figure 3. They confirm in case of random topologies, the convergence factor is indeed independent of the network size, and approximates the $1/(2\sqrt{e})$ value predicted by the analysis. Also, as long as it is sufficiently random, the protocol is insensitive to the choice of underlying topology.

### 3.2.2 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




(a) Network size estimation with protocol COUNT where $50\%$ of the nodes crash suddenly. The x-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.

Figure 4: Effects of node crashes on the COUNT protocol in a NEWSCAST network.

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, relatively large values are present, obtained from the first exchanges originated by the initial value 1. These observations are confirmed by Figure 4(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.

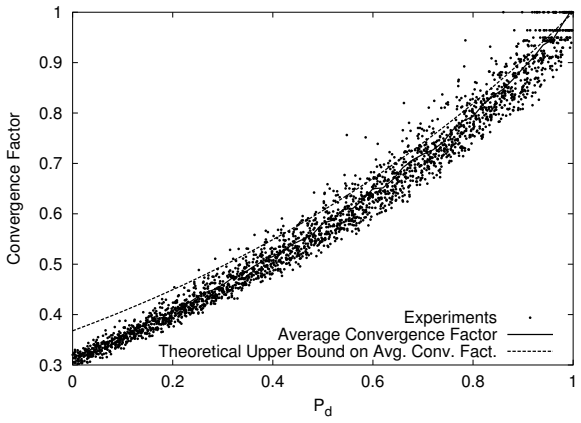
A more realistic scenario is a network subject to churn. Figure 4(b) illustrates the behavior of aggregation in such a network. Churn is modeled by removing a number of nodes from the network and substituting them with new nodes at each cycle. In other words, the size of the network is constant, while 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.

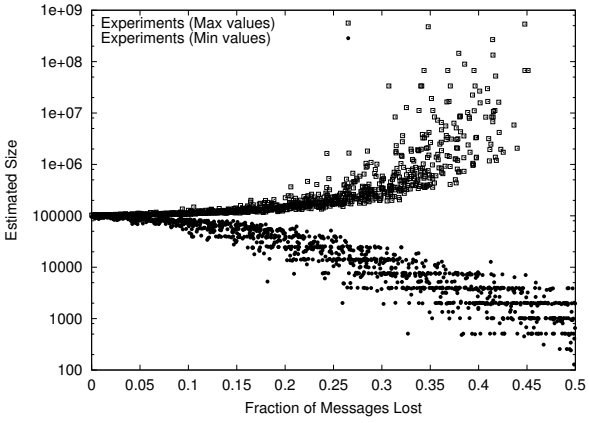
The figure demonstrates that (even when a large number of nodes are substituted during an execution) most of the estimates are included in a reasonable range. The above experiment can be considered as a worst case analysis since the level of churn was much higher than it could be expected in a realistic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

### 3.2.3 Robustness to Communication Failures

Figure 5(a) shows the convergence factor of COUNT in the presence of link failures. Link failure do not result in any loss of approximation quality or increased unreliability, since no value is lost when




(a) Convergence factor of protocol COUNT as a function of link failure probability.




(b) Network size estimation with protocol COUNT as a function of lost messages.

Figure 5: Effects of communication failures on the COUNT protocol in a NEWSCAST network.

a link is not functioning. On the other hand, an increasing percentage of message losses results in a proportionally slower convergence, as illustrate by the figure.

Apart from link failures that interrupt communication between two nodes in a symmetric way, it is also possible that single messages are lost. If the message sent to initiate an exchange is lost, the final effect is the same as with link failure: the entire exchange is lost, and the convergence process is just slowed down. But if the message lost is the response to an initiated exchange, the global average may change (either increasing or decreasing, depending on the value contained in the message).

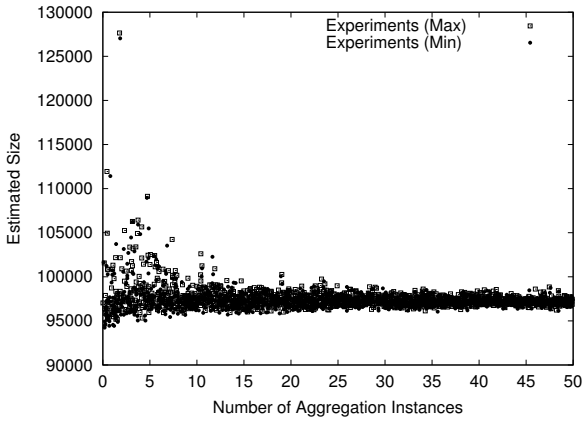
The effect of message omissions is illustrated in Figure 5(b). The given percentage of all messages (initiated or response) was dropped. For each experiment, both the maximum and the minimum estimates over the nodes in the network are shown, represented by the ends of the bars. As can be seen, when a small percentage of messages are lost, estimations of reasonable quality can be obtained. Unfortunately, when the number of messages lost is higher, the results provided by aggregation can be larger or smaller by several orders of magnitude.

