AI System in Manet

Author [1]
Dr Ritu Soni
(HOD, Dept. of Computer Science and Application)
Guru Nanak Girls College
Yamuna Nagar, Haryana –135001 (India)
Tel.: 01732-329869

Author [2]
Dr. Sudhir Dawra
(Astt. Professor, Dept. of Computer Engineering)
Al- Falah School of Engineering and Technology
Dhuaj, Faridabad, Haryana-12004 (India)
Tel.: 0129-2206223,2206593 Mob: 9811677627

ABSTRACT

Mobile ad-hoc networks have been deployed in various scenarios, but their scalability is severely restricted by the human operators' ability to configure and manage the network in the face of rapid change of the network structure and demand patterns. In this paper, we present a self-organizing approach to MANET management which have AI in their nodes that follows general principles of engineering for communicating between each other.

INTRODUCTION

The management of mobile ad-hoc networks presents different challenges that may overwhelm traditional network management approaches. Such networks are highly dynamic, severely constrained in their processing and communications resources, distributed and decentralized. Thus, centralized management approaches requiring accurate and detailed knowledge about the state of the overall system may fail, while decentralized and distributed strategies become competitive.

Designing decentralized and distributed applications in an environment as dynamic, noisy and unpredictable as MANETs requires an approach that is robust, flexible, adaptive and scalable.[1] Self-organizing systems of agents with emergent system-level functions offer these features, but it is often far from obvious how the individual agent processes need to be designed to meet the overall design goal. In this paper we present a artificial intelligence in the nodes of MANET domain.

Then we discuss the nature of self-organizing applications in general and propose a set of design principles. We offer a solution to the MANET management problem based on fine-grained agents dynamically interacting in the network environment.

MANET MANAGEMENT PROBLEM

A network of moving nodes that may communicate within a limited range, that may fail temporarily, and that may host client and server processes. Every node carries a client and some nodes carry a server process. A server provides a stateless and instantaneous service to a client upon request if there exists a communications path between the client and the server and if the server node is currently active. Servers in our model have no capacity constraints, and may serve as many clients at the same time as requests arrive.

The design goals are three-fold: for the current topology of the network determined by node locations, communications ranges and node availability, decide

- which server nodes should actually expend battery power to execute the server process;
- to which server node a particular client should send its next service request; and
- where to relocate server nodes to meet the current demand by the clients.

Thus, the network must be provided with mechanisms that self-diagnose the current network state (e.g., breaking of connections, availability of new connections, failure of nodes) and provide the information in a way that enables it to self-configure the ongoing processes for optimal performance.

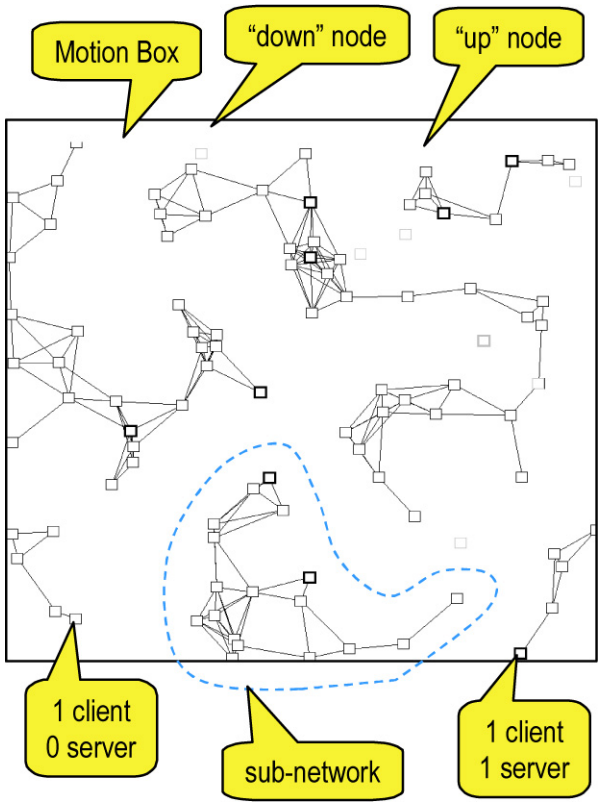

Fig. 1. Application domain: ad-hoc network of (randomly) moving nodes with power and communications limitations.

