A Bio-Inspired Architecture for Division of Labour in SANETs

Thomas Halva Labella and Falko Dressler

Autonomic Networking Group
Dept. of Computer Science 7, University of Erlangen-Nuremberg
Martensstr. 3, 91058 Erlangen, Germany
hlabella@ulb.ac.be, dressler@informatik.uni-erlangen.de

Abstract. Division of labour is one of the possible strategies to efficiently exploit the resources of autonomous systems. It is also a phenomenon often observed in animal systems. We show an architecture that implements division of labour in Sensor/Actuator Networks. The way the nodes take their decisions is inspired by ants' foraging behaviour. The preliminary results show that the architecture and the bio-inspired mechanism successfully induce self-organised division of labour in the network. The experiments were run in simulation. We developed a new type of simulator for this purpose. Key features of our work are cross-layer design and exploitation of inter-node interactions. No explicit negotiation between the agents takes place.

1 Introduction

Sensor/Actuator Networks (SANETs) are interesting research objects. They are hybrid systems consisting of networked sensor nodes (also called *motes*) and mobile robots (wheeled, legged, flying, and so on). Robots and motes (henceforth referred to as *agents*) can communicate together via wireless communication for further cooperation, e.g. to achieve a common goal.

SANETs might be used for fire fighting in forests, or to assist a rescue team after an earthquake or similar natural disasters. Motes are deployed to sense the environment. Their output can then be directed to a base host which elaborates the data and sends out commands to the robots. A more challenging issue is to have robots responding autonomously to the motes' output.

Akyildiz et al. [1] cites as unique features of a SANET: node heterogeneity, real-time requirements, different deployment strategies for motes and robots, mobility and co-ordination paradigm—mote/robots and not only mote/sink as typically in Wireless Sensor Networks (WSNs). Research issues include power management, routing, co-ordination algorithms, design, and many other topics.

This work addresses one common problem that arises in autonomous systems: *division of labour*. Suppose that the agents in a SANET have to perform different tasks. Considering the mentioned rescue example, the tasks would probably be: report the state of the environment (temperature, pictures, sounds, etc.) and help victims. There are obviously a number of interesting applications that might fit this scenario. We focus only on coverage [2]. An autonomous SANET has to decide which and how many agents should perform each task, in such a way that tasks do not overlap in the area. This is what we mean by division of labour. The aim of division of labour is to efficiently exploit the available resources of the network.

Not all authors agree on our definition of division of labour. On the contrary most of them call the problem of deciding which and how many agents should perform a task as *task allocation* (we discussed on this in [3]). We think this is mostly due to different interpretations of the word "task". In most cases, it is intended as an operation associated with a starting and a duration (or terminating time). The task "lives" less than the overall system. In our case, a task is something that "lives" as long as the SANET itself. The examples provided in Sec. 3 make this definition clearer.

Our previous work [3, 4] demonstrated how a simple learning mechanism could be useful to improve the efficiency of a group of autonomous robots. The learning, inspired by ants' foraging, was also able to introduce self-organised division of labour in the group. The discussions and results presented in this paper extend our work in the context of SANETs.

We describe here an architecture for division of labour in SANETs. The architecture is based on probabilistic decisions. During the lifetime of the SANET, the agents adapt the probability to execute one task among a given set. The architecture exploits the interactions between agents, but only within a limited range. The local interactions are however enough to induce division of labour at the global level, i.e. to provide a self-organising behaviour [5]. No particular knowledge of the environment or of the other nodes' activity is required. Additionally, the architecture is based on a cross-layer design. Application and network layers are both responsible for the division of labour.

We could not find any related work about division of labour in SANETs. There are however a few about task allocation, where "task" is meant as a short-living set of actions to be performed. Gerkey et al. [6] proposed a market-based task allocation schema. The agents bid to acquire a task based on the estimation of their capabilities. The authors' auction involves the whole SANET. They are aware that this might be a problem and envisage to solve it by running only localised actions. Clustering methods provide the inherent capability of performing operations in a local context. Younis et al. exploited this behaviour for task allocation in WSNs [7]. Batalin et al. [8] used a greedy policy to allocate tasks. Every task is allocated to the best available agent. Low and co-workers [9] used a bio-inspired task allocation mechanism. It is based on the threshold model [10] and is used to allocate the task of tracking objects in the network. As in the architecture we describe below, their agents adapt during the network lifetime. Jesi et al. [11] also use an adaptive threshold model for the purpose of selecting superpeers in an overlay network. They employ a different formula to adapt the system behaviour. It will be part of future work to compare the different formulae in detail.