### 3.2.4 Increasing Robustness

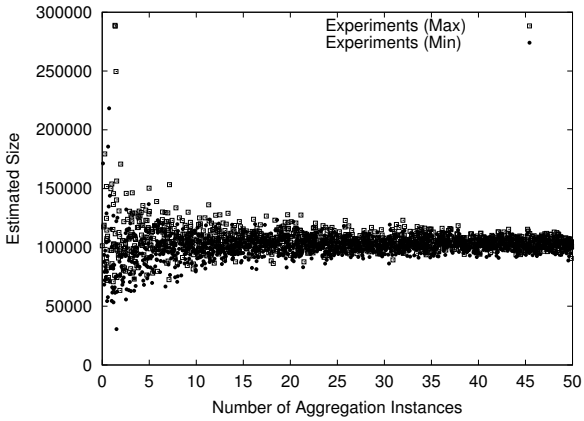
To reduce the impact of "unlucky" runs of the aggregation protocol that generate incorrect estimates due to failures, one possibility is to run multiple concurrent instances of the aggregation protocol. To test this solution, we have simulated a number $t$ of concurrent instances of the COUNT protocol, with $t$ varying from 1 to 50. At each node, the $t$ estimates that are obtained after 30 cycles are ordered. Subsequently, the $\lfloor t/3 \rfloor$ lowest estimates and the $\lfloor t/3 \rfloor$ highest estimates are discarded, and the reported estimate is given by the average of the remaining results.

Figure 6(a) shows the results obtained by applying this technique in a system where 1000 nodes per cycle are substituted with new nodes, while Figure 6(b) shows the results in a system where 20% of the messages are lost.

The results are quite encouraging; by maintaining and exchanging just 20 numerical values (resulting in messages of still only a few hundreds of bytes), the accuracy that may be obtained is very high, especially considering the hostility of the scenarios tested. It can also be observed that the estimate is very consistent over the nodes (the bars are short) in the crash scenario (as predicted by our theoretical results), and using multiple instances the variance of the estimate over the nodes decreases significantly even in the message omission scenario, so the estimate is sufficiently representative at




(a) Network size estimation with multiple instances of protocol COUNT. 1000 nodes crash at the beginning of each cycle.




(b) Network size estimation with protocol COUNT as a function of concurrent protocol instances. 20% of messages are lost.

Figure 6: Effects of failures on protocol COUNT with multiple concurrent instances.

every single node.

# 4 Search in Unstructured Overlay Networks

In this section we focus on *unstructured* overlay networks. In these networks the nodes are connected with each other regardless of their content, geographical location, etc, in a random fashion. Unstructured networks are attractive because of a number of reasons. They are extremely easy to maintain, they are highly robust to failure and the dynamic changes in the participating nodes (churn). Besides, search algorithms implemented over unstructured networks support arbitrary *keyword based* search [10].

Flooding techniques have generally been used to implement search in unstructured networks. Although, flooding fulfills the criterion of robustness and also information is found very fast, it produces a huge number of query messages which ultimately overwhelm the entire system—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 [45]. In this section, we report search algorithms based on *proliferation*, which produces a comparable number of messages to the $k$-random walker algorithm, however, 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 [34], which–in our terminology—represents a variation on the replication primitive. Proliferation helps in increasing the number of antibodies which can then efficiently track down the antigens (foreign bodies). In our problem, the query message is conceived as antibody which is generated by the node initiating a search whereas antigens are the searched items hosted by other nodes of the overlay network. Like in the natural immune system, the messages undergo proliferation based upon 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 [26–30].

## 4.1 System Model

We focus on the two most important aspects of a peer-to-peer system: network topology, and query and data distribution. For simplicity, we assume 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 the time of change in network topology and change in query distribution, results obtained from the fixed 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 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 by simple distributed protocols [36]. In the rest of this section we fix the network size to be $N = 10000$, and the average degree $Np = 4$.

**Data and query distribution** Files are modeled as collections of keywords [43]. Hence the data distribution is represented in terms of keywords. It is assumed that there are 2000 different keywords in the system. Each node hosts some keywords. The number of keywords (not necessarily unique) in each node follows a Poisson distribution with mean 1000. The data profile of a node is represented as $D = \{(\delta_1, n_1), (\delta_2, n_2), \cdots\}$ where $\delta_i$ is an individual unique keyword and $n_i$ indicates the number of times $\delta_i$ is contained in the node.

