Anthill: a Framework for the Design and the Analysis of Peer-to-Peer Systems

Alberto Montresor

Department of Computer Science, University of Bologna
via Mura Anteo Zamboni 7, I-40127 Bologna (Italy)
Email: montresor@cs.unibo.it

Abstract

The peer-to-peer (P2P) paradigm for building distributed applications has recently gained attention from both industry and the media, following the enormous success of P2P systems like Napster, Gnutella and Freenet. The multitude of P2P projects appeared recently must confront common problems including security, reliability and routing. Unfortunately, traditional techniques for dealing with these issues are not completely adequate, given the substantial increase in the scale of these systems. In this paper, we introduce a new approach for designing new peer-to-peer applications, based on the biological model of ant colony networks. Furthermore, we present Anthill, a framework supporting developers and researchers in the design and the analysis of new protocols based on this approach.

1 Introduction

The *peer-to-peer* (P2P) paradigm for building distributed applications has recently gained attention from both industry and the media. Simply stated, in a P2P network each machine is a *peer* that can cooperate with every other machine in the network, both providing and consuming services at the same time. Clearly, this paradigm differs from client-server, in which few centralized servers satisfy requests from a large number of clients.

The concept of P2P is not new: for example, the original Usenet was carried over the peer-to-peer dialup UUCPnet. Similarly, in the beginning all nodes in Internet were peers, which cooperate to route packets among themselves. With the growth of these systems, however, the original peer-to-peer design has been lost. Currently, the Internet has evolved into a more hierarchical, client-server structure, in which a relatively small number of servers provide facilities like routing, e-mail, web and newsgroups to millions of client machines which simply act as service consumers. These client machines are second-class Internet citizens, as they are at the edges of the network, cut off from the DNS system because they have no fixed IP address.

The reason for the increasing interest for peer-to-peer can be attributed to the enormous success obtained by P2P applications like Napster [10], and more recently by Gnutella [6] and Freenet [3]. These systems enable end-users to establish a file-sharing network for the exchange of digital documents including music, movies and software. With their diffusion, millions of machines normally relegated to a client role started providing services to their respective communities. The big difference between existing Internet applications and peer-to-peer systems is that the latter are capable of exploiting what has been defined the "the dark matter" of the Internet, i.e. the huge amount of resources available at the edges of the network [11]. Unfortunately, most of the exchanged material is copyrighted, and this has resulted in equating P2P with the subversion of intellectual property.

Despite its poor popular reputation, P2P is extremely interesting from a technical point of view. Its completely decentralized model enables the development of many useful applications with reliability and high-availability characteristics previously unseen in Internet. As an example, systems like Freenet can solve what has been called the "slashdot effect": the more popular a piece of information is, the less available it becomes [1]. On the contrary, the number of replicas of a document (and thus its availability) on Freenet increases proportionally to its popularity [3]. Furthermore, P2P is not limited

Partial support for this work was provided by Sun Microsystems, Inc. through a Collaborative Research Grant and the Italian Ministry of University, Research and Technology.

to file sharing: several peer-to-peer projects have appeared in fields like distributed computing (e.g., Seti@Home [14] and Distributed.net [4]) and messaging and collaborative tools (e.g., Groove [7]).

The multitude of open-source and commercial P2P projects must confront common problems including security, scalability, reliability and routing. Unfortunately, given the novelty of the field, a theoretical and technical framework capable of supporting application development is still missing. This lack has been recently recognized by Intel, which founded the Peer-to-Peer Working Group with the aim of defining new standards for P2P applications [12], and by Sun Microsystems, which announced the JXTA initiative to provide basic infrastructure services for P2P application development [8].

The Anthill project, currently under development at the University of Bologna, was born with similar considerations. But differently from industry-proposed initiatives, which are deemed at establishing new programming and communication standards for the emerging P2P market, our goal is to provide a framework supporting researchers in the *design* and the *analysis* of new P2P algorithms.

