Proceedings of ESOA04 (Engineering Self-Organising Agents, at AAMAS 04, New York, NY), Springer, 2005. http://www.springeronline.com/lncs, © Springer-Verlag.

Analyzing Stigmergic Learning for Self-Organizing Mobile Ad-Hoc Networks (MANET's)

H. Van Dyke Parunak, Sven A. Brueckner

Altarum Institute
3520 Green Court, Ann Arbor, MI 48105, USA
{sven.brueckner, van.parunak}@altarum.org

Abstract. In recent years, mobile ad-hoc networks (MANET's) 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 based on stigmergic agents and demonstrate how to analyze its performance under different deployment assumptions. Our results emphasize the importance of attention to notions from dynamical systems theory in designing and deploying multi-agent systems.

1. Introduction

The challenges of managing mobile ad-hoc networks (MANET's) [1] 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.

We have successfully applied fine-grained agent architecture modeled on algorithms used in biological systems [11] to a range of real-world problems, including manufacturing control [2], pattern recognition in sensor networks [4], collaboration and task assignment among multiple mobile platforms [13], path planning for unmanned vehicles [16], and information retrieval in massive data [17]. This paper explores the applicability of these mechanisms to another domain, mobile ad-hoc communication networks (MANET's). Like other domains in which swarming is effective, MANET's are distributed, decentralized, and dynamic. Self-organizing systems of agents with emergent system-level functions offer an approach that is robust, flexible, adaptive and scalable. By applying our techniques to a new domain, we gain experience with their capabilities and restrictions, and further exercise the development methodology that we are developing for such systems [12, 15].

Section 2 presents a concrete management problem in the MANET domain. Section 3 offers a solution based on fine-grained agents dynamically interacting in the network environment. Section 4 offers experimental evidence for the effectiveness of our solution. Section 5 concludes.

2. The MANET Server Management Problem

Figure 1 offers an overview of the MANET domain. Assume a network of (randomly) moving nodes that may communicate within a limited range, and may fail temporarily. A canonical example of an application for a MANET is a fleet of vehicles (say, trucks or dismounted troops in a military operation, or rovers exploring a remote planet) equipped with line-of-sight radios.

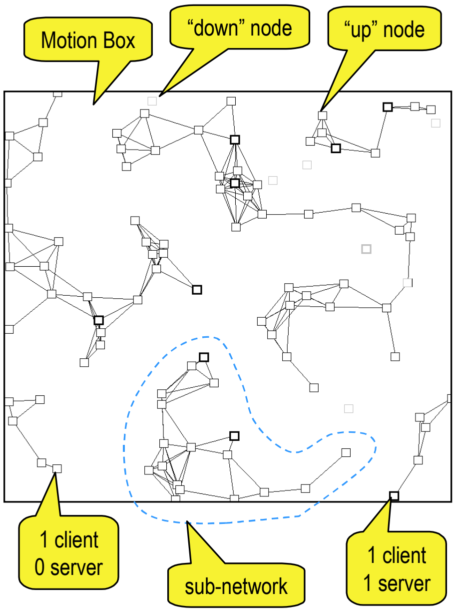

Fig. 1. Domain Overview

We focus our attention on configurations in which nodes may host distinct client and server processes. Every node carries a client and some nodes carry a server process. Examples of services that might be restricted to some vehicles include

- long-range communications links back to a remote commander;
- wide-range sensors that can provide an integrating context for more local sensors carried on most vehicles;
- target recognition databases and data fusion capabilities that can provide interpretive support for platforms with more local access.

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.

Because the nodes are mobile, weight and space are constrained, limiting the power available for communications and processing. Some of the likely services (long-range communications or sensing) impose especially high power demands on the servers, making it desirable to operate them only when they are needed to support the demands from the rest of the fleet. Vehicle movement must satisfy two constraints: achieving mission objectives and maintaining communication connectivity. In the simple example we describe here, all vehicles share both objectives, but techniques that we have demonstrated elsewhere [13] permit vehicles to specialize for different tasks, so that some vehicles would dedicate themselves to serving as communication relays, reducing the constraints on the other vehicles imposed by the need to maintain connectivity.

The server management problem requires answering three questions: given the current topology of the network determined by node locations, communications ranges and node availability, decide

1. which server nodes should actually expend battery power to execute the server process;
2. to which server node a particular client should send its next service request; and
3. 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 appropriately. These functions could be satisfied if all servers executed constantly and if all clients had global knowledge of the overall system (Figure 2), but such a solution is impractical.