We tested our architecture in a proof-of-concept using a simulated environment. There are still some minor things that have to be improved before porting to real hardware. We took particular care in designing a reliable simulator. Our previous experience with similar simulations make us confident in a successful port on real hardware and to more complex environments in the future.

Section 2 briefly describes the simulator that we developed for our experiments. Sections 3 and 4 describe the application layer (for task selection) and the network layer (for message routing) of our architecture. Section 5 shows the results of the experiments, and Section 6 draws our conclusions.

2 Simulation

Good simulation tools can be found for WSNs as well as for robots. The former focus on the simulation of the communication layers and environments, the latter on the physical environment and how the robots perceive it. Something that does both is, to the best of our knowledge, missing.

On the side of network simulations, OMNeT++[1] is becoming more and more popular. OMNeT++ is a modular discrete event simulator. It can be extended by means of additional components. The Mobility Framework (MF) simulates mobile nodes that communicate through wireless communication. The environment in which the nodes move is quite simple. Nodes are simulated as points in a rectangular two-dimensional area. Most of the simulations of wireless networks can be reliable and effective also in such a simple environment.

On the side of robot simulation, a number of libraries have appeared for the simulation of rigid bodies. Open Dynamics Engine (ODE)[2] is one of them. A body is defined by its position, mass, velocity, orientation, and momentum of inertia. More bodies can be connected through different types of joints. Joints constrain the movement of one body w.r.t. another. Each body might also be given a shape. Shapes are used by the library to detect when and where two bodies collide. ODE solves the equations of dynamics to obtain the trajectory of the bodies.

We extended OMNeT++ to include in it the rigid-body simulation provided by ODE. We called the resulting simulator BARAKA. Unfortunately, there is not enough place here to describe BARAKA in details. The interested reader can refer to [12]. Fig. 1 shows the outcome of our hybrid simulator. BARAKA is based on he principles set forth by Jakobi et al. [13]. These principles make more likely a successful port of the algorithms on real hardware.

3 Task Selection

We first describe the application layer, which is in charge of selecting the tasks to perform. Agents have the capability to perceive whether their action is successful

[1] `http://www.omnetpp.org/`
[2] `http://www.ode.org`

or not, either by directly sensing the environment, or by receiving a feedback from other nodes (we discuss more on this in Sec. 4.3). Before choosing a new task, the application layer adapts its parameter on the base of the outcome of the previous task.

Robots and motes know *a priori* the list of possible tasks that they can perform. They have generally different sets of tasks. In our experiments we used four tasks both for robots and motes, therefore we can indicate the set of tasks as $T_{\text{agent}} = \{\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4\}$. The robots' and motes' four tasks are described below.

Each agent k associates a task to a real number τ_{i}^{k}, with $i \in T_{\text{agent}}$. At the moment of selecting a task to perform, the agent chooses randomly between the tasks. The probability to choose task i is

$$P(i) = \frac{(\tau_{\mathrm{i}}^{k})^{\beta_{\text{task}}}}{\sum\limits_{k \in T_{\text{agent}}} (\tau_{\mathrm{k}}^{k})^{\beta_{\text{task}}}} \;, \tag{1}$$

with $\beta_{\text{task}} \geq 1$ (in the experiments of Sec. 5, $\beta_{\text{task}} = 3$). This equation is like the one used to model how ants choose one path among the several that bring to a food source [14]. In the original formulation, τ_{i}^{k} was the concentration of pheromone on path i. The parameter β was introduced to increase the exploitation of good paths.

The agent initialises $\tau_{\mathrm{i}}^{k} = \tau_{\text{init}}$, $\forall i \in T_{\text{agent}}$. If the agent is successful in performing task i, then

$$\tau_{\mathrm{i}}^{k} = min\{\tau_{\max}, \tau_{\mathrm{i}}^{k} + \Delta\tau\} \tag{2}$$

and

$$\tau_{\mathrm{i}}^{k} = max\{\tau_{\min}, \tau_{\mathrm{i}}^{k} - \Delta\tau\} \tag{3}$$

if it is unsuccessful.