The query is a set of keywords $Q = \{q_1, q_2, \cdots\}$ As of generating queries: for 95% of the cases, the query length is less than 5, while it is between 6 and 10 in the rest of the cases. In the 95% of the cases where the query length is less than 5, each length 1 to 5 is equiprobable. This is also true within the remaining 5% of the cases. The 2000 keywords are distributed in a way such that they follow Zipf's distribution [65].

## 4.2 Algorithms

In this section, we introduce two proliferation based search algorithms. All nodes 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 $Q$ as follows. Let us assume that the receiving node has data profile $D$ and the total number of (not necessarily unique) keywords in $D$ is $K = \sum_i n_i$. The number of hits is given by

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

where $q_j \oplus \delta_i = 1$ if $q_j = \delta_i$, otherwise 0. The number of successful matches calculated this way is then recorded to calculate search statistics.

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 in the case of the different algorithm variants:

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

**Proliferation (P)** In the proliferation scheme, the queries possibly undergo proliferation at each node they visit, that is, they might be forwarded to several neighbors. The node first calculates the number of messages it needs to forward ($\eta_p$) using a proliferation controlling function to be

described below. The function determining the value of $\eta_p$ ensures that $\eta_p$ is less than the number of neighbors of the node.

All forwarding approaches have a corresponding *restricted* version. Restricted forwarding means that the copies of the query to be forwarded are sent to "free" neighbors only. By "free", we mean that the respective neighbors have not been previously visited 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).

Let us now define how the number of copies is calculated in the proliferation scheme. 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 $Sm/K$ where where the value of $Sm$ is calculated through (1) and $K$ is defined as in (1). Note that $0 \leq Sm/K \leq 1$. The number of copies to be forwarded is defined as

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

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

### 4.3 Experimental Results

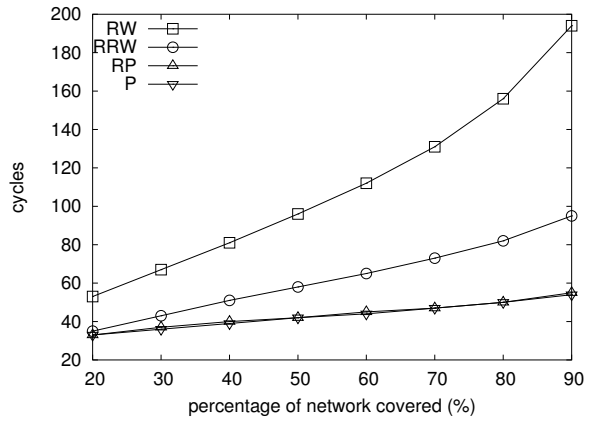
In this section we compare random walk and proliferation. The overlay network and the query and data distributions are described in Section 4.1. The experiments focus on efficiency aspects of the algorithms, and use the following simple metrics that reflect the fundamental properties of the algorithms:

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

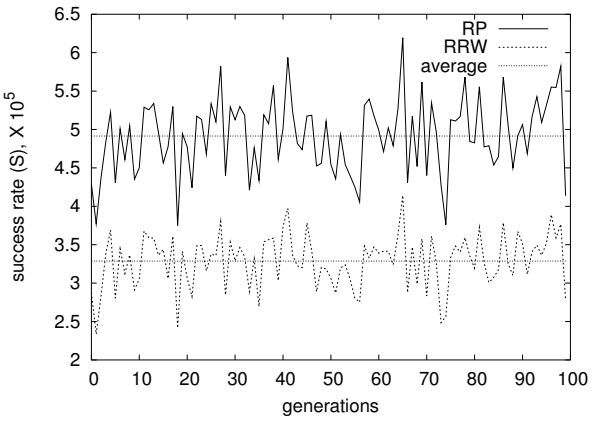
**Search Efficiency** The number of similar items found by the query messages within a given time period.

Both proliferation and random walk algorithms are distributed in nature 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. The order of execution of the nodes during one cycle is arbitrary. The length of the message queue is considered 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 the 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 clogging 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




(a) Network coverage of all the protocol variants.




(b) Search efficiency of RP and RRW

Figure 7: Experimental results on network coverage (a) and search efficiency (b).

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 based on the above considerations. To ensure fairness in "power" between the two proliferation algorithms P and RP, we keep the proliferation constant $\rho$ and the value of $k$ the same for both algorithms.