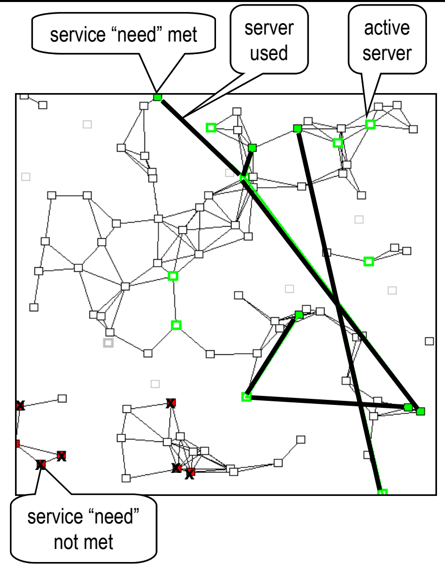

Fig. 2. Global Solution

3. Emergent MANET Management

A fine-grained, self-organizing agent system can solve the service location problem specified in Section 2. Our solution starts with the following initial conditions:

- Server processes shut down immediately if no requests arrive.
- A client does not know about the location of servers in the network, unless the client is co-located with a server on the same node.
- Server nodes move randomly (a zeroth order approximation to mission-motivated movement)..

Thus, in terms of our design goals, we preserve maximum battery power, but most clients' service needs are not met since they don't know which server to address.

We now define a co-evolutionary learning process based on individual reinforcement. This learning process has three components.

1. The server population learns to maintain an appropriate number of active server processes,
2. and to adjust the position of these processes as they learn about the clients who are using them.
3. The client population learns to direct requests to active servers.

3.1 Server Activation Learning

Any server node is aware of the incoming requests from one or more clients. If the server process is running, then these requests are served, otherwise they fail, but the node will immediately start up the server process to be available for any new requests in the next cycle. While the server process is running, it tracks the number of incoming requests. If there are no requests, it will begin a countdown. It will either abort the countdown if new requests arrive (and are served), or shut down if it reaches the end of the countdown.

Initially, the duration of the countdown is zero. Thus, server processes are shut down as soon as no new requests come in. We define the following simple reinforcement learning process to adjust the duration of the next countdown:

(+) If a request from a client arrives and the server process is down, we increase the length of the countdown period for subsequent countdowns, since apparently the server should have been up and we lost performance (failed to serve a request) while the server was down.

(–) If no request arrives while the countdown proceeds and the server process reaches the end of the countdown, then we decrease the length of the countdown period for subsequent countdowns, since apparently the server could have been down already and we wasted resources (battery power) while the server was up.

Driven by the demand pattern as it is perceived at the particular server node, the server process learns to maintain the optimal availability. In effect, the server learns the mean time between requests and adjusts its countdown length accordingly to stay up long enough. With this learning mechanism in place, the client population will now assume the role of the teacher as it generates a demand signal that leads some servers to stay down (extremely short countdown) while others stay consistently up (extremely long countdowns).

3.2 Client Preference Learning

Initially, only clients that are co-located with a server on the same node have any information about possible server addresses. These clients will become the source of knowledge of the client population as they share this information with their neighbors.

Knowledge Representation.—Clients manage their knowledge about and evaluation of specific servers in a dynamic set of scorecards, one for each server they know. A scorecard carries the address of the server, a score in favor (*pro*) and a score against (*con*) using this server. The current score of a server is computed as *pro - con*.

Decision Process.—When a client needs to select a server, it normalizes the current scores of all scorecards so that they add up to one and selects a server with a probability equal to the normalized score (roulette wheel selection). Thus, servers with a low current score compared to others have a lower probability of being chosen by the client. If the client currently does not have any scorecards, then it can only contact a server if it co-located with one, otherwise its service need will not be met in this decision cycle.

Information Sharing.—If a client selects a server on a node that is currently within reach, it sends a request to the server and shares the outcome of this interaction with its direct neighbors. If the request is met, the client increases its own *pro* score of that server by one and sends the same suggestion to its direct neighbors. If the request is not met, the *con* scores are increased in the same way. These suggestions to the neighbors may lead to the creation of new score cards at those neighbors if they had not known about this server before. Thus knowledge about relevant servers spreads through the network driven by the actual use of these servers. Furthermore, the success or failure of the interaction with a server reinforces the preferences of the client population and thus (with a random component to break symmetries) dynamically focuses the attention on a few active servers while encouraging de-activation for others (see "Server Activation Learning").