The agents in our experiments have four tasks. For three of them, the behaviours of robots and sensors are the same. We suppose the agents are used to sense the environment and to report to a base host. The tasks are:

T_1. measure the temperature locally and send it to the base;
T_2. record the sound in the surroundings and send it to the base;
T_3. record a video of the place and send it to the base.

Task T_1 ends immediately because it creates and sends only one small packet. Tasks T_2 and T_3 occupy the node for more time, because they generate a stream of packets, which is usually a big load for the network. T_2 differs from T_3 because it generates less packets than the latter. These tasks are thus the sensing task that both motes and robots have to perform. They are successful if the packets are not rejected by a node on the route to the host (Sec. 4.3).

Motes' and robots' behaviours are different in the case of the fourth task:

T_4. motes broadcast help requests, robots answer to them and travel where they are needed.

In a general application, motes might decide that they need help by analysing their data, or after instructions from the base. In our experiments, motes' help needs are modelled with a stochastic process. Robots listen to incoming help requests and answer to them by travelling where they are required. Robots and motes co-ordinate their actions through a well defined protocol . In the context of this paper, it is sufficient to say that motes consider a failure if they do not need to send any help request, a success otherwise (a mote's success does not depend on robots' action). A robot considers the task successful if it can reach the requesting mote, a failure if there are no pending requests or if it can not reach its destination.

Assuming that the robots can know the direction of a nearby mote[3], the robots can use the route discovery capability of the network layer to find a path to their destination. In WSNs, the topology of the network usually corresponds to the topology of the environment. Robots do not need a map of the environment because the network topology can be seen as a simple map.

4 Networking

The literature proposes several routing protocols for networks with mobile nodes [15]. We focused our attention on *AntHocNet* [16] which is a self-organising routing algorithm, inspired by the food searching behaviour of ants. There are two main reasons why we chose this algorithm: firstly, it is inspired by ants' behaviour and perfectly fits in the context of our work; secondly, and more important, Di Caro et al. [16] showed that it performs better than AODV [17], which is the common reference point for *ad hoc* networks. We modified the original algorithm to fit into our work. We describe below the algorithm used for our experiments. We are going to explicit the differences w.r.t. the original algorithm during our exposition.

Each message coming from the application layer belongs to different classes, one for each type of task. The network layer of node i keeps routing information in a number of tables ${}_c\mathbf{R}^i$ for each class c. There is an additional class that is used for the messages originated by the network layer, like delivery error notifications or route discovery responses.

Each entry ${}_c\mathbf{R}^i_{nd}$ is a tuple $\langle t, h, e, s, m, v \rangle$[4] that records some statistics about the path from node i to node d using node n as next hop for a message belonging to class c. The entry t is the estimation of the transmission time, h is the number of hops in the route, e is the energy required for transmission, s is the minimal signal-to-noise ratio of all the links in the path, $m \in \{\text{true}, \text{false}\}$ denotes if the

[3] It is not the purpose of our work to address this problems, but it might be done, e.g., by triangulating the signal emitted by a node, by using directional antennas, or by means of a vision system.

[4] In *AntHocNet*, there is only one routing table, because it does not differentiate messages into classes. Each entry is just one real number: the estimation of delivery time for a message.

next hop n is mobile, and $v \in \{\text{true}, \text{false}\}$ whether it is still valid for routing. ${}_c\mathbf{R}^i_{nd} \in \mathbf{R} = \mathbb{R}^3 \times \mathbb{N} \times \{\text{true}, \text{false}\}^2$.

Storing several statistics allows the node to use different routing strategies. Every node might have different objectives to maximise. It might choose, for instance, the route with higher minimal signal-to-noise ratio, increasing the reliability of the message delivery. The node might also choose to use different criterion in different moments. If it detects important information to be sent, it might decide to use a route that minimise the end-to-end delay. If the node has low power level, it might decide to send the message to a near node to reduce transmission power.[5]

When the application layer wants to send a message to a host for which there is no known path, the network layer starts a route discovery process. When one or more routes have been found, the network layer routes the data on the newly discovered paths. The processes of route discovery and routing are done independently for each message class. If a route is known for a class c_1 but the network layer receives a message of class c_2, a new route discovery takes place. This was introduced to address problems, like anycast communication and spatial load balancing of the routes, that we do not address in this paper.