### 4.3.1 Network Coverage

Here we are interested in how efficiently do 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 7(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 7(a) it is seen that P and RP need an almost identical number of cycles to cover the network. It is, however, much less than that needed by RRW and RW. Algorithm RRW is much more efficient than RW. Simple proliferation (run with the same proliferation constant $\rho$ as RP), produces much more messages than RP (not shown). So, although P and RP produce similar results in terms of the number of cycles, we can conclude that the restricted version of both the random walk and proliferation algorithms is more efficient.

### 4.3.2 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 (set 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 consist of 100 searches. In each search, we collect all the hits in the system, summing up the number of hits ($Sm$) 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 7(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 $5 \cdot 10^5$, while it is $3.2 \cdot 10^5$ for RRW.)

### 4.4 Discussion

In this section, we have presented experimental results showing that the simple immune-system inspired concept of 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 [26] 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 define an efficient flooding mechanism, a mechanism which will not generate a huge number of messages like traditional flooding but will have comparable speed. 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 [64]. Our goal is to design proliferation schemes which will take only $O(t^2)$ time, yet will use a much lower number of message packets than flooding.

## 5 Routing in Mobile Ad Hoc Networks

Routing is the task of directing data flows from sources to destinations 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 in the same time they should be efficient with respect to bandwidth use. Such properties can be expected to result from the implementation of the biological functions described earlier. In particular, we use *stigmergy* and *learning through reinforcements*, taking inspiration from the foraging behavior of ants which allows the colony to find the shortest path between the nest and a food source [8]. 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* [19]. In particular, it was the inspiration for a number of adaptive routing algorithms for communications networks, such as *AntNet* [14] (see [13] for an overview). In what follows, we describe a new MANET routing algorithm, called *AntHocNet* [16, 22]., which implements *stigmergy* and *learning through reinforcements* modeled after the ant behavior, as well as a *diffusion function*.

## 5.1 Description of the algorithm

MANET routing algorithms are usually classified according to whether they are proactive or reactive [57]. Purely *proactive* algorithms, such as DSDV [53], try to maintain routes between all pairs of nodes at all times. Purely *reactive* algorithms, such as AODV [54] only gather routing information when a data session to a new destination is started, or when a route which is in use fails. Reactive algorithms are in general more efficient and scalable [7] since they reduce routing overhead, but they are less adaptive since they only obtain information when it is strictly necessary. In practice, many algorithms are *hybrid* (e.g. ZRP [31]), using both proactive and reactive components. *AntHocNet is a hybrid algorithm*: it 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 as long as communication is going on. 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: nodes send out ant-like agents which *sample* and *reinforce* good paths to their assigned destination. Routing information is kept in arrays of stigmergic variables, called *pheromone tables*, which are followed and updated by the ant agents. This mechanism is further supported by a *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 each of the algorithm's components. A detailed description and evaluation of AntHocNet can be found in [16, 17, 22].

### 5.1.1 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 and number of hops. These values play the role of stigmergic variables in the distributed reinforcement learning process: they are followed by ant agents which sample paths to a given destination, and are in turn updated by ants according to the estimated goodness of the sampled paths (see 5.1.2). The routing tables are therefore termed *pheromone tables*. Since AntHocNet only maintains information about destinations which are active in a communication session, and the neighbors of a node change continually, the filling of the pheromone tables is sparse and dynamic. The learned pheromone tables are used to route data packets in a stochastic forwarding process (see 5.1.4).

### 5.1.2 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*. 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 chooses its 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 $\mathcal{N}_d^i$ is the set of neighbors of $i$ over which a path to $d$ is known, and $\beta$ 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 keeps 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*, which travels back to the source retracing $\mathcal{P}$. At each intermediate node $i \in \mathcal{P}$ ($i < d$), the backward ant reads a locally maintained estimate $\hat{T}_{i+1}^i$ of the time it takes to reach the neighbor $i + 1$ the ant is coming from. The time $\hat{T}_d^i$ it would take a data packet to reach $d$ from $i$ over $\mathcal{P}$ is calculated incrementally as the sum of the local estimates $\hat{T}_{j+1}^j$ gathered by the ant between $i$ and $d$. A pheromone value is a goodness measure, expressed as an inverted cost, that takes into account both end-to-end delay and number of hops. It has the dimension of an inverted time. Therefore, to calculate the pheromone update value $\tau_d^i$, we combine the estimated delay $\hat{T}_d^i$ with the number of hops to the destination $h$ as follows:

$$\tau_d^i = \left( \frac{\hat{T}_d^i + hT_{hop}}{2} \right)^{-1}, \tag{4}$$

where $T_{hop}$ is a fixed value representing the time to take one hop in unloaded conditions. Defining $\tau_d^i$ 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. The value of $\mathcal{T}_{nd}^i$ is updated as follows:

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

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.