Truth Maintenance.—The constant change of the network topology, driven by the node movements and their failures, requires that the client population continuously update its knowledge about reachable servers and their evaluation. While the score-sharing mechanism ensures that the performance of a reachable server is continuously re-evaluated, the clients still need a mechanism to forget references to servers that do not exist anymore or that are out of reach now. Otherwise, in long-term operation of the system, the clients would drown in obsolete addresses.

A client "evaporates" its scores (*pro* and *con* individually) by multiplying them with a globally fixed factor between zero and one in each decision cycle. Thus, both scores approach zero over time if the client or its neighbors do not use the server anymore. If both scores have fallen below a fixed threshold, then the scorecard is removed from the client's memory – the client forgets about this server.

A client also chooses to forget about a particular server, if the *con* score dominates the *pro* score by a globally fixed ratio ($con / (con + pro) > \text{threshold} > 0.5$). Thus, servers that are trained by the client population to be down are eventually removed from the collective memory and are left untouched. They only return into the memory of clients if all other servers have also been forgotten and their co-located client is forced to use them.

3.3 Server Node Location Learning

In a co-evolutionary process, the server and client populations learn which clients should focus on which servers. We can stabilize this preference pattern and reduce the need for re-learning by decreasing the likelihood that the connection between a client and its chosen server is disrupted. Since the risk for a disruption of the path between a client and a server generally increases with the distance between their nodes, moving the server node towards its current clients will decrease this risk.

We assume that any client and server processes have means to estimate their respective node's current spatial location and that the server node may actually control its movement within certain constraints if it chooses to.

As a client sends a request to a server, it includes its current location in the request message. The server node computes the vector between the client and the server location and adds up all vectors from all requests within a decision cycle. Vectors of requests that failed are negated before they are added to the sum. The resulting combined vector determines the direction of the next move of the server node. If the requests failed because the server process was down, then the node moves away from the “center of gravity” of the clients that contacted this server. Otherwise, the node will move toward these clients. The length of the step for the server node is fixed to a global constant, characterizing the physical ability of the node to move.

3.4 Stigmergic Coordination

The coordinated behavior of many simple agents (server, client, node) in the highly dynamic and disruptive MANET environment emerges from peer-to-peer interactions in a shared environment driven by simple rules and dynamic local knowledge. The individual components of the system are not explicitly aware of the overall system functions of self-diagnosis and self-reconfiguration.

The coordination mechanism detailed in this demonstration is an example of stigmergy, in which individual agent activity is influenced by the state of the agent and its local environment. As agent activity manipulates the environment, subsequent agent activity dynamics may change (Figure 3). If this flow of information between the agents through the environment establishes a feedback loop that decreases the entropy of the options of the individual agents, then coordinated behavior emerges in the population. We engineer the agent behavior and the indirect information flow, so that the emergent coordinated behavior meets the design goal.

Three populations of processes (agents) contribute to the emerging system functionality. Because each population operates in the shared network environment, the other populations influence its dynamics. For instance, the clients coordinate their server choice through the exchange of scores, but their ability to focus on only a few servers depends on the server population’s ability to identify the emerging intention of the clients and to maintain the server processes on the correct nodes. Figure 4 identifies the main flow of information among the three populations driven by their respective dynamics and linked by the occurrence of successful or failed utilization events – requests from clients to servers.

A common feature of the server activation learning and client preference learning in our scheme is the combined reinforcement and decay of a critical decision parameter (the countdown on the server; pro and con scores on the server scorecards maintained by clients). Elsewhere [14] we describe this sort of process as “pheromone learning,” because it combines two of the hallmarks of insect pheromones: periodic deposits, and constant background evaporation. Pheromone learning can be viewed as reversing the traditional approach to truth maintenance. Rather than maintaining any knowledge until it is proven wrong, we begin to remove knowledge as soon as it is no longer reinforced. This approach is successfully demonstrated in natural agent systems, such as ant colonies, where information stored in pheromones begins to evaporate as soon as it is laid down.