To describe the routing algorithm, we need to describe the route discovery process (Sec. 4.1) and the route selection (Sec. 4.2) separately. We then describe an additional feature of our system in Sec. 4.3: a packet filtering mechanism. Sec. 4.4 describes the additional packet types used by the network layer.

4.1 Route Discovery

If a node wants to send packets to a destination for which it does not know any route, it broadcasts a RouteDiscovery packet, containing the address of the desired destination d. The request is treated by the other nodes as a normal data packet.

The packet is transmitted to the node's neighbours. They might know a route that reaches d, or not. If a neighbour knows how to reach the destination, it randomly chooses a next hop n to relay the request. The node uses a function $r : \mathbf{R} \to \mathbb{R}^+$ to transform the statistics about the route to d in a positive real value.[6] The probability $\mathcal{P}^i_{nd}$ at node i to choose n as next hop to reach d is:

$$\mathcal{P}^i_{nd} = \frac{r({}_c\mathbf{R}^i_{nd})^{\beta_{\text{disc}}}}{\sum\limits_{j \in N^i_d} r({}_c\mathbf{R}^i_{jd})^{\beta_{\text{disc}}}} \; . \tag{4}$$

[5] The distance of a node can be estimated by the receiving power of the message, as measured by the antenna, that is, by the physical layer. It should be noted however that it is not really important in this case to know the real distance, but the power required for the transmission.

[6] *AntHocNet* does not need the function $r(\cdot)$ since the routing table contains already positive real numbers.

where N_d^i is the set of neighbours for which a path to d is known. β_{disc} is a parameter that can control the exploratory behaviour of the algorithm, in the same fashion as for task selection. For route discovering, it is however set to 1.

The actual $r(\cdot)$ used in our experiments is:

$$r_{\text{time}}({}_c\mathbf{R}_{nd}^i) = \begin{cases} \frac{1}{t} & \text{if } t \neq 0, \\ H & \text{if } t = 0. \end{cases} ,$$

$$r({}_c\mathbf{R}_{nd}^i) = \begin{cases} 0 & \text{if } n \text{ is not a valid hop },\\ H & \text{if } n \text{ is a valid hop and } n = d,\\ \frac{r_{\text{time}}({}_c\mathbf{R}_{nd}^i)}{2} & \text{if } n \text{ is a valid hop, } n \neq d \\ & \text{and } n \text{ is a mobile hop,}\\ r_{\text{time}}({}_c\mathbf{R}_{nd}^i) & \text{otherwise.} \end{cases} ,$$

where H is a high-value constant. As it can be seen, this function increases the probability to route packets through nodes which are not mobile, i.e., the sensors. This is because a link to a mobile node is likely to break soon, while the sensors can form a sort of stable backbone.

If a neighbour does not know anything about d, it broadcasts the incoming request again. Due to broadcasting, the discovery messages can proliferate quickly and follow different paths in the network.

The RouteDiscovery stores the path travelled so far. If a node receives several requests originating from the same node, it compares the path of each packet with the shortest known path, the distance being measured in number of hops. We apply the same filters as [16]: only packets that have not travelled over very bad paths are let through.

We assume that the paths are symmetric: if node A can directly communicate with node B, than node B can directly communicate with node A. On arrival to destination, the receiving node generates a RouteDiscoveryResponse packet. The RouteDiscoveryResponse is sent back with high priority along the same path of the incoming packet. During its travel back, it collects the statistics about the path that will be used to update $\mathcal{R}_{nd}^i$. The statistics are not collected and stored only in the RouteDiscoveryResponse packets, but in all the packets that the network layer receives. In this way, other nodes can passively set up routes to other hosts when they receive a message from it.[7]

At arrival of a new packet with data about the route to d through n, the network layer updates the routing table using a custom function $\oplus : \mathbf{R}^2 \rightarrow \mathbf{R}$. If $\mathbf{r}_{nd}^i$ is the information obtained from the incoming packet,

$${}_c\mathbf{R}_{nd}^i = {}_c\,\mathbf{R}_{nd}^i \oplus \mathbf{r}_{nd}^i$$

[7] In *AntHocNet* only the returning packets set up the routes. *AntHocNet* keeps the routing table up-to-date by means of a proactive strategy. While communicating with the destination, *AntHocNet* generates new RouteDiscovery packets to find new routes. Our solution avoids this, at the cost of slightly bigger network packets.

performs a weighted sum of the real values t, h, e, s and sets m to the new value. All occurs only if both the previous information and the new information are valid.[8]

Once a node starts the route discovering process, it waits for a response for some time and buffers the data to send. If it did not receive any response after some time (5 s), it starts the discovering process again. The node repeats the process for a maximum number of times (5) before giving up. Once a route has been found, it is kept in the routing table for 120 s and then removed.