SELF-ORGANIZING APPLICATIONS

In engineering a self-organization application such as a MANET, it is helpful to think in terms of imposing three successive restrictions on the space of all possible multiprocess systems, outlined in Fig. 2.

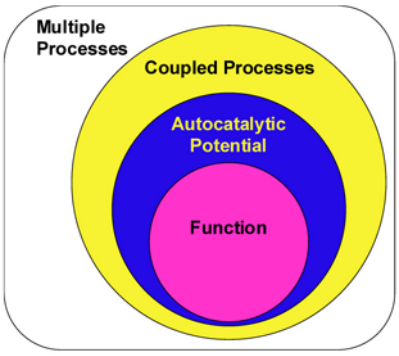

Fig. 2. Three Steps to self-organization

- The various processes must be *coupled* with one another so that they can interact.
- This interaction must be self-sustaining, or *autocatalytic*.
- The self-organizing system must produce *functions* that are useful to the system's stakeholders.

In discussing each of these, we first review the concept and its mechanisms, and then discuss design principles to which it leads. We also offer further discussion of these and other issues for engineering self-organizing systems.

COUPLED PROCESSES

Agents must exchange information if they are to self-organize. Different patterns of information exchange are possible, and can be classified along two dimensions: Topology and Information Flow.

The Topology dimension depends on two different kinds of relations that agents can have with one another. When the agents can say "No" to one another within the rules of the system, they are "peer agents." When one of them (say agent A) can say "No" to the other (B), but B cannot say "No" to A, we call A the "distinguished agent" and B the "subordinate." These concepts can be developed more formally through dependency and autonomy theory. Centralized information exchange is between a distinguished and a subordinate agent, while decentralized information exchange is between peer agents.

The Information Flow dimension relies on environmental state variables that the agents can manipulate and sense.[2] All information exchange is ultimately mediated by the environment, but the role of the environment is sometimes not modeled explicitly.

The information flow from one agent to another is Direct if no intermediate manipulation of information is modeled, and Indirect if it is explicitly modeled. Decentralized indirect mechanisms have a number of advantages, including simplicity, scalability, robustness and environmental integration. Four design principles support coupling among processes, and in particular stigmergic coupling.

Coupling 1: Use a distributed environment. —Stigmergy is most beneficial when agents can be localized in the environment with which they interact. A distributed environment enhances this localization, permitting individual agents to be simpler (because their attention span can be more local) and enhancing scalability.

Coupling 2: Use an active environment. —If the environment supports its own processes, it can contribute to overall system operation. For example, evaporation of pheromones in the ants' environment is a primitive form of truth maintenance, removing obsolete information.

Coupling 3: Keep agents small.—Agents should be small in comparison with the overall system, to support locality of interaction. This criterion is not sufficient to guarantee locality of interaction, but it is a necessary condition. The fewer agents there are, the more functionality each of them has to provide, and the more of the problem space it has to cover.

Table 1. Categories of Information Exchange

		Topology of Inter-Agent Relationships	
		Centralized (between Distinguished and Subordinate agents)	**Decentralized** (among Peer agents)
Information Flow	**Direct** (messages between agents)	***Construction*** (Build-Time) ***Command*** (Run-time)	***Conversation***
	Indirect (non-message interaction)	***Constraint***	***Stigmergy*** (generic) ***Competition*** (limited resources)

Coupling 4: Map agents to Entities, not Functions. —Choosing to represent domain entities rather than

functions as agents takes advantage of the physical fact that entities are bounded in space and thus have intrinsic locality.[3] Functions tend to be defined globally, and making an agent responsible for a function is likely to lead to many non-local interactions.