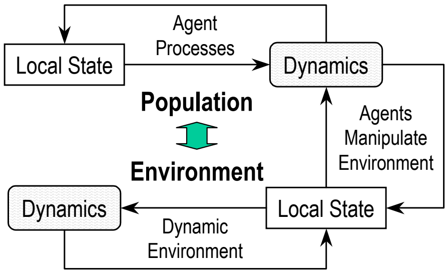

Fig. 3. Stigmergic Coordination (general schema)

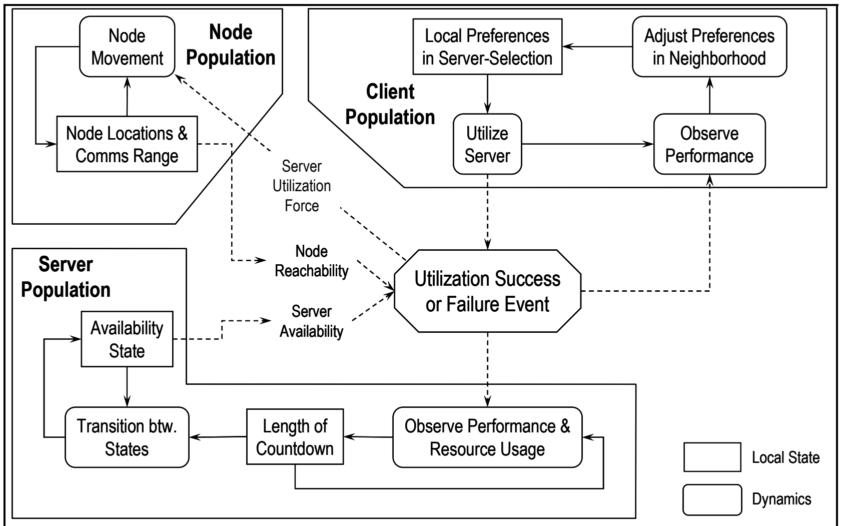

Fig. 4. Stigmergic Coordination in MANET's

4. Performance Analysis

As engineers, we need not only to conceive innovative architectures to address challenging real-world problems, but also to analyze these architectures to determine their performance as a function of deployment conditions. Such analysis requires three elements: a baseline against which to compare the performance of the innovation, a set of metrics to make this comparison, and experiments to apply the metrics to the new system.

4.1 Baseline

Baselines for performance evaluation can be of two kinds. Sometimes we have performance data for a conventional system and wish to show how our system compares with it (a relative evaluation). In other cases we have an upper bound on performance, a bound that may not be achievable in practice, but that shows how close to the theoretically best performance our solution (or any other) comes (an absolute evaluation).

In the case of MANET's, we can define a global solution that provides the highest possible request-success rate for the clients. We ignore the desire to preserve battery power and let all server nodes execute the server process at all times (maximum server availability). We use global knowledge (requiring very large bandwidth) to determine for a client that wants to send a request, which available server nodes are currently in range (path exists), and then we select the recipient of the request randomly from this set.

This solution formally avoids sending requests to servers that are out of reach, whose node is currently down, or whose server process is currently not executing. But its resource requirements are too large to meet the severe constraints of the application domain (ad-hoc mobile wireless network among battery-powered nodes). Also, from a more programmatic point of view, this solution does not demonstrate emergent cognition, since the complexity of the individual node (client) is as high as the system-level complexity. Nevertheless, this solution provides us with a performance and resource-usage baseline against which we measure our local approach in the demonstration.

4.2 Metrics

We focus our attention on two metrics of a system under a particular set of deployment constraints: resource gain and performance loss. Both are ratios comparing a key system-level feature with the baseline.

Resource gain describes the percentage of servers that our mechanism keeps on standby, that would be running and burning power in the baseline. The total number of servers is a constant in this scenario, and all of them are running in the baseline. So resource gain is directly proportional to the total number of servers on standby.

Performance loss measures the failure of service events in our mechanism compared with the baseline. Let

N = total number of service requests

N_b = total number of requests satisfied by the baseline;

N_t = total number of requests satisfied by the test system.

Since the baseline is the best possible in any given circumstance, $N_t \leq N_b \leq N$. Performance loss is defined as $(N_b - N_t)/(N - N_b)$. Unlike resource gain, performance loss is compared against a changing baseline, since N_b varies with system configuration, so we also track raw performance of our scheme.

4.3 Comparison with the Global Solution

With a baseline and metrics in hand, we can explore the performance of our system. The following discussion is meant to be exemplary, not exhaustive. We explore the variation in metrics as a function of three network characteristics: the degree of connectivity, the dynamics of individual servers, and the overall demand from the clients. Error bars in the plots are at ± 1 standard deviation, adjusted to avoid unphysical values (e.g., probabilities outside of [0,1]).