### 5.1.3 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. 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 $\mathcal{T}_{m^*d}^n$, $m^* \in \mathcal{N}_d^n$, 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 [5].

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.

### 5.1.4 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 $\beta$ 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.

### 5.1.5 Link failures

Nodes can detect link failures (e.g., a neighbor has moved away) when unicast transmissions fail, or when expected periodic pheromone diffusion messages were not received. When a neighbor is assumed to have disappeared, the node takes a number of actions. In the first place, it removes the neighbor from its view and all associated entries from its pheromone table. Next, the node starts a diffusion process to notify other nodes of the changed situation. The diffused message contains a list of the destinations to which the node lost its best path, and the new best pheromone to this destination (if it still has entries for the destination). All its neighbors receive the message and update their pheromone table using the new estimates. If they in turn lost their best only path to a destination, they will pass the updated message on to their neighbors, until all concerned nodes are notified of the new situation.

If the link failure was detected via the failed transmission of a data packet, and the node has no

further paths available for the destination of this packet, the node starts a *local route repair*. The node broadcasts a *route repair ant* that travels to the involved destination like a reactive forward ant: it follows available routing information when it can, and is broadcast otherwise. One important difference is that it has a restricted number of broadcasts so that its proliferation is limited. If the local repair fails, the node starts a new diffusion process to notify other nodes.

### 5.2 Experimental results

AntHocNet's performance was evaluated in an extensive set of simulation tests using QualNet [58], a 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 [15–17, 21, 22].

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 $3000 \times 1000$ $m^2$. Each experiment is run for 900 seconds. Data traffic is generated by 20 constant bit rate (CBR) sources sending one 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 pathloss model is used in the radio propagation model. The radio range of the nodes is 300 meters, 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 [39]: 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 20 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)* [54], a de facto standard algorithm and commonly used in comparative studies.

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 [6]). The results of these tests are presented in Figures 8 (average delay and delivery ratio) and 9 (average jitter and overhead). AntHocNet shows much better effectiveness than AODV, in terms of average delay, delivery ratio and jitter. AODV has better efficiency, measured as routing overhead, but the difference is rather small. The bad performance for high pause times are due to the reduced connectivity. 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 10 and 11. We can see that AntHocNet's advantage for all measures of effectiveness grows 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 AODV for small networks, it increases less fast and is much lower than AODV's for the larger networks.

## 6 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.

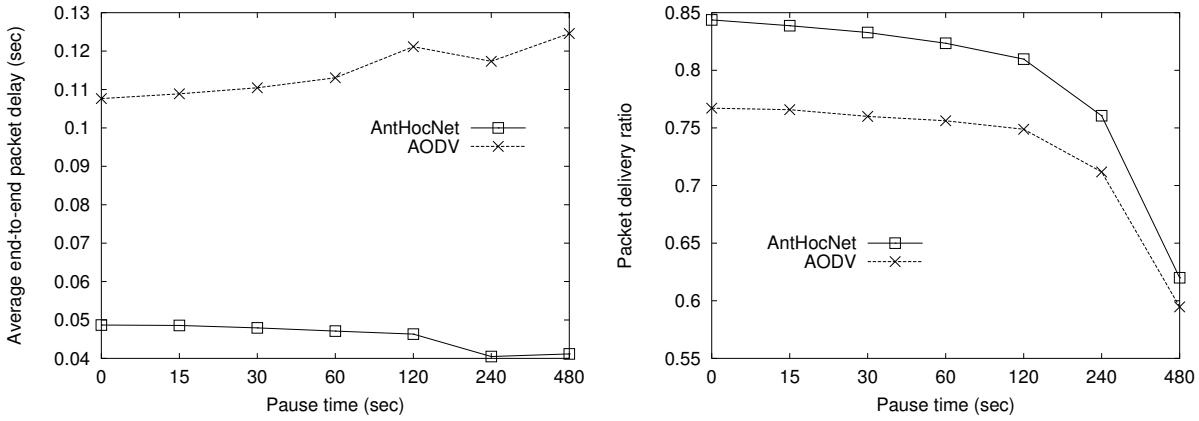



Figure 8: Average delay (left) and delivery ratio (right) for increasing pause times