In order to pursue these goals, we introduce a new approach for designing P2P systems, based on the ant colony paradigm [5] — an agent-based technique that has proven to be a powerful tool for solving optimization and communication problems in networks [13, 2]. In ant-based algorithms, artificial ants of limited individual capabilities move across a network of peer nodes trying to solve a particular problem. While moving, they build solutions and modify the problem representation by adding collected information. This reflects the behavior of real ants that cooperate through *stigmergy*, i.e. the capability to communicate among individuals through modifications induced in the environment. Real ants are known to locate the shortest path to a food source using as only information the trails of chemical substances called *pheromones* deposited by other ants. Although individual ants are unintelligent and have no explicit problem solving capability, ant colonies manage to perform several complicated tasks with high degrees of consistency. In other words, intelligent global behavior *emerges* through the interactions within colonies of such agents.

The Anthill project builds upon the similarities between P2P systems and social colonies of ants[1]. We believe that this biology-inspired paradigm could serve as a basis for a new framework supporting the design of completely decentralized P2P applications in large scale systems with a highly dynamic environment. In the Anthill framework, a P2P system is composed of a network of interconnected nests. Each nest is a peer entity capable of performing computations and store documents, running on the machine of some user. Nests react to request coming from users by generating ants, which are autonomous agents capable of traveling across the network by moving from one nest to another. While in a nest, ants are enabled to perform basic operations like performing computation, query the nest for documents, inserting new documents in it, and release information ("pheromone") about other documents and nests in order to help other ants to locate documents.

Anthill takes care of all low-level details such as communication, security, ant scheduling, etc. Researchers willing to implement new P2P protocols are simply required to write suitable ant algorithms using the framework API. In this way, they are free to concentrate their efforts on the organization of their P2P systems. Anthill provides also an evaluation framework which helps researchers to analyze the behavior of their ants and assess them. The evaluation is based on the execution of ant algorithms in simulated nest networks.

The similarities between Anthill and natural systems has been pushed even further, by enabling the integration of evolution techniques (i.e., genetic algorithms [9]) in our evaluation framework. In this way, it is possible to associate to ant algorithms a set of parameters influencing their behavior. This parameters constitutes the "genetic code" of an ant. The evaluation framework enable the selection of the fittest genetic codes (i.e., those which better solve a particular problem) using genetic algorithms.

The rest of the paper is organized as follows. Section 2 introduces the main elements of Anthill and discusses part of its API. Section 3 illustrates how evolution techniques can be used in Anthill. Section 4 describes the current implementation status. Finally, Section 5 concludes the paper and points out the directions for future work.

[1] Although ant colonies are generally competing for the control of the territory, there are ant species in which large networks of interconnected but autonomous social units cooperate by exchanging ants through the pheromone trails linking the nests. As an example, one super-colony of *Formica Yessensis* on the coast of Japan is reported to have had 1,080,000 queens and 306,000,000 workers in 45,000 interconnected nests.

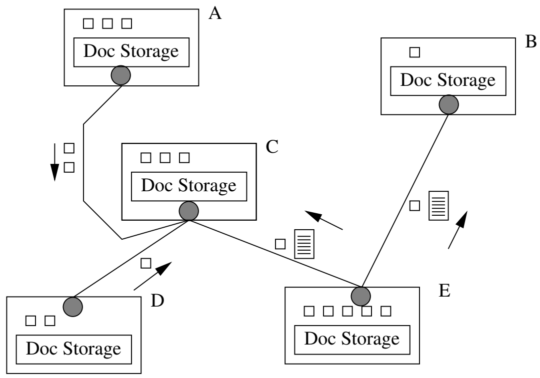

Figure 1: A nest network composed of five nests. Each nest contains a document storage and communicate with other nests through a gateway (gray spots). Some ants (black spots) are managed by ant managers included in nests, while others travel through the network possibly carrying documents with them.

2 The Anthill Framework

In this section, we describe the Anthill framework by illustrating its basic elements and its API specification. We have structured the framework in three main components: the Anthill *infrastructure*, the *ant algorithms* and the *evaluation framework*.