4.3.1 Configuration

Our experiments use a population of 100 nodes, of which 25 can serve as servers. They are initially distributed randomly in an arena sized 100 x 100, so the average area per node is 100, with radius ~5.6, and a mean internode separation on the order of 11. At each time step, several parameters determine the dynamics of the system.

- Range is a measure of the communications range of the nodes, in the same units that define the dimensions of the virtual world within which the nodes are distributed. The default setting is 15, which is greater than the mean internode separation of 11.

- DownProb (p_d) is the probability that a node will go out of service due to failure. The default setting is 0.02.
- UpProb (p_u) is the probability that a failed node will resume operation. The default setting is 0.90.
- UtilizationRate is the probability that a given node requests service. The default setting is 0.50.
- NodeMovementPolicy can be either directed (in which case servers and clients implement the algorithm outlined in Section 3.3) or random (in which case the direction of movement is chosen randomly, as a zeroth-order approximation to mission movement).
- ClientStepLength and ServerStepLength define the distance (in the same units as Range) that a node moves in adjusting its location under either movement policy. The defaults are 0.5 and 3.5, respectively.

4.3.2 Impact of Demand

Adaptive schemes such as ours require a steady stream of information about the environment, which in our case is provided by the success or failure of service requests. When service requests are at a very low level, the system cannot adapt effectively, reflected in the performance changes. Figure 5 shows the impact of changing utilization. All other parameters are fixed at their default values.

The mean value of raw performance increases with utilization, and performance loss decreases, but the error bars show that these changes are swamped by noise. It is important to note that the variance is much greater for low utilization (10%) than for the higher levels. At low utilization, the algorithm does not get sufficient information to make useful decisions, but at higher utilization levels, its behavior converges.

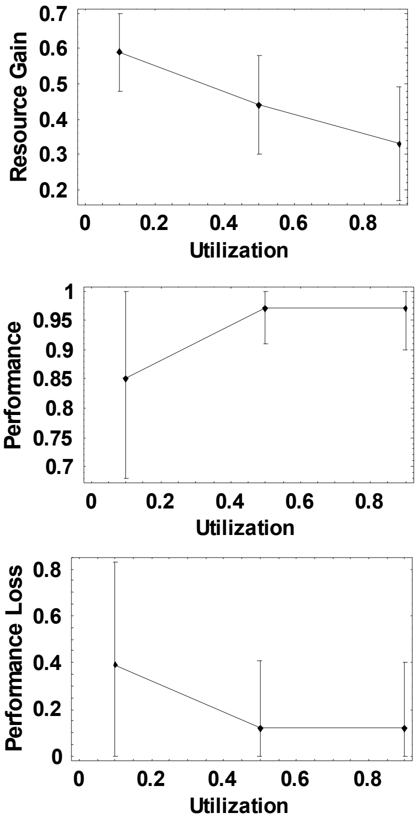

Fig. 5. Impact of Varying Utilization

Resource gain drops with increased utilization. The higher message traffic stimulates servers to remain awake that would otherwise go to sleep, lowering the resource benefits. The system successfully adapts the number of active servers to changes in the overall message load.

This experiment is the basis for fixing utilization in subsequent experiments at 50%, a level that provides sufficient information to enable the algorithm to converge, while still making it worthwhile for servers to sleep.

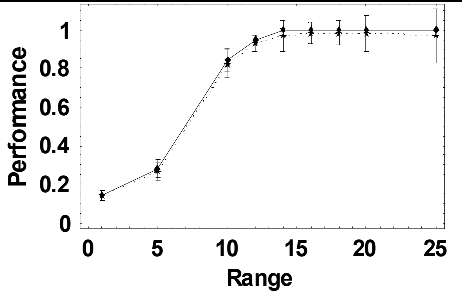

Fig. 6. Performance as Function of Range in Baseline (solid) and adaptive algorithm (dashed).

4.3.3 Impact of Network Connectivity

A critical characteristic of a MANET is the range of the radios that provide the communication links. Figure 6 shows the raw performance of our scheme and of the baseline, using random node movement. We hold all parameters at their default settings and vary NodeRadius. As expected, performance increases monotonically with radio range. Importantly, the performance of our adaptive algorithm is indistinguishable from the baseline.

We have found that the directed movement of servers toward selected clients is not effective as currently implemented, as shown in Figure 7. More realistic movement models, suggested below, might yield a different outcome. We do not report further results with directed movement.