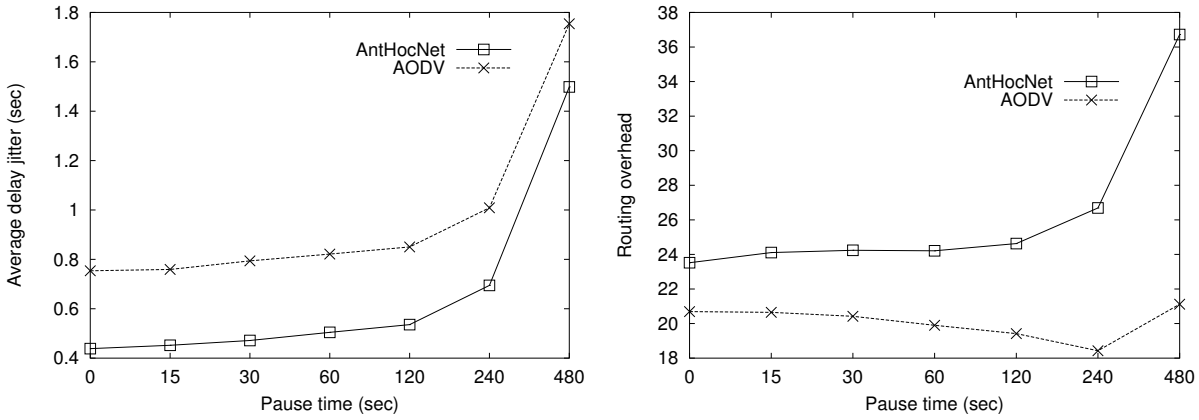



Figure 9: Average jitter (left) and routing overhead (right) for increasing pause times

Examples include evolutionary computing [23], ant colony optimization [20], artificial life [42], artificial immune systems [12], artificial neural networks [32], cellular automata [33], DNA computing [56] and membrane computing [55]. 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 as well, from the opposite direction, starting from problems and identifying a range of natural systems as possible sources for solutions. IBM's autonomic computing initiative [41] 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. Besides, the idea of building computer systems using ideas from emergence and self-organization is also gaining momentum [18]. Other lines of research similar to our approach include amorphous computing [1] and several ideas based on various complex adaptive systems [59].

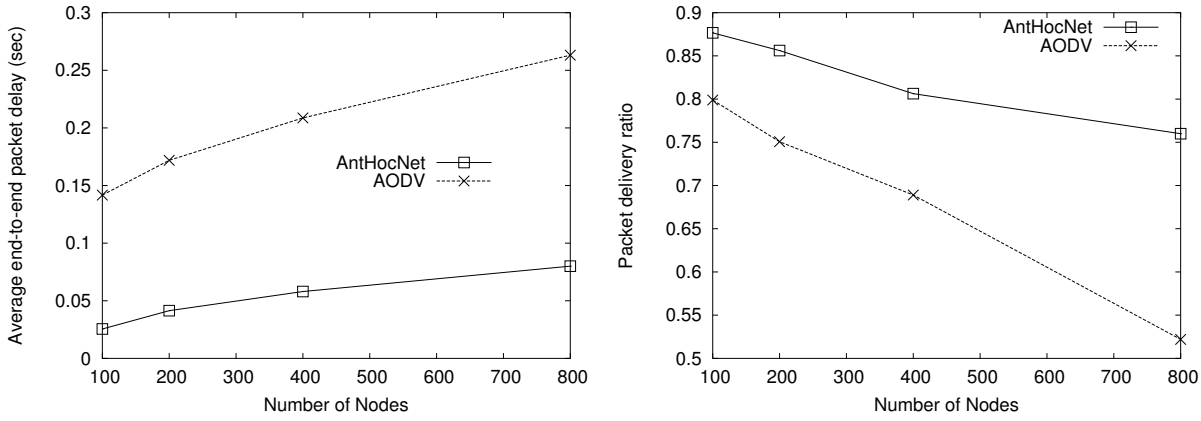



Figure 10: Average delay (left) and delivery ratio (right) for an increasing number of nodes

