# Emergent Distribution of Operating System Services in Wireless Ad Hoc Networks

Peter Janacik and Tales Heimfarth

Heinz Nixdorf Institute, University of Paderborn
Fuerstenallee 11, 33102 Paderborn, Germany
{pjanacik, tales}@uni-paderborn.de

**Abstract.** Despite the advances in wireless, energy-constrained ad hoc networks, there are still many challenges given the limited capabilities of the current hardware. Therefore, our aim is to develop a lightweight, yet powerful operating system (OS) for these networks. We reject the brute force method of provisioning all necessary OS services at each node of the system. Instead, our approach aims to distribute the set of requested OS services over the network to reduce and balance load, improve quality of service, increase fairness and predictability. To limit the burden imposed on the network by the service distribution mechanism, only a subset of nodes, the coordinators, chosen by an underlying state-of-the-art topology control, are concerned with this task. Coordinators observe the state of nodes and OS services within their one-hop vicinity, i.e. their decision area, incorporating different aspects, such as energy, utilisation, or available resources in their decisions. Although each coordinator acquires information and triggers migrations of service states only locally within its decision area, a global-level result emerges, as decision areas naturally overlap. In this manner, an increased amount of work load e.g. in one decision area "floats" to the surrounding decision areas attracted by better conditions. In ns-2 simulations we demonstrate that the mechanism of emergence, which produces many fascinating results in natural systems, can successfully be applied in artificial systems to considerably increase the efficiency and quality of OS service distribution.

## 1 Introduction

Given current hardware limitations of wireless nodes, e.g. commercial off-the-shelf sensor nodes (see [1]), there are severe restrictions on the software executed on them. For the same reason, operating systems (OS) for this type of nodes, like *TinyOS* [2], do not provide the means to handle more complex applications. To cope with these challenges, we use the paradigm of OS service distribution within our lightweight, distributed operating system *NanoOS* [3]. The OS consists of different services such as scheduling, synchronisation, time, etc. Traditional OS offer the set of all needed services at every node of the system resulting in excessive resource waste. Moreover, this limits the possible number

*Please use the following format when citing this chapter:*

Janacik, P., Heimfarth, T., 2006, in IFIP International Federation for Information Processing, Volume 216, Biologically Inspired Cooperative Computing, eds. Pan, Y., Rammig, F., Schmeck, H., Solar, M., (Boston: Springer), pp. 199–208.

of OS services utilised at one node at the same time. In contrast, our approach distributes the set of needed OS services over different nodes leading to a lower per-node load, a greater amount of possible service types, and the option of load adjustment. In particular, a distribution service observes the network and initiates migrations of OS service states (associated with service requestors) to achieve the following aims: *load balancing*, i.e. the uniform distribution of load over available nodes and services, *fairness*, i.e. the equal treatment of service requestors, *quality of service*, i.e. short answering times, and *predictability* of service quality. Providing these properties is a *global*-level aim, which is achieved solely from numerous interactions among *lower*-level components, i.e. the nodes. Moreover, the rules specifying these interactions are executed using only *local*, i.e. one-hop, information without reference to the *global* pattern (or aim). The *emergent* property (as defined in [4]) of our system is of utter importance in the scenario of volatile, energy-constrained networks: it translates to a highly increased amount of robustness, resilience, and a considerably lower communication overhead.

To lower the burden imposed on the network, our approach makes a subset of nodes, the *coordinators*, responsible for service distribution. This set is chosen dynamically by an underlying state-of-the-art topology control (such as [5, 6, 7]), so that each node has at least one coordinator in one-hop distance. Coordinators run a *distribution service* that is responsible for observing the state of the system within their one-hop neighbourhood, i.e. their *decision area*, and for deciding on the migration of OS service states. As already discussed above, at first glance, the mechanism for service distribution is local. But given the natural overlap of decision areas, there is also an inter-decision area migration. This way, load can "float" to neighbouring areas, so that a global-level result emerges. Using ns-2 [8] simulations, we demonstrate the considerable improvements in terms of the above-defined aims provided by our approach. We are aware that reducing the distance between OS service requestors and providers, efficient service discovery, or failure handling are also crucial in wireless, energy-constrained ad hoc networks. These topics are however beyond the scope of this document and will therefore be addressed in other publications.

This paper is organised as follows: Section 2 presents the state-of-the-art, while Section 3 subsequently describes the proposed emergent distribution of OS services. Section 4 then presents the results of our simulations. Finally, Section 5 ends this paper with brief concluding remarks.