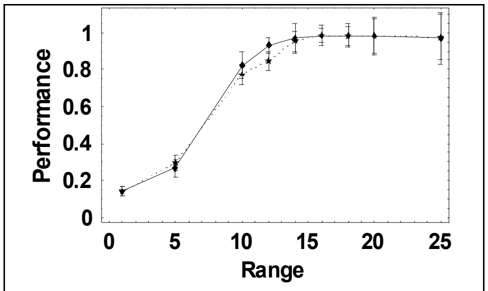

Fig. 7. Performance as Function of Range, with (solid) and without (dashed)t Location Learning

While the performance is comparable between our mechanism and the baseline, resource gain is not (Figure 8; by definition, gain for the baseline is 0). Clearly, our mechanism improves resource utilization significantly without impacting performance, compared with a best-case solution that may not be implementable.

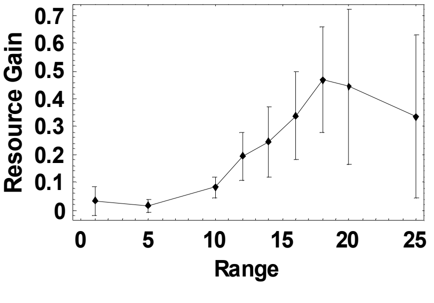

Fig. 8. Resource Gain as Function of Range

4.3.4 Impact of Network Dynamics

Figure 9 shows how resource gain, raw performance, and performance loss vary as a function of server dynamics. Utilization is set at 0.5 and range at 15. For each metric, the figure shows four cases.

p_d = 0.1, p_u = 0.9.—This configuration reflects highly reliable servers that seldom go down and are quickly repaired, a "best case" scenario from the operational point of view.

p_d = 0.9, p_u = 0.1.—This configuration reflects highly unstable servers that take a long time to repair, a "worst case" scenario.

p_d = p_u = 0.5.—This configuration reflects symmetric mean-time-to-failure (MTTF) and mean-time-to-repair (MTTR) with a moderate value.

p_d = p_u = 0.1.—This configuration reflects symmetric MTTF and MTTR with a low value.

Consider first performance and performance loss. As might be expected, performance is good in the best case, bad in the worst case, and intermediate with symmetric MTTF and MTTR. Interestingly, performance is not significantly different between the two symmetric cases. The mean values of performance loss follow the same general trend, though wide variances make the differences less significant. Performance loss is least in the best case, when the system can reliably learn which servers to employ.

Our algorithm shows resource gain in all configurations, though with high variances in both worst and best case conditions. (It is important to recognize that wide variances that reach 0 do not mean that the benefit is not statistically significant. Resource gain for the base case is identically zero by definition. Any resource gain produced by the adaptive algorithm is a real benefit, since it reflects power savings. The high variance simply means that the variation in this savings from one cycle to another is subject to wide swings, but the integral over these swings, reflecting total power saved, is unambiguously positive.) The mean resource gain in these two cases is almost the same, reflecting the benefits of adaptivity in coping with unstable systems.

In the case of equal and moderate failure and recover probabilities, there is little resource gain over the baseline. This configuration changes so frequently

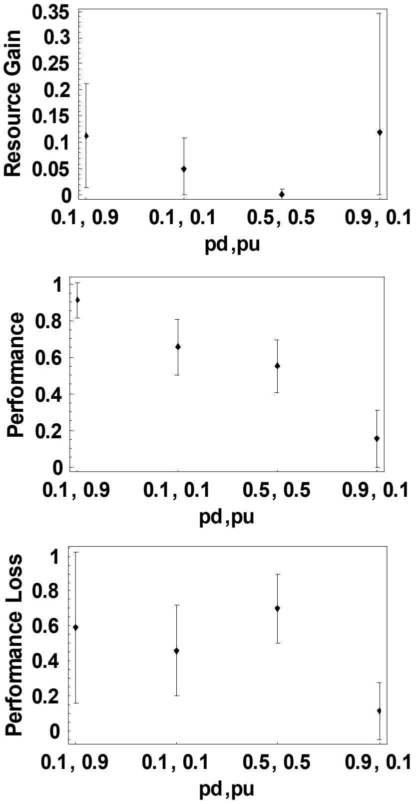

Fig. 9. Impact of Server Dynamics

that our learning process does not have time to adapt to the changed environment.

5. Comparison with Previous Research

Our system addresses all three aspects of the server management problem: given the current topology of the network determined by node locations, communications ranges and node availability, decide

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

MANET's are an active area of current research, but until recently the focus of the MANET community has been on issues such as routing [9], access control [6], and security [19]. These are important issues, but largely orthogonal to the question of server management.