If a node does not find a route to the destination, it takes one of two actions according to who originated the message: if the message came from the application layer, the network layer notifies it about the failure; if the message came from someone else, it sends a RouteFailure packet to the origin.

4.2 Routing

Once one or more routes have been found, the network layer starts sending the data. The node chooses randomly the next hop for each packet. The probability for each hop is calculated with (4), only using another exponent, β_{route}, higher than β_{disc} (in our experiments, $\beta_{\text{route}} = 2$). The higher exponent results in a greedier behaviour w.r.t. good paths. The probabilistic data routing leads to data load spreading, relieving congested paths.

4.3 Packet Filtering

Most of the packets travelling though the SANET are likely to contain data coming from sensor readings. In order to reduce the congestion of the network, a node might decide, instead of routing a packet, to drop it, for instance, if it contains the same information. The rejection of others' messages plays an important role in the division of labour. It is the source of the competition that is required by the agents to specialise. If an agent can successfully send a message associated to a task, it increases the probability to perform the task again. If it is not successful, someone has taken over it, and thus it decreases its probability to perform the task again. We already showed [3] that similar interactions play a key role in division of labour.

Agents do not need to signal to their neighbours the task they are currently executing, because their output is likely to be read anyway by nearby agents. This is why it is reasonable to directly use this 'free' source of information as base mechanism for agents' differentiation.

[8] In our experiments, it does happen that sensors become mobile, and thus changing m might seem useless. It could happen in other applications that some robot decides to "become" a sensor, that is, not to move. The robot could decide it because the network in that point is particularly under load, or because there was a failure of one sensor and the network lost connectivity.

Each node remembers the last packet that it received of a given class and for a given destination.[9] Upon arrival of a new packet to route, the network layer compares it with the one previously stored. It then randomly decides whether the packet should be routed or not. In case the node decides to route the packet, it increases the probability to route it again later and decreases the τ_{i}^k related to packet using (3). If the node rejects the packet, it decreases the probability to route it later, sends a RouteFailure message to the source and the application layer increases its τ_{i}^k using (2).

This mechanism does not take place in the following cases:

1. the packet is broadcast (it might be some important message to spread in the network);
2. the packet is not the first packet of a stream generated by T_2 or T_3 (streams are interrupted at the beginning, but not when the connection with the destination has already been established);
3. it is a packet belonging to T_4 (this task requires a strict co-ordination between robots and motes, thus it should not be interrupted);
4. the packet comes from a source further than a given hop-count threshold D (packets from near sources have correlated content, and thus they can be dropped without losing much information).

Each node i keeps a table ${}_c\mathcal{Q}^i$ of values ${}_c\mathcal{Q}^i_d \in [\mathcal{Q}_{\min}, \mathcal{Q}_{\max}]$ for known destinations d and packet class c. The probability ${}_c\mathcal{P}^i_d$ to route a packet is

$$
{}_c\mathcal{P}^i_d = \begin{cases} {}_c\mathcal{Q}^i_d & \text{if this is the first packet} \\ & \text{of class } c \text{ to } d \text{ seen, or} \\ \alpha_1\alpha_2({}_c\mathcal{Q}^i_d - \mathcal{Q}_{\min}) + & \\ \quad + \mathcal{Q}_{\min} & \text{otherwise.} \end{cases} ,
$$

$$
\alpha_1 = (1 - e^{-\gamma_1 h}) , \qquad \alpha_2 = (1 - e^{-\gamma_2 \Delta t}) ,
$$

where h is the number of hops travelled by the incoming packet, Δt is the elapsed time from the previous known message and the incoming packet, $0 < \mathcal{Q}_{\min} < \mathcal{Q}_{\max} \leq 1$, $\gamma_2 > 0$ (in our experiments, $\gamma_2 = 0.01\ \mathrm{s}^{-1}$). The coefficients α_1 and α_2 decrease the probability to route a packet for near sources and for information recently transmitted. We want that the effect of α_1 smoothly decreases for $h \to D$. Because $\alpha_1 \approx 1$ when the exponent $\gamma_1 h \approx 5$, we set $\gamma_1 = 5/D$.