## 2 State-of-the-Art

Current ad hoc or sensor network node hardware imposes severe restrictions on the software executed on top of it. Therefore, *TinyOS* [2] e.g. tries to solve this problem with its extremely small footprint. But since all components of a TinyOS instance have to fit into one node, its functionality is severely limited, so that it cannot cope with more complex applications. The *MagnetOS* approach

[9], as another example, is very different from most OS: its aim is to offer a single-system image of a unified Java virtual machine (JVM) across nodes. Migration of objects in MagnetOS is carried out over one or multiple hops in the direction of the greatest communication, reducing the distance between call-initiators and -receivers. Our work, however, has different aims: improving load balancing, fairness, quality of service, and predictability. In the related field of dynamic distributed scheduling algorithms, there has also been research on the reduction of communication cost and load balancing. However, such approaches like [10, 11] were developed for static networks of UNIX workstations and are not suited for mobile, volatile, resource- and energy-constrained networks. Further, they impose the burden of service distribution on all workstations in a network.

## 3 Emergent Distribution of OS Services

After providing an outline of our approach in the introduction and reviewing the state-of-the-art, this section describes the system components and how division of labour between nodes is employed. Subsequently, the main part of this section concentrates on migration source and target determination.

### 3.1 System Components

We assume a wireless network consisting of resource- and energy-constrained, mobile hardware nodes. Our OS, composed of services, and applications, composed of tasks, run on top of it. In addition to the functionality of traditional OS, our OS provides an *uniform system call environment* across the mobile nodes and further services like a *distribution service*, observing the system state and initiating migrations, or distributed event, memory, and synchronisation services. As depicted in Figure 2 (a), *OS services* and *application tasks* are subtypes of the abstract *processing entity*. An OS service maintains states associated with each of its requestors, which are sharing it and may reside on different, remote nodes. Services may act as both, service requestors and providers, while tasks only act as requestors.

### 3.2 Division of Labour between Nodes

In order to reduce the burden imposed on the network by the mechanism of service distribution and to enable the fusion of relevant system data, we assign the task of service distribution only to a subset of nodes, called *coordinators*. This subset, created by a state-of-the-art topology control (such as [5, 7] or our work from [6]), should consist of a low number of nodes, while ensuring that each node has at least one coordinator in one-hop communication distance (as depicted in Figure 1 (a)). Further, this implies that the number of coordinators scales with the density and number of nodes. The idea of our work is that each coordinator runs a distribution service that monitors the coordinator's *decision*

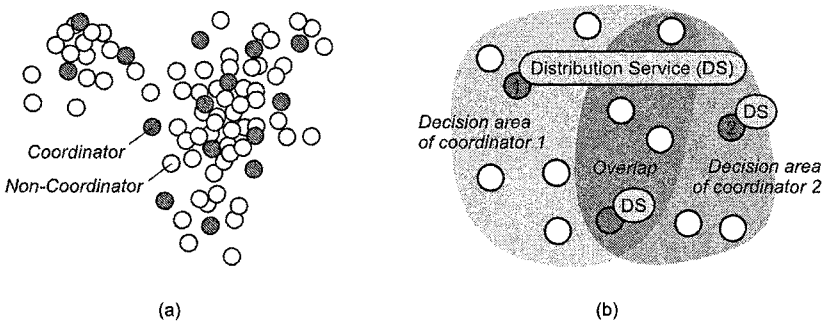


**Fig. 1.** Division of labour between nodes. (a) Different node types. (b) Decision areas of coordinators and distribution service placement.

*area*, which contains all non-coordinators in one-hop distance. Figure 1 (b) shows the overlapping decision areas of coordinators 1 and 2. We assume the amount of overlap to be a tuneable parameter of topology control. Coordinator status changes take place as *reaction to changes* of the environment (e.g. node density), but also *over time*, so that nodes "take turns" being coordinators balancing the burden of service distribution.