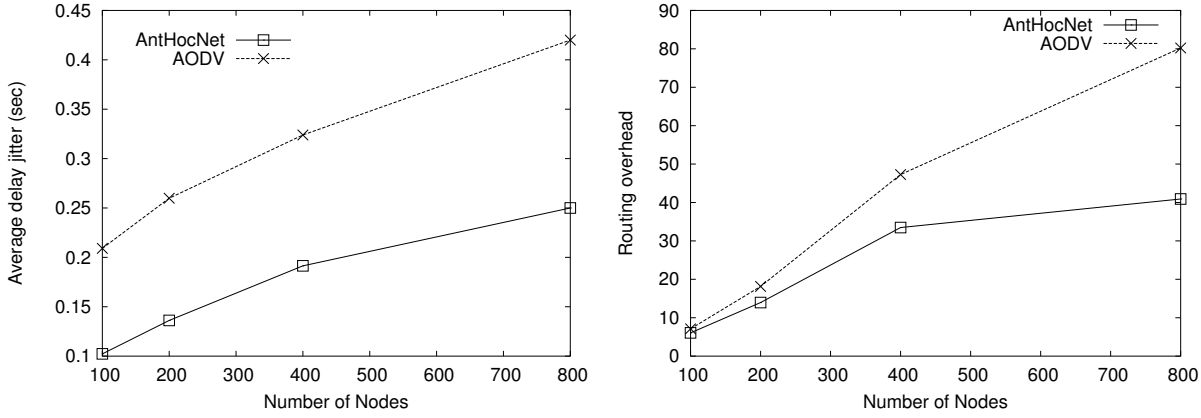



Figure 11: Average jitter (left) and routing overhead (right) for an increasing number of nodes

# 7 Discussion and Conclusions

In this paper we proposed a simple conceptual framework that facilitates the adoption of biology-inspired ideas in distributed systems engineering. The framework is aimed at capturing 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).

These strategies are not unlike *design patterns* familiar from software engineering [25]. They 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 for application in distributed systems. The 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 "idea" 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 three applications that are based on these

design patters: aggregation (based on diffusion), search (based on replication) and routing (based on stigmergy). These applications have been presented in detail, however, a few more applications have also been developed. We describe them here briefly:

**load balancing** We have proposed a load balancing scheme based on the idea of chemotaxis, where load balancing is carried out by a relatively slow diffusion process guided by a relatively fast "signal", that is emitted by the load. That is, high load emits a signal that defines a gradient over the communication topology along which load can be intelligently transferred towards regions with low load. We discuss this protocol in detail in [9] where we show that the chemotactic signaling mechanism indeed results in a significant improvement in load balancing performance w.r.t. plain diffusion (a well-known approach to load balancing), and that the protocol is insensitive to a wide range of environmental parameters.

**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 by 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 [47, 48] implement state-of-the-art centralized techniques for power minimization [46, 49] in a local, distributed fashion. The use of these distributed protocols lead to a system where the optimization of the global network emerges as a result of the behavior of local nodes, each one carrying out a myopic, local optimization and exchanging information with the other nodes through a reaction/diffusion mechanism.

**unstructured overlay topology management** We have proposed protocols that can maintain a random network in the face of extreme circumstances such as catastrophic failures, and extremely high churn [36]. This topology can be used as a basis for many other protocols, in particular, all protocols that need to communicate regularly with random peers from the network such as the protocols for aggregation, load balancing and search presented in this paper. The underlying idea here is replication.

**structured overlay topology management** Most peer-to-peer applications require some special overly topology such as semantic or geographical proximity, or sorting according to some property of the nodes or according to abstract keys. The T-MAN protocol [35] offers a solution to this problem, based on the idea of *adhesion*. The basic idea of adhesion is that cells preferentially select some other cells to be their neighbors, based on some markers. Combined with a stochastic "cooling" process, the cell adhesion model can explain pattern formation.

The goal of the identification and application of design patterns from biology is the promise that we can replicate the scalability, robustness and adaptivity of biological systems. Having carefully evaluated the performance of the proposed applications, we can conclude that they indeed inherit some of these "nice properties" of the underlying abstract ideas they are based on.

However, a large number of open questions remain. In particular, along the specialization process, when we apply a pattern in a specific network topology possibly under some specific constraints, we need to understand well how 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.

## Acknowledgments

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, Andrea Rizzoli and Tore Urnes for their valuable contribution to the ideas presented in the paper.

# References

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

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

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

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

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

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

[7] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom98)*, 1998.

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

[9] G. Canright, A. Deutsch, and T. Urnes. Chemotaxis-inspired load balancing, 2005. submitted for publication.

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

[11] P. Dayan. Motivated reinforcement learning. In *Advances in Neural Information Processing Systems 14 (NIPS14)*, pages 11–18. MIT Press, 2001.

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

[13] G. Di Caro. *Ant Colony Optimization and its application to adaptive routing in telecommunication networks*. PhD thesis, Faculté des Sciences Appliquées, Université Libre de Bruxelles, Brussels, Belgium, 2004.

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

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

[16] G. Di Caro, F. Ducatelle, and L. Gambardella. AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Transactions on Telecommunications*, 15(4), 2005. To appear.