Every ${}_c\mathcal{Q}^i_d$ is first initialised to $\mathcal{Q}_{\mathrm{init}}$. If a node decides to reject a packet, it updates ${}_c\mathcal{Q}^i_d$ using

$$
{}_c\mathcal{Q}^i_d = max\{{}_c\mathcal{Q}^i_d - \Delta\mathcal{Q}, \mathcal{Q}_{\min}\} ,
$$

[9] This implementation might require much memory and might not well scale. Other solutions can be used on devices with limited memory. We could use a limited array for recording the last messages. If the array is full and a new message should be stored, then the oldest element can be deleted. This system would have the effect only to weaken the interaction between nodes. This effect could be compensated by some other means, like increasing $\Delta\mathcal{Q}$—see later.

and uses

$$_c\mathcal{Q}_d^i = min\{_c\mathcal{Q}_d^i + \varDelta\mathcal{Q}, \mathcal{Q}_{\max}\} ,$$

if it decides to route the packet.

The threshold D grants that the interactions between agents are localised. Each agent looks only at neighbours no more than D hops away and adapts accordingly. Therefore, agents decide only based on local information, but their decisions have an effect on the whole system, as we show in Sec. 5.

4.4 Other Packet Types

For the correct functioning of the network layer, the nodes need some other information to be exchanged. In addition to RouteDiscovery, RouteDiscoveryResponse and RouteFailure (used for route discovery), the network layer generates Data packets. They contain the information coming from the application layer. The information is routed as explained in Sec. 4.2 and using the routing table corresponding to the message class.

The network layer regularly broadcasts to all its neighbours a HelloMessage. After a node receives a HelloMessage, it expects to receive it regularly. If it does not occur within a given amount of time, the neighbour and all its associated routes are deleted from the routing tables.

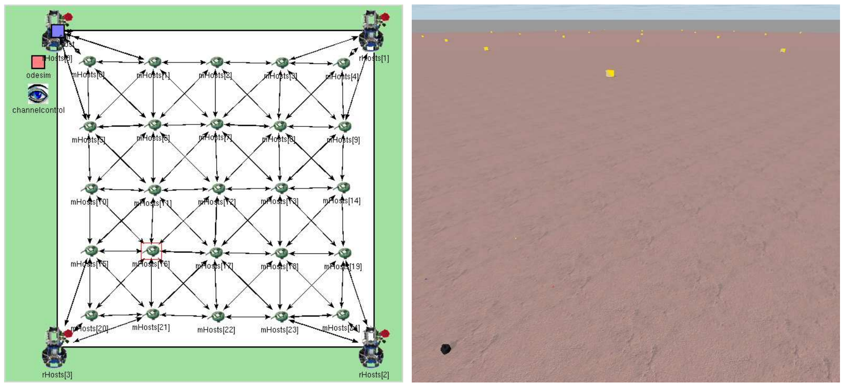
The HelloMessage rates are different for robots and motes.[10] The robots use a higher sending rate (30 s) than the motes (120 s). The nodes use also two different timeouts in case the node is mobile (180 s) or not (600 s). The information about the mobility of the nodes is contained in the HelloMessage.

5 Experiments

We simulated SANETs in a squared area, whose side is 500 m. Twenty-five motes were placed in a grid that covers the environment. This is a likely placement if the motes are dropped from above and the purpose is to uniformly cover the environment. To simulate however a real deployment process, the motes were randomly placed in an area 5 m around the actual grid point. Robots were placed at the corners of the environment. We tested 1, 2 and 3 robots at each corner, for a total group sizes of 4, 8 and 12 robots. Figure 1 shows the set-up, from the point of view both of the network and of the simulated three-dimensional world.

The remaining parameters of our architecture were set as follows: $\mathcal{Q}_{\min} = 0.01$, $\mathcal{Q}_{\max} = 1.0$, $\mathcal{Q}_{\text{init}} = 1.0$, $\varDelta\mathcal{Q} = 0.02$, $D = 2$, $\tau_{\min} = 0.1$, $\tau_{\max} = 10$, $\varDelta\tau = 0.5$ and $\tau_{\text{init}} = 3.0$. The sound stream size was 40 kB and the video stream size 400 kB. The choice of the values was based on empirical experience.