### 3.3 State Migration Source and Target Determination

*Migration*, initiated by the distribution service, consists of the migration of states, which are associated with the requestors of services, between existing OS services, but also the migration of states to newly-started OS services. Figure 2 (b) illustrates a typical scenario, where migration is applied. The OS service running at node 4 is overburdened, as indicated by the service request queue length. At the same time, the OS services at nodes 3 and 5 are almost idle increasing the overall execution overhead. To improve the configuration, some OS service states from the service at node 4 would be migrated to the service at node 6. The services at nodes 3 and 5 would be fused, on the other hand, by migrating all states from one to the other service. To be more concrete, the migration decision policy (intuitively speaking) has the following main operational goals: First, prevent too long service request queues; second, disburden services at nodes, whose remaining energy level is considerably below the average energy level of nodes in the decision area of the coordinator; moreover, avoid the execution of services with very short queues, since these do not justify the associated overhead and hold resources which may prevent the start of new services. To enable migration decisions, every distribution service is provided with the information utilised in the descriptions below from all nodes and services in its coordinator's decision area (by underlying protocols).

In our model, we assume the utilisation of a service to be indicated by the *average length of its queue for pending service requests* in terms of processing time needed. For the sake of simplicity, we will only refer to it as (service

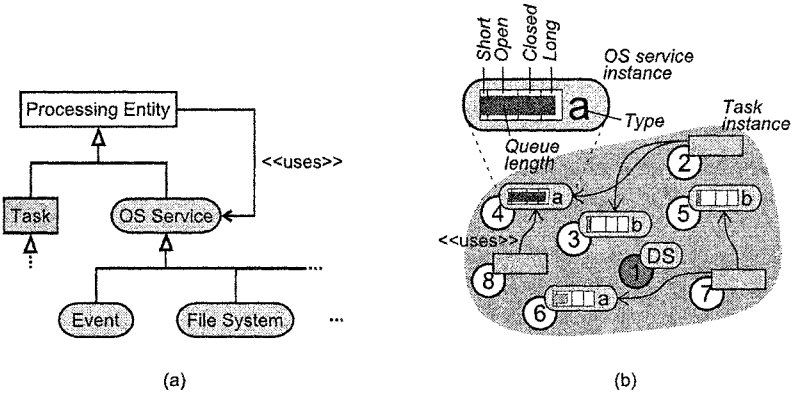


**Fig. 2.** Processing entities. (a) Relationship between processing entity and its subclasses. (b) Distribution of processing entities over the network, illustrating a problem scenario. Key: DS—distribution service

request) queue length below. In order to enable decisions based on the degree of utilisation, our approach distinguishes the following categories of service request queue lengths (see also Figure 2 (b)):

Short: Low utilisation
Open: Fair utilisation
Closed: Fair or slightly higher utilisation, rejecting new requestors
Long: Critically high utilisation, rejecting new requestors

There are different *priority classes* to characterise the *severity* of the problem at the migration source (according to the policy described above): (1) Long queue; (2) queue open or closed and energy low at hosting node; (3) queue short. For an additional, more fine granular ranking within priority classes, we use the following *OS service fitness metric* that rates the service incorporating the hosting node, e.g. whether the service load is appropriate and the host's resources are not exhausted: $M_{service\_fitness} = \omega_{CPU} \cdot M_{CPU} + \omega_{mem} \cdot M_{mem} + \omega_{ql} \cdot M_{ql} + \omega_E \cdot M_E$. It uses metrics taking into account CPU and memory utilisation[1] $M_{\{CPU,mem\}} = \frac{avail\_\{CPU,mem\}}{max\_\{CPU,mem\}}$ describing the proportion of available to maximum resources, the queue length metric $M_{ql} = 1 - \min(\frac{ql}{ql_{long\_min}}, 1)$ reflecting the relation of the actual queue length to the minimum long queue length, and the energy metric

$$M_E = \begin{cases} 1 & \text{if} \quad E_{host} \geq E_{avrg\_decision\_area} \\ 1 - \frac{E_{avrg\_decision\_area} - E_{host}}{E_{avrg\_decision\_area}} & \text{else} \end{cases}$$

describing the proportion of remaining energy at the host to the average amount of remaining energy at nodes in the decision area. $\omega_{CPU}$, $\omega_{mem}$, $\omega_{ql}$, and $\omega_E$ are

[1] Different members of sets exclude each other in the following formula.

weights for the corresponding metrics, such that $\omega_{CPU} + \omega_{mem} + \omega_{ql} + \omega_E = 1$. They can be adjusted in order to reflect characteristics of a certain hardware type, e.g. $\omega_{mem}$ can be increased if memory is the more valuable resource. Moreover, the above functions use $avail\_\{CPU, mem\}$, reflecting the amount of available CPU and memory resources, $max\_\{CPU, mem\}$, describing the maximum available corresponding resources at a node. $ql_{long\_min}$ represents the minimum queue length for the "long" category and $ql$ the actual service request queue length. $E_{avrg\_decision\_area}$ contains the average of remaining energy levels in the decision area, whereas $E_{host}$ describes the remaining energy level of the service host.