Recent research considers one aspect of the server management problem in MANET's, the second of our three questions (known as the service discovery problem). Efforts in this area can be divided into two groups.

Our approach is most similar to decentralized techniques such as flooding, swamping, and name-dropping (usefully reviewed in [7]), which all involve sharing knowledge of accessible services among adjacent nodes. The novelty of our approach lies in the use and propagation, not only of pointers to servers, but of scorecards to guide in selecting the server that will be tried on a given attempt. The probabilistic nature of our selection process adds robustness in the face of dynamic change. Conventional sharing schemes explore such options as whether to share with all neighbors or only with a subset at each cycle, and these options are reasonable enhancements to explore with our mechanisms.

More recent work on service discovery, and that devoted specifically to MANET's, uses service brokers to maintain directories of available servers [5, 8, 10, 18]. Highly dynamic environments (such as those encountered in military applications) can frustrate directory-based schemes.

In addition to providing a robust decentralized solution to the widely studied service discovery mechanism, our approach offers an integrated solution to the less explored problems of server activation and location. By addressing all three problems with a single set of mechanisms, we reduce the complexity of the overall system and facilitate making necessary trade-offs against different operating options, compared with approaches that piece together independent solutions to each problem.

6. Discussion and Conclusion

Swarming fine-grained agents offer an effective approach to real-time control of mobile ad-hoc networks. Our experiments show that we can reduce the resource requirements for servers in a MANET without significantly diminishing the system's performance, relative to an optimistic and probably unachievable baseline. Our experiments suggest two guidelines for when such approaches are applicable.

1. Because we rely on feedback from client attempts to access service as our source of information about the environment, the system requires a reasonable level of utilization. It is not appropriate for systems that are rarely utilized, but that must work appropriately when they are occasionally activated. However, the algorithms do adapt appropriately over a wide range of utilization levels.
2. Our methods work well when either failure probability or repair probability is low, since these characteristics lead to fairly stable server populations. When the probabilities of server failure and server repair are both high, the world changes too rapidly for our agents' pheromone learning mechanisms, and system efficiency (as measured by resource gain) suffers.

The system described here is a highly simplified initial model of the MANET domain. We hope to explore several extensions of this domain.

- This model assumes that the movements of all vehicles are equally constrained by the same movement policy, either random (to simulate mission movement) or directed (to improve communications effectiveness). Using task allocation mechanisms similar to those we explored in [13], it would be interesting to examine fleets in which different platforms follow different movement policies, enabling some platforms learn to specialize as communication relays, and leaving other platforms more latitude for their mission-oriented tasks.
- It will also be important to examine the effect of more realistic models of mission-related movement, instead of the surrogate of random motion used here. For example, we might explore space-filling behavior to model exploratory missions, or divergence and reforming of the fleet as it moves in a general geographical direction.
- The preliminary results reported here do not show any benefit to directed movement of servers with respect to their emerging client populations. This result is counter-intuitive, and we wish to do further analysis and experimentation to understand whether and under what circumstances servers can improve system performance by directed movement.
- The breakdown of our system at low utilization levels may be mitigated in part if we make use of the "heartbeat" signals that communication nodes routinely exchange to monitor their connectivity, and we wish to explore ways that these signals can contribute to the service provider problem.
- Service provision is only one of many functions that a MANET can provide. We believe our mechanisms hold far more general promise, and look forward to expanding them into a general scheme for MANET management.

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. We discuss this aspect of the problem elsewhere [3].

6. Acknowledgments

This work is supported in part by the DARPA Knowledge-Plane seedling study, contract N00014-03-M-0252 to Altarum, under DARPA PM Christopher Ramming. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

7. References

[1] G. Aggelou. Mobile Ad Hoc Network (MANET) Papers. vol. 2004, pages Web page, University of Surrey, Guildford, UK, 1999. Available at http://www.ee.surrey.ac.uk/Personal/G.Aggelou/MANET_PUBLICATIONS.html.

[2] S. Brueckner. *Return from the Ant: Synthetic Ecosystems for Manufacturing Control*. Dr.rer.nat. Thesis at Humboldt University Berlin, Department of Computer Science, 2000. Available at http://dochost.rz.hu-berlin.de/dissertationen/brueckner-sven-2000-06-21/PDF/Brueckner.pdf.

[3] S. Brueckner and H. V. D. Parunak. Self-Organizing MANET Management. In *Proceedings of Workshop on Engineering Self-Organising Agents (AAMAS 2003)*, pages (in press), Springer, 2003.