2.1 The Anthill Infrastructure

The Anthill infrastructure is composed of a network of nests (see Figure 1 for an example). *Nests* are P2P applications run by users on their machines. Each nest is capable of performing computations and to store documents on behalf of its user, or on behalf of foreign ants coming from other nests run by different users. A user may ask its nest for a specific service by performing requests and listening for responses. For example, in a music-sharing network a request could be a query for the songs of a particular author, and the response may be constituted of a set of addresses from which these songs can be downloaded. Alternatively, the response could contain the songs themselves.

The nest interface does not impose any particular format to requests and responses; moreover, it does not specify which services a nest should provide. The interpretation and satisfaction of requests are demanded to ants; in this way, programmers may develop new P2P services simply by implementing new ant algorithms based on the facilities offered by

```
public interface Nest {
   NestId getId();
   void addDocument(Document doc);
   void request(Request request,
      ResponseListener listener);
   void addNeighbour(NestId aid);
   void removeNeighbour(NestId aid);
   Neighbour[] getNeighbours();
}
```

Figure 2: The Nest interface.

the Anthill infrastructure.

Each nest in the Internet is associated with a unique *nest identifier*, which is composed of an IP address and a port number. A nest that knows the identifier of another nest may communicate directly with it; in this case, we say that the nest is a *neighbor* of , or that there is a connection between and . A collection of nests together with their connections form a *nest network*. The set of neighbors of a nest may be dynamic: additional neighbors may be added at run-time, thus modifying the shape of the network, simply by notifying their identifiers to the nest.

The interface Nest used to interact with nests is shown in Figure 2. The included methods enable to obtain the identifier of the nest, insert new documents, perform requests and obtain responses through listener interfaces, and finally add, remove and obtain information about neighbors.

Nest implementations are composed of three modules: a document storage, an ant manager and a gateway. A *document storage* is responsible for storing documents; different nest implementations may adopt different policies for managing the (inherently limited) space assigned to a storage for performing its duties. For example, last-recently-used policies may be used to discard documents rarely accessed. The *ant manager* takes care of scheduling computations of visiting ants. It is responsible for killing ants that are using too much resources. Finally, a *gateway* is the communication module that is responsible for sending and receiving ants, and for monitoring reachability of remote nests.

2.2 Ant Algorithms

As mentioned in the introduction, ants are autonomous agents capable of traveling across a nest network and interact with the nests they visit in order to pursue their goal. Ants must implement the Ant interface shown in Figure 3. The generic run() method included in it contains the ant's algorithm

```
public interface Ant {
   void run(AntView view);
}
```

Figure 3: The Ant interface.

```
public interface AntView {
   void move(NestId id);
   Document[] request(Request request);
   void addNeighbour(NestId id);
   NestId getNeighbours();
   Pheromone getPheromone(NestId id);
   void setPheromone(NestId id,
      Pheromone pheromone);
   void addDocument(Document doc);
   void result(Request request,
      Result result);
}
```

Figure 4: The AntView interface.

and is invoked by the ant manager described in the previous section. Apart from the algorithm, ants are characterized by a small amount of memory containing the ant's state. The behavior of an ant is determined by its algorithm and its current state, possibly in a stochastic way. For example, an ant may probabilistically decide to not follow what is believed the best route to a document in order to explore new regions of a network and find shorter paths.

Ants are generated by nests in response to requests; each ant is aimed at satisfying the request for which it has been generated. Ants move from nest to nest until they succeed in their goal (e.g., until they reach a nest containing the required documents). A successful ant returns to the nest that has generated it in order to deliver a response to the user. When an ant moves from a nest to another, its state is transmitted from the source nest to the destination, where the ant is re-created. The ant object at the source is destroyed. Note that if the ant algorithm is not known to the destination nest, the algorithm must be transmitted as well. This is obtained by exploiting the possibility of remote code download offered by Java, as explained in the next section.