The service in the *highest* priority class with the *lowest* $M_{service\_fitness}$ ranking is chosen first as *migration* source. In order to reduce interference of overlapping decision areas, only a distribution service at a coordinator, which is connected to a non-coordinator with the best link from all links connecting it to surrounding coordinators, may choose a service from such a non-coordinator as migration source.

Finding a *migration target* works similar to the migration source finding process, but using the following *priority classes*: (1) Open queue and sufficient energy at hosting node; (2) short queue; (3) no service running at hosting node. Priority class 3 is only an option, if the queue length of the migration source is above the minimum long queue length, so that near-idle services are not migrated unnecessarily. The decision process proceeds similar to finding a source, except that the whole decision area is taken into account. The service in the *highest* priority class with the *highest* $M_{service\_fitness}$ ranking is chosen first as migration target. After migration initiation, migration source and target are locked, excluding them from the migration process for a specified period of time in order increase the stability of the system.

## 4 Results

We implemented our emergent distribution of OS services and a reference approach using C++ and the ns-2 network simulator [8]. For lower layers, we used our topology control [6] and ant colony-based routing [12]. The reference approach employs a greedy, demand-based OS service placement without service migration and topology control, in conjunction with ad hoc on-demand distance vector (AODV) [8] routing from ns-2. To simulate running processing entities (PE), we specified a set of PE types. For each PE type, a recurring sequence of behaviour items is defined. Each *behaviour item* includes information on its execution duration, CPU, memory, and OS service requirement (a service type or none), as well as, the processing time needed by the required OS service. The assignment of application task instances to PE types and nodes was randomised.

The simulations further employed a 914 MHz Lucent WaveLAN DSSS radio, the two-ray ground reflection model, 80 joules initial energy per node and an

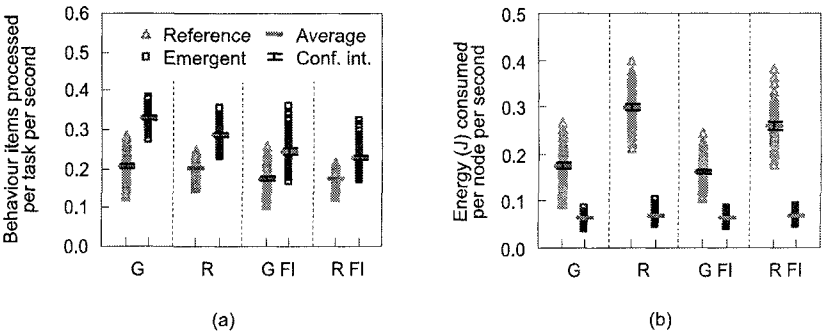


**Fig. 3.** Performance (a) and energy results (b). Key: G—grid, R—RPGM, FI—failure injection

802.11b MAC protocol, provided by ns-2. To cover static and dynamic scenarios we used two settings: (1) A *grid*, 44 nodes, 3 x 15 arrangement, horizontal and vertical distance of $d = 25$ m between nodes, 1000 s simulation time; (2) a *reference point group mobility* (RPGM) model [13], 64 nodes, 4 x 16 groups, logical group centres movement 2–8 m/s based on random walk model [8], 850 x 850 m area, 900 s simulation time. In order to simulate volatile and failure-prone networks, we injected failures during simulation. Our *failure injection* (FI) model employs $k$ failure points (FP) $f_{1,\ldots,k}$. At the start of a run, FP probabilities $p_{f1,\ldots,k}$ are set randomly between $p_{min}$ and $p_{max}$. Next, each node makes a probabilistic decision based on $p_{f_{1,\ldots,k}}$, whether to fail at FP $f_{1,\ldots,k}$. Each FP $f_i$ is associated with a failure time $t_{f_i}$ in ascending temporal order, so that for some $i \in 1,\ldots,k$, $t_{f_i} < t_{f_{i+1}}$ applies. Failing at an FP $f_i$ means for a node that its network interface is out of order between $t_{f_i}$ and $t_{f_{i+1}}$. A failure at the last FP ($i = k$) persists until the end of the simulation. We used two FP ($k = 2$) and failure times ($t_{f_{1,2}}$) at 333 and 667 seconds of simulation time. The minimum FP probability ($p_{min}$) was set to 0, the maximum ($p_{max}$), to 0.5.