AUTOCATALYTIC POTENTIAL

The concept of autocatalysis comes from chemistry. A catalyst is a substance that facilitates a chemical reaction without being permanently changed. In autocatalysis, a product of a reaction serves as a catalyst for that same reaction. An autocatalytic set is a set of reactions that are not individually autocatalytic, but whose products catalyze one another. The result is that the behaviors of the reactions in the set are correlated with one another. If reaction A speeds up (say, due to an increased supply of its reagents), so does any reaction catalyzed by the products of A.[4] If A slows down, so do its autocatalytic partners. This correlation causes a decrease in the entropy of the overall set, as measured over the reaction rates.

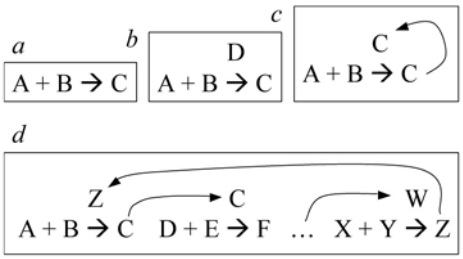

Fig. 3. Relations among Processes. --a) A simple reaction. b) D catalyzes the conversion of A and B to C. c) An autocatalytic reaction. d) An autocatalytic set of processes (shown as a ring, but other topologies are possible).

Not all coupled processes are autocatalytic. Autocatalyticity requires a continuous flow of information among the processes to keep the system moving. If the product of process A catalyzes process B, but process B's products have no effect (either directly or indirectly) on process A, the system is not autocatalytic. Furthermore, a system might be autocatalytic in some regions of its state space but not in others. [5]

It is natural to extend this concept from chemistry to any system of interacting processes, such as a multi-agent system. A set of agents has *autocatalytic potential* if in some regions of their joint state space; their interaction causes system entropy to decrease (and thus leads to increased organization). In that region of state space, they are *autocatalytic*.

Two points are important to understand about autocatalyticity.

1. In spite of the reduction of entropy, autocatalyticity does not violate the Second Law of Thermodynamics. The rationalization is most clearly understood in the stigmergic case. Entropy reduction occurs at the macro level (the individual agents), but the dissipation of pheromone at the micro level generates more than enough entropy to compensate. This entropy balance can actually be measured experimentally.

2. Information flows are necessary to support self-organization, but they are not sufficient. A set of coupled processes may have a very large space of potential operating parameters, and may achieve autocatalyticity only in a small region of this space.

Nevertheless, if a system does not have closed information flows, it will not be able to maintain self-organization.

The need to achieve autocatalysis leads to three design principles.

Autocatalysis 1: Think Flows rather than Transitions. — Computer scientists tend to conceive of processes in terms of discrete state transitions, but the role of autocatalysis in supporting self-organization urges us to pay attention to the flows of information among them, and to ensure that these flows include closed loops.

Autocatalysis 2: Boost and Bound. —Keeping flows moving requires some mechanism for reinforcing overall system activity. Keeping flows from exploding requires some mechanism for restricting them. These mechanisms may be traditional positive and negative feedback loops, in which activity at one epoch facilitates or restrains activity at a successive one. Or they may be less adaptive mechanisms such as mechanisms for continually generating new agents and for terminating those that have run for a specified period ("programmed agent death").

Autocatalysis 3: Diversify agents to keep flows going. — Just as heat will not flow between two bodies of equal temperature, and water will not flow between two areas of equal elevation, information will not flow between two identical agents. They can exchange messages, but these messages carry no information that is new to the receiving agent, and so cannot change its state or its subsequent behavior. Maintaining autocatalytic flows requires diversity among the agent population. This diversity can be achieved in several ways. Each agent's *location in the environment* may be enough to distinguish it from other agents and support flows, but if agents have the same movement rules and are launched at a single point, they will not spread out. If agents have different experiences, *learning* may enable them to diversify, but again, reuse of underlying code will often lead to stereotyped behavior. In general, we find it useful to incorporate a *stochastic element* in agent decision-making. In this way, the decisions and behaviors of agents with identical code will diversify over time, breaking the symmetry among them and enabling information flows that can sustain self-organization.