The run() method of an ant is invoked in every nest visited during its trip. The ant manager is responsible for confining the ant algorithm in a controlled environment, where a limited set of actions is possible. This confinement is obtained through the security features of Java, as explained in the next section. The only methods which may be invoked are those contained in interface AntView (Figure 4), which enable to perform the following operations:

- move to another nest by specifying its identifier;
- query the local document storage looking for a particular set of documents;
- inform the nest of the existence of other nests by adding new nest identifiers;
- obtain the list of neighbors known to the local nest;
- getting and setting the pheromone information associated to links to other nests;
- insert new documents in the storage;
- notify the nest about a request for which the ant has been able to produce a response.

Note that in order to insert new documents in the storage, ants may be required to carry documents with them when they travel across the network. Figure 1 shows an example of this behavior: the ant going from nest to nest is carrying a document with it; the same is doing the ant going from nest to nest .

As for requests and responses, the Anthill framework does not specify any particular format for pheromone information; we left ant developers free to design the format they prefer, in relation to their particular ant implementation. For example, a Freenet-like ant will calculate the hash keys of the stored documents and will organize the pheromone information on the basis of these keys [3].

2.3 The Evaluation Framework

In order to evaluate the "goodness" of new ant algorithms, the Anthill framework includes also an evaluation framework through which the behavior of a particular ant implementation may be simulated and assessed. Each simulation is called *scenario* and it is composed of a collection of interconnected nests and a scheduling of requests to be performed. The simulation proceeds by executing the various requests and by monitoring several algorithms parameters such as the number of request performed, the number of request satisfied, the number of ant moves performed, the number of documents copied from one nest to another, and so on. Scenarios are generated randomly on the basis of some probabilistic distribution. Different scenarios implementations may use distinct distributions and thus produce different network organizations. Using the framework, programmers can evaluate several scenarios at once and obtain averaged values for the collected statistics.

3 Using Evolution in Anthill

In Anthill, it is possible to further extend the analogy between P2P systems and the natural world by using evolution-based techniques such as genetic algorithms [9]. As mentioned above, the behavior of ants is determined by their algorithms. These algorithms may be parameterized in various way; for example, an ant may deterministically choose to follow the path presenting the strongest pheromone concentration, or stochastically select any of the neighbors of its current nest following some exploration probability parameter. We believe that genetic algorithms may help to select the best set of parameters for an ant algorithm.

We have extended the evaluation framework by inserting the concept of *chromosome*, which represents a set of parameters for a particular ant algorithm. The evaluation framework maintains a *population* (i.e., a collection) of chromosomes. At each generation of the genetic algorithm, every chromosome in the population is evaluated by using it as parameter set for the ants used in a scenario. The fittest chromosomes (i.e., those that produces the ants which better solve a particular problem) are manipulated genetically using crossover and mutation operators to produce the next generation. The other chromosomes are discarded. This genetic selection continues until a fixed number of generations are executed, or some fitness objectives are met.

Besides from exploiting genetic techniques for the off-line selection of the fittest ant parameters, we plan also to investigate if genetic techniques may be used during the run-time functioning of nest networks; for example, a nest network could select the fittest ant algorithm by launching ants of distinct types and with different chromosomes (i.e., set of parameters) for the same request, and then rating them using a local fitness evaluation using measures like the time needed to satisfy a request. Nests may "steal" the algorithm and the chromosomes of the ants visiting them and use cross-over and mutation techniques to generate new ants. It is interesting to note that this would lead to a distributed computing peer-to-peer system whose task is to select the fittest algorithm for other P2P applications.

4 Implementation Status

The Anthill project is still in its early stages. As a first step, we have defined the set of interfaces representing the basic components of the infrastructure, including nests, document storages, ant managers and gateways. We have also designed interfaces for the main elements of the evaluation and genetic frameworks, including scenarios, chromosomes and populations. Furthermore, we have written a first prototype implementation for all these interfaces.

We have designed our framework in order to minimize the differences between real networks, composed of a distributed collection of nests communicating through the Internet, and evaluation networks simulated in a single machine. For example, the same nest implementation is used in both cases; what differs are the implementations of gateways and nest identifiers, which are based on TCP in the former case and on local method invocations in the latter. Ants are not required to be aware if they are being deployed in a real or simulated network, as their interactions are limited to interface AntView.