The presented figures were obtained using the following settings: 100 runs, reference and emergent approach, grid and RPGM topology, with and without FI; lower and upper bounds of confidence intervals, with probability of error $\alpha = 0.05$, to indicate the significance of the presented results (marks for not applicable or too narrow intervals for a reasonable visualisation are omitted).

### 4.1 Performance and Energy Consumption

Figure 3 (a) depicts the processing speed of both approaches. The emergent approach outperforms the reference approach by a clear margin, which is also an indication for a higher quality of service. The energy consumed by both approaches is depicted in Figure 3 (b). Again, the emergent approach clearly outperforms its reference counterpart. FI does not influence energy consumption considerably, possibly, since the failed nodes do not actively participate

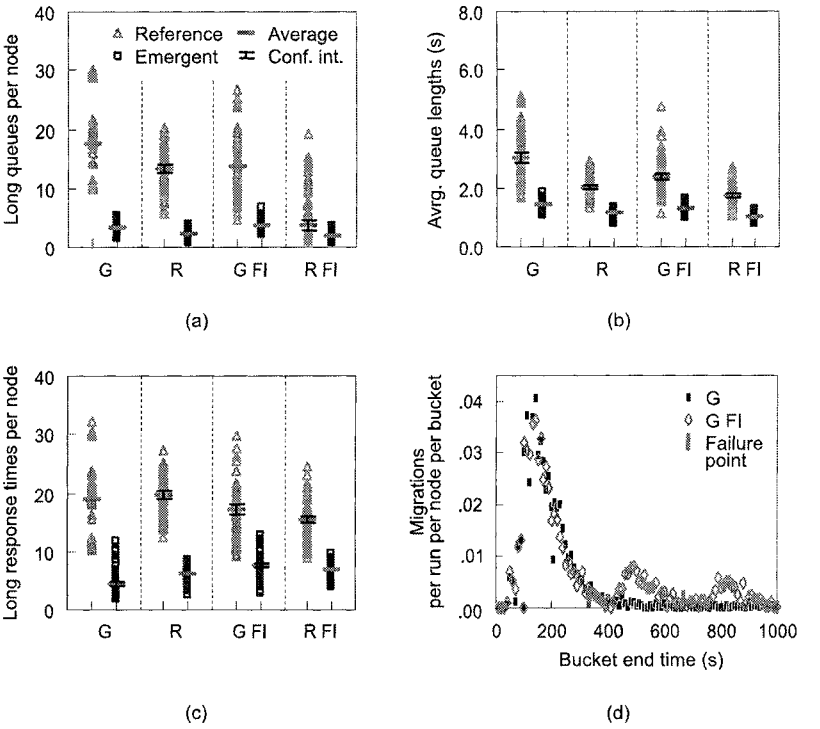


**Fig. 4.** Load balancing, fairness, quality of service, predictability (a–c) and migration behaviour (d) results. Key: G—grid, R—RPGM, FI—failure injection

in communication, thus saving the corresponding amount of energy. Further, the figure clearly demonstrates that the variance of energy consumption of the emergent approach is intelligibly lower, which is also an indicator for a higher predictability.

### 4.2 Load Balancing, Fairness, Quality of Service, Predictability

Quality of service depends on service request queue lengths in general, whereas load balancing, fairness, and predictability depend on the uniform distribution of queue lengths throughout the system. Therefore, our next studies are focused on these indicators. Figure 4 (a) depicts the number of long service request queues (i.e. "which are in the long category") per node in each of the runs. Measurements were taken at service reply issuing. Evidently, the number of long queues in the reference approach is several times higher. Nevertheless, some RPGM FI runs for the reference approach exhibit a very low number of long queue lengths. This is supposedly owed to the reference approach, starting a high number of services that are utilised only to a minimum extent, which increases overhead and prevents new services from being started. In contrast to

the reference approach, the figures for the emergent approach appear to stay constantly low with only little variance, indicating good quality of service and a high predictability. Further, the high amount of long queues in the reference approach hints at the lack of fairness: some service requestors are served significantly slower than others, which greatly affects, and leads to high variances of service requestors' own processing speeds.