FUNCTION

Self-organization in itself is not necessarily useful. Autocatalysis might sustain undesirable oscillations or thrashing in a system, or keep it locked in some pathological behavior. We want to construct systems that not only organize themselves, but that yield structures that solve some problem. There are two broad approaches to this problem, broadly corresponding to the distinction in classical AI between the scruffy and the neat approaches.

As the use of self-organizing systems matures, a hybrid of both approaches will probably be necessary.

One approach, exemplified by work in amorphous computing is to build up, by trial and error, a set of programming metaphors and techniques that can then be used as building blocks to assemble useful systems.

An alternative approach is to seek an algorithm that, given a high-level specification for a system, can compute the local behaviors needed to generate this global behavior.

State-of-the-art algorithms of this sort are based not on *design*, but on *selection*. Selection in turn requires a system with a wide range of behavioral potential, and a way to exert pressure to select the behaviors that are actually desired.

One way to ensure a broad range of behavioral potential is to construct nonlinear systems with formally chaotic behavior. The basic idea is to let the chaotic dynamics explore the state space, and when the system reaches a desirable region, to apply a small control force to keep the system there. It may seem that chaos is a complicated way to generate potential behaviors, and that it would be simpler to use a random number generator. In fact, virtually all such generators *are* in fact nonlinear systems executing in their chaotic regime. [6]

In a multi-agent system, the key to applying this generate-and-test insight is finding a way to exert selective pressure to keep the system balanced at the desired location. Natural systems have inspired two broad classes of algorithm for this purpose: synthetic evolution, and particle swarm optimization.

Synthetic evolution is modeled on biological evolution. Many different algorithms have been developed, but they share the idea of a population of potential solutions that varies over time, with fitter solutions persisting and less fit ones being discarded.

The variational mechanisms explore the system's potential behaviors, while the death of less fit solutions and the perpetuation of more fit ones is the control pressure that selects the desired behavior.

Particle swarm optimization is inspired by the flocking behavior of birds. In adaptations of this behavior to computation, solutions do not undergo birth and death. Instead, they are distributed in some space (which may be the problem space, or an arbitrary structure), and share with their nearest neighbors the best solutions they have found so far. Each solution entity then adjusts its own behavior to take into account a blend of its own experience and that of its neighbors. Market-based bidding mechanisms are a variation on particle swarm optimization.

The similarity lies in selection via behavioral modification through the exchange of information rather than changes in the composition of the population. The approaches differ in the use that is made of the shared information. In the particle swarm, agents imitate one another, while in bidding schemes, they use this information in more complicated computations to determine their behavior.

The need to adjust the system's function to meet requirements leads to three design principles.

Function 1: Generate behavioral diversity. —Structure agents to ensure that their collective behavior will explore the behavioral space as widely as possible. One formula for this objective has three parts.

1. Let each agent support multiple functions.

2. Let each function require multiple agents.

3. Break the symmetry among the agents with random or chaotic mechanisms. The first two points ensure that system functionality emerges from agent interactions, and that any given functionality can be composed in multiple ways. The third ensures a diversity of outcomes, depending on which agents join together to provide a given function at a particular time.

Function 2: Give agents access to a fitness measure. — Agents need to make local decisions that foster global goals. A major challenge is finding measures that agents can evaluate on the basis of local information, but that correlate with overall system state. In one application, we have found the entropy computed over the set of behavioral options open to an agent to be a useful measure of the degree of overall system convergence that agents can use to make intelligent decisions about bidding in resource allocation problems.

Function 3: Provide a mechanism for selecting among alternative behaviors.— If an adequate local fitness metric can be found, it may suffice to guide the behavior of individual agents. Otherwise, agents should compare their behavior with one another, either to vary the *composition* of the overall population (as in synthetic evolution) or to enable individual agents to vary their *behavior* (as in particle swarm optimization).

EMERGENT MANET MANAGEMENT

Before we present our self-organizing design, we discuss a baseline solution that achieves the globally optimal service performance, but does not satisfy the severe resource constraints inherent to MANETs.