Apart from the realization of the infrastructure, we are working also on the development of the first ant implementations and their testing using the evaluation framework. The first results seem to be encouraging, as we are observing improvements in the efficiency of ant algorithms as the fittest chromosomes are selected. In order to compare our implementations with existing P2P systems, we plan to implement ants mimicking the behavior of Freenet [3] and Gnutella [6].

The framework is being written in Java, in order to exploit its portability, its rich security features and the possibility of remotely download code on demand. These features are particularly useful in agent-based systems like Anthill, enabling the transmission of the agent code and its execution in a controlled environment. We have implemented a class loader capable of requesting the downloading of the code of an unknown ant from the nest that sent it, and caching downloaded ant classes on the local disk to avoid multiple downloads. Using the security model of Java, it has been possible to confine the execution of ants in a controlled environment ("sandbox'), forcing them to use the AntView interface as the only mean of interactions with the nest.

5 Conclusions

In this paper, we have introduced a new approach for building distributed peer-to-peer applications and we have presented a programming framework aimed at supporting this approach.

The Anthill framework is based on the separation of roles existing between nests and ants. Nests constitutes the infrastructure on which P2P applications will be builded; they manage low-level details as security, communication, neighbor management, resource management, ant scheduling, and so on. Ants, on the other hand, will implement the actual P2P algorithm through which high-level services will be provided. Using Anthill, developers of new P2P systems are free to concentrate their efforts on the design of ants, and thus on the design of efficient and adaptive peer-to-peer protocols.

Anthill may be compared with similar efforts of developing new infrastructures for the P2P arena. Our framework differs from the proposals from the P2P Working Group and from Sun, as it is aimed at supporting the design and the analysis of new P2P protocols, rather than simply establishing new communication and programming standards for P2P. Nevertheless, we believe that if opportunely integrated with them, our framework will be able to benefit from these standards. We are particularly interested in JXTA, that like Anthill is based on Java and promises to provide security features currently missing from Anthill. We plan to investigate intersections among our project and JXTA as soon as a specification for it is available.

As mentioned above, the Anthill project started recently. Much work remains to be done, including the implementation of new ant algorithms, the definition of realistic test scenarios on which simulating the behavior of ants, and the realization of the run-time genetic framework. The Anthill project is based on an open-source license. We welcome every form of cooperation on the project, such as the development of ants implementing new P2P protocols, but also the improvement of the Anthill framework prototype implementation. In fact, an additional aim of this paper is the stimulation of such cooperations. We plan to use Anthill in the "Complex Adaptive Systems" course to be held in the Department of Computer Science at the University of Bologna during second semester 2001. Students will be invited to experiment with our framework by developing new ant implementations and comparing them.

References

[1] S. Adler. The Slashdot Effect: an Analysis of three Internet publications. *Linux Gazette*, 38, March 1999.

[2] G. Di Caro and M. Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[3] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In Hannes Federrath, editor, *Proceedings of the-Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, July 2000.

[4] Distributed.net Home Page. Url: `http:// www. distributed. net`.

[5] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1):29–41, 1996.

[6] Gnutella Home Page. Url: `http:// gnutella. wego. com`.

[7] Groove Home Page. Url: `http:// www. groove. net`.

[8] K. Kayl. Pushing the Boundaries of Distributed Computing. Url: `http:// www. javasoft. com/ features/ 2001/ 02/ peer.html`.

[9] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, April 1998.

[10] Napster Home Page. Url: `http:// www. napster. com`.

[11] A. Oram. Peer-to-peer Makes the Internet Interesting Again. Url: `www.oreillynet.-com/pub/a/linux/2000/09/22/-p2psummit.html`.

[12] The Peer-to-Peer Working Group Home Page. Url: `http:// www. peer-to-peerwg. org`.

[13] R. Schoonderwoed, O. Holland, J. Bruten, and L. Rothkrantz. Load Balancing in Telecommunications Network. *Adaptive Behavior*, 5(2), 1997.

[14] Seti@Home Home Page. Url: `http:// setiathome. ssl. berkeley. edu`.