Looking at average queue lengths in Figure 4 (b), the conclusions are similar, but the averages of both approaches are more close-by. This could be explained as above by reference service distribution starting a high number of little-utilised services. Emergent service distribution, in contrast, attempts to avoid running near-idle services in order to minimise overhead and provide enough room for the start of new services. This is, however, to its disadvantage in this particular metric. The results for service response times in Figure 4 (c) additionally take into account communication delays and are very similar to the figures in (a).

### 4.3 Reaction Behaviour and Stability of Migration

Figure 4 (d) depicts the migration activity in a grid topology. Migration times encountered are sorted into buckets of 10 seconds. All runs exhibit an initial peak, reflecting initial optimisations. Without FI, migration settles down thereafter, yielding a highly stable solution for the rest of the simulation. If however the need for optimisations is brought about by FI at 333 and 667 seconds, migration reacts swiftly shortly after the occurrences, settling down subsequently leading to a stable solution.

## 5 Conclusion

Within the scope of our efforts to create a lightweight, yet powerful operating system (OS) for wireless, energy-constrained nodes, this paper introduces an efficient method for the distribution of OS services. Our approach only imposes load on a selected subset of nodes, the coordinators. They observe the state of the system locally within their decision areas. Given the natural overlap of these areas, when one decision area suffers e.g. under high load, this load "floats" to the surrounding areas attracted by better conditions. Therefore, although each coordinator acquires information and triggers migrations of service states only locally, there is an emergent global result.

Given the restrictions of current hardware, an efficient distribution method is crucial for our OS. Even more, we strive to provide an OS behaviour that is rather associated with OS which exhibit a much larger footprint: load balancing, fairness, and predictability, combined with a high quality of service. Using ns-2 simulations we show that our approach reduces energy consumption by a significant amount compared to a reference system. Further, quality of service is increased by more than 80 % in most cases, while load balancing is improved by 200 to 400 % exhibiting a low deviation from the average values.

This in particular results in considerably improved fairness and predictability. The state obtained by the proposed mechanism is characterised by stability and swift adjustment to changes in the environment at the global level, emerging from execution of solely local actions based on local information. Concludingly, the observations give yet another piece of evidence that emergence as a mechanism often encountered in nature can be transferred to computer systems while preserving its inherent character.

## References

1. The Scientist and Engineer's Guide to TinyOS Programming. http://ttdp.org/tpg/html/book/book1.htm, accessed January 7, 2006.
2. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Proc. of ACM ASPLOS*, pages 93–104, Cambridge, MA, November 2000.
3. F. J. Rammig, M. Goetz, T. Heimfarth, P. Janacik, and S. Oberthuer. Real-time operating systems for self-coordinating embedded systems. In *Proc. of IEEE ISORC*, Gyeongju, Korea, April 2006. Accepted for publication.
4. S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems.* Princeton Studies in Complexity. Princeton University Press, first edition, 2003.
5. A. Cerpa and D. Estrin. ASCENT: Adaptive self-configuring sensor networks topologies. *IEEE TMC*, 3(3):272–285, 2004.
6. P. Janacik, T. Heimfarth, and F. Rammig. Emergent topology control based on division of labour in ants. In *Proc. of IEEE AINA*, Vienna, Austria, April 2006.
7. F. Ye, G. Zhong, J. Cheng, S. Lu, and L. Zhang. PEAS: A robust energy conserving protocol for long-lived sensor networks. In *Proc. of IEEE ICDCS*, pages 28–37, Providence, RI, May 2003.
8. The network simulator. http://www.isi.edu/nsnam/ns/, accessed July 8, 2005.
9. R. Barr, J. C. Bicket, D. S. Dantas, B. Du, T. W. D. Kim, B. Zhou, and E. Gün Sirer. On the need for system-level support for ad hoc and sensor networks. *ACM SIGOPS OS Review*, 36(2):1–5, April 2002.
10. H.-U. Heiss and M. Schmitz. Decentralized dynamic load balancing: The particles approach. *Information Sciences*, May 1995.
11. C. Lang, M. Trehel, and P. Baptiste. A distributed placement algorithm based on process initiative and on a limited travel. In *Proc. of PDPTA*, 1999.
12. P. Janacik, O. Kao, and U. Rerrer. An approach combining routing and resource sharing in wireless ad hoc networks using swarm-intelligence. In *Proc. of the ACM/IEEE MSWiM*, pages 31–40. CTi Press, 2004.
13. X. Hong, M. Gerla, G. Pei, and C.-C. Chiang. A group mobility model for ad hoc wireless networks. In *Proc. of ACM/IEEE MSWiM*, August 1999.