Comparison with the Global Solution

We implemented both solutions (global and local) in a software demonstration that allows us to observe the ongoing operation of the system visually, but which does not provide any systematic experimentation (parameter sweep, formal analysis) opportunities. Let us consider the static case first. We reduce the probability of nodes to fail to zero and freeze their random motion. Thus, the MANET topology remains fixed to the initial state. With the global solution, clients that are in reach of a server immediately find the server and succeed in utilizing the service, while all others continually fail.

The performance of the system in providing the service to the client is maximal. Since the servers are always on, the

usage of processing power by the server processes is maximal too.

Our local solution begins with a very low performance, since all clients except those that are co-located with a server on the same node do not have any knowledge about available servers. But once a co-located client initiates its first service request to its local server, the knowledge about this server rapidly propagates through the server's sub-network and other clients start utilizing the server too. If there are more than one server in a sub-network, then the clients' asymmetric reinforcement of their scorecards and the servers' activation learning quickly breaks symmetries and focuses the clients' request onto one server. [7] At this point, the performance of the system is as good as with the global solution, but the resource usage is lower if the network topology provides for multiple servers in a sub-network.

Let us now introduce a change in the topology as either a node fails or random movement changes the link structure. Such a change may or may not change the reachability or availability of servers for a sub-population of clients. The more clients are affected, the more severe are the effects of this topology change. With the global solution, the performance of the system immediately adjusts to the maximally possible utilization success rate. In contrast, our local approach experiences a temporary drop off in performance as the affected clients first try to utilize servers that are no longer available and then exchange knowledge about other locally available servers that had shut down. The more severe the topology change is, the larger is the aggregated loss of performance during this re-learning cycle.

Finally, let us repeatedly trigger topology changes. These changes occur stochastically and we hypothesize that the frequency of changes of a fixed severity follows a power law, where more severe changes are exponentially less frequent. As the duration of the re-learning cycle increases with the severity of the changes, there will be a point where the next change occurs before the re-learning is completed. We expect a clear phase change in the performance (and thus applicability) of our local solution as the dynamics of topology change surpass a threshold.

CONCLUSION

Using self-organization and emergence to engineer system-level functionality may be advantageous in many application domains, but often it is not obvious how to design the underlying processes to achieve the desired function. In this paper we introduced a set of general design principles that reduce the overwhelming set of options at design decision points and thus guide the engineer towards a successful application design.

We demonstrate the AI design approach in the real-world example of a MANET management system. MANETs meet many criteria of application domains in which self organization and emergence of functions is required to provide the necessary robustness and scalability. These networks are highly Dynamic, processing and communication is Decentralized, and local decision makers are Distributed and Deprived of resources.

Our self-organizing agent system for MANET management performs close to the optimum achieved by a global coordination mechanism that could not realistically be implemented on a real-world network since it does not meet the resource constraints. The agents in our system rely on stigmergy to coordinate their activities.

REFRENCES

[1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous Computing. *Communications of the ACM*, 43(5): 74-82, 2000.

[2] S. Brueckner and H. V. D. Parunak. Information-Driven Phase Changes in Multi-Agent Coordination. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 2003)*, Melbourne, Australia, pages 950-951, 2003.

[3] C. Castelfranchi. Founding Agent's 'Autonomy' on Dependence Theory. In *Proceedings of 14th European Conference on Artificial Intelligence*, Berlin, Germany, pages 353-357, IOS Press, 2000.

[4] C. Jacob. *Illustrating Evolutionary Computation With Mathematica*. San Francisco, Morgan Kaufmann, 2001.

[5] J. Kennedy, R. C. Eberhart, and Y. Shi. *Swarm Intelligence*. San Francisco, Morgan Kaufmann, 2001.

[6] E. Ott, C. Grebogi, and J. A. Yorke. Controlling Chaos. *Physical Review Letters*, 64(11):1196-1199, 1990.

[7] H. V. D. Parunak and S. A. Brueckner. Engineering Swarming Systems. In F. Bergenti, M.-P. Gleizes, and F. Zambonelli, Editors, *Methodologies and Software Engineering for Agent Systems*, pages (forthcoming). Kluwer, 2003.

.