[4] S. A. Brueckner and H. V. D. Parunak. Swarming Agents for Distributed Pattern Detection and Classification. In *Proceedings of Workshop on Ubiquitous Computing, AAMAS 2002*, 2002. Available at http://www.altarum.net/~vparunak/PatternDetection01.pdf.

[5] L. Cheng. Service Advertisement and Discovery in Mobile Ad hoc Networks. In *Proceedings of Workshop on Ad hoc Communications and Collaboration in Ubiquitous Computing Environments (ACM CSCW 2002)*, ACM, 2002. Available at http://www.cs.uoregon.edu/research/wearables/cscw2002ws/papers/Cheng.pdf.

[6] Z. J. Haas, J. Deng, and S. Tabrizi. Collision-Free Medium Access Control Scheme for Ad-Hoc Networks. In *Proceedings of IEEE MILCOM'99*, IEEE, 1999. Available at http://wnl.ece.cornell.edu/Publications/milcom99_mac.ps.

[7] M. Harchol-Balter, T. Leighton, and D. Lewin. Resource Discovery in Distributed Networks. In *Proceedings of 18th Annual ACM-SIGACT/SIGOPS Symposium on Principles of Distributed Computing*, pages 229-238, ACM, 1999.

[8] U. C. Kozat and L. Tassiulas. Network Layer Support for Service Discovery in Mobile Ad Hoc Networks. In *Proceedings of IEEE INFOCOM 2003*, IEEE, 2003. Available at http://www.ieee-infocom.org/2003/papers/48_02.PDF.

[9] S. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia. A performance comparison study of ad hoc wireless multicast protocols. In *Proceedings of IEEE Infocom'2000*, pages 565–574, IEEE, 2000.

[10] J. Liu, Q. Zhang, W. Zhu, and B. Li. Service Locating for Large-Scale Mobile Ad Hoc Network. *International Journal of Wireless Information Networks*, 10(1 (January)):33-40, 2003. Available at http://research.microsoft.com/asia/dload_files/group/wireless/2002p/IJWIN.pdf.

[11] H. V. D. Parunak. 'Go to the Ant': Engineering Principles from Natural Agent Systems. *Annals of Operations Research*, 75:69-101, 1997. Available at http://www.altarum.net/~vparunak/gotoant.pdf.

[12] H. V. D. Parunak. Making Swarming Happen. In *Proceedings of Swarming and Network-Enabled C4ISR*, ASD C3I, 2003. Available at http://www.altarum.net/~vparunak/MSH03.pdf.

[13] H. V. D. Parunak and S. Brueckner. Swarming Coordination of Multiple UAV's for Collaborative Sensing. In *Proceedings of Second AIAA "Unmanned Unlimited" Systems, Technologies, and Operations Conference*, AIAA, 2003. Available at http://www.altarum.net/~vparunak/AIAA03.pdf.

[14] H. V. D. Parunak, S. Brueckner, R. Matthews, and J. Sauter. How to Calm Hyperactive Agents. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 2003)*, pages 1092-1093, 2003. Available at http://www.altarum.net/~vparunak/AAMAS03Ritalin.pdf.

[15] 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, 2004. Available at http://www.altarum.net/~vparunak/MSEAS03.pdf.

[16] H. V. D. Parunak, M. Purcell, and R. O'Connell. Digital Pheromones for Autonomous Coordination of Swarming UAV's. In *Proceedings of First AIAA Unmanned Aerospace Vehicles, Systems,Technologies, and Operations Conference*, AIAA, 2002. Available at www.altarum.net/~vparunak/AIAA02.pdf.

[17] P. Weinstein, H. V. D. Parunak, P. Chiusano, and S. Brueckner. Agents Swarming in Semantic Spaces to Corroborate Hypotheses. In *Proceedings of AAMAS 2004*, pages (forthcoming), 2004. Available at http://www.altarum.net/~vparunak/AAMAS04AntCAFE.pdf.

[18] J. Wu and M. Zitterbart. Service Awareness and its Challenges in Mobile Ad Hoc Networks. In *Proceedings of Workshop der Informatik 2001: Mobile Communication over Wireless LAN*, 2001. Available at http://www.iponair.de/publications/Wu-Informatik01.pdf.

[19] L. Zhou and Z. J. Haas. Securing Ad Hoc Networks. *IEEE Network Magazine*, 13(6 (November-December)), 1999. Available at http://wnl.ece.cornell.edu/Publications/network99.ps.