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

[18] G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, editors. *Engineering Self-Organising Systems*, volume 2977 of *Lecture Notes in Artificial Intelligence*. Springer, 2004.

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

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

[21] F. Ducatelle, G. Di Caro, and L. Gambardella. 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, January 18–19, 2005.

[22] F. Ducatelle, G. Di Caro, and L. Gambardella. Using ant agents to combine reactive and proactive strategies for routing in mobile ad hoc networks. *International Journal of Computational Intelligence and Applications (IJCIA)*, 2005. To appear.

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

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

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

[26] N. Ganguly, L. Brusch, and A. Deutsch. Design and analysis of a bio-inspired search algorithm for peer to peer networks. In *Post Proceeding, Self-Star Conference*, June 2004.

[27] N. Ganguly, G. Canright, and A. Deutsch. Design of a Robust Search Algorithm for P2P Networks. In $11^{th}$ *International Conference on High Performance Computing*, December 2004.

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

[29] N. Ganguly and A. Deutsch. A Cellular Automata Model for Immune Based Search Algorithm. In $6^{th}$ *International conference on Cellular Automata for Research and Industry*, October 2004.

[30] N. Ganguly and A. Deutsch. Developing Efficient Search Algorithms for P2P Networks Using Proliferation and Mutation. In $3^{rd}$ *International Conference on Artificial Immune Systems*, September 2004.

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

[32] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd edition, 1998.

[33] A. Ilachinski. *Cellular Automata: A Discrete Universe*. World Scientific, 2001.

[34] C. A. Janeway, P. Travers, M. Walport, and M. Shlomchik. *Immuno Biology: The Immune System in Health and Disease*. Garland Publisher, 5th edition, 2001.

[35] M. Jelasity and O. Babaoglu. T-Man: Gossip-based Overlay Topology Management. In *3rd Int. Workshop on Engineering Self-Organising Applications (ESOA'05)*, 2005.

[36] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In H.-A. Jacobsen, editor, *Middleware 2004*, volume 3231 of *Lecture Notes in Computer Science*, pages 79–98. Springer-Verlag, 2004.

[37] M. Jelasity and A. Montresor. Epidemic-Style Proactive Aggregation in Large Overlay Networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 102–109, Tokyo, Japan, Mar. 2004. IEEE Computer Society.

[38] M. Jelasity, A. Montresor, and O. Babaoglu. A modular paradigm for building self-organizing peer-to-peer applications. In G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, editors, *Engineering Self-Organising Systems*, volume 2977 of *Lecture Notes in Artificial Intelligence*, pages 265–282. Springer, 2004.

[39] D. Johnson and D. Maltz. *Mobile Computing*, chapter Dynamic Source Routing in Ad Hoc Wireless Networks, pages 153–181. Kluwer, 1996.

[40] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, pages 482–491. IEEE Computer Society, 2003.

[41] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, Jan. 2003.

[42] C. G. Langton, editor. *Artificial Life: An Overview*. MIT Press, 1997.

[43] D. L. Lee, H. Chuang, and K. Seamons. Document ranking and the vector-space model. *IEEE Softw.*, 14(2):67–75, 1997.

[44] K. N. Lodding. The hitchhiker's guide to biomorphic software. *ACM Queue*, 2(4):66–75, 2004.

[45] Q. Lv, P. Cao, E. Cohen, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of the* $16^{th}$ *ACM International Conference on Supercomputing*, June 2002.

[46] R. Montemanni and L. Gambardella. Exact algorithms for the minimum power symmetric connectivity problem in wireless networks. *Computers and Operations Research*, 32(11):2891–2904, November 2005.

[47] R. Montemanni and L. Gambardella. Power-aware distributed protocol for a connectivity problem in wireless sensor networks. In *Post-Proceedings of Self-Star Properties in Complex Information Systems. Lecture Notes in Computer Science 3460*, to appear.

[48] R. Montemanni and L. Gambardella. Swarm approach for a connectivity problem in wireless networkws. In *Proceedings of the IEEE Swarm Intelligence Symphosium (SIS 2005)*, to appear.

[49] R. Montemanni, L. Gambardella, and A. Das. 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, to appear.

[50] A. Montresor, M. Jelasity, and O. Babaoglu. Robust Aggregation Protocols for Large-Scale Overlay Networks. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, pages 19–28, Florence, Italy, June 2004. IEEE Computer Society.

[51] J. D. Murray. *Mathematical Biology*. Springer-Verlag, 1990.

[52] J. M. Ottino. Engineering complex systems. *Nature*, 427:399, Jan. 2004.

[53] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

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

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

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

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

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

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

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

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

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

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

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

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