[10] *AntHocNet* uses one rate for all the nodes. It should be noted however than in the networks studied in [16] the nodes are all homogeneous. The different rates are then a natural extension to face heterogeneous nodes.



(a) Network view (b) Simulated world

Fig. 1. Set-up of the experiments: views of the network and of the simulated three-dimensional world. In the bottom view, the real dimension of the motes, usually few centimetres, were increased to make them visible. For comparison, the robot in the bottom left corner in the simulated world view is 42 cm tall.

The base host is at the top left corner of Fig. 1(a). The resulting set-up is symmetrical along one of the diagonals.

Robots and motes did not know the address of the base. The application layer of the base broadcast regularly a packet with its address. The agents started working only after the arrival of this message. Given the importance of this information, it was replicated and broadcast also in each node's HelloMessage. Apart from broadcasting its address, the application layer of the base only received and recorded the messages it had received. The network layer of the base was the same as the other nodes.

The MAC and physical layers simulated the IEEE 802.11 protocol. We used the modules included in the MF.

The help requests were generated by a stochastic process. Each mote decided every second whether it required help or not. We call *help density* the probability per second that a mote requires help. In this set of experiments, help densities were constant during each run. Each newly generated help request was put in a queue in the mote. When motes executed T_4, they checked if the queue was empty or not. If it was empty, they considered it a failure and adapted $\tau^k_{\mathrm{T}_4}$ using (3). If there were pending requests, they broadcast a HelpRequest packet and adapted $\tau^k_{\mathrm{T}_4}$ using (2). We used help densities $12.5\ 10^{-6}\mathrm{s}^{-1}$, $25\ 10^{-6}\mathrm{s}^{-1}$ and $50\ 10^{-6}\mathrm{s}^{-1}$. Each combination of robot group size/help density was tested in forty different runs. Each run was described by a seed for the random number generator and the misplacement of the motes. Another random number generator, initialised with a different seed, was used to generate the help-request events during the simulation.

5.1 Evaluation

The intuitive way of measuring the division of labour is to count how many nodes are involved in a task. This is the method we followed for instance in [3]. This method has one problem: it does not consider the spatial positions of the nodes. Two neighbouring nodes performing the same task would be treated in the same way as two nodes far away. If the task were measuring the temperature, the neighbouring nodes would be doing redundant work.

To account also for the spatial distribution of the tasks, we used the *hierarchic social entropy* [18]. Each agent was represented as a point in a 5-dimensional space. The co-ordinates of each point were given by the (x, y) co-ordinates of the node in the arena (normalised by the arena side), and by the probability of performing T_1, T_2 and T_3. There is no point to consider also the probability to perform T_4 because it is dependent on the other three tasks.

Then, we fixed the value of a parameter d, and we clustered all the points that were no further away than d in the 5-dimensional space. The number of clusters depends on d, thus we denote it with $C(d)$. The total number of agents is A. A cluster i contains $I(i, d)$ agents. Picking up an agent randomly, it has probability $p_i = I(i, d)/A$ of belonging to cluster i. The entropy of the system [18] can be measured by:

$$H(d) = -K \sum_{i=1}^{C(d)} p_i \log_2 p_i \ ,$$

where K is an arbitrary constant, which we set to 1. The hierarchic social entropy is defined as

$$S = \int_0^\infty H(d)\ dd\ . \tag{5}$$

Note that, thanks to the integral, S is not dependent on d, but only on the positions of the points in the 5-dimensional space. Note also that $\exists M : \forall d > M, C(d) = 1$ and $H(d) = 0$. Balch [18] shows that S increases when the system becomes more and more heterogeneous and the agents differentiate themselves. In our case, S increases either if two neighbours have different probabilities of performing the same task, or if the nodes have the same probabilities but are far away.

5.2 Results

There might several interesting data to show about our simulations, like delivery rate, end-to-end delays, and so on. Unfortunately we have no space for all of them. We focus then only on those measurements that are strictly related with the division of labour.

At the beginning of the experiments, the agents of the SANET had uniform probability of performing each task. The hierarchic social entropy was thus at its minimum since the only source of differentiation was given by the positions in the arena of the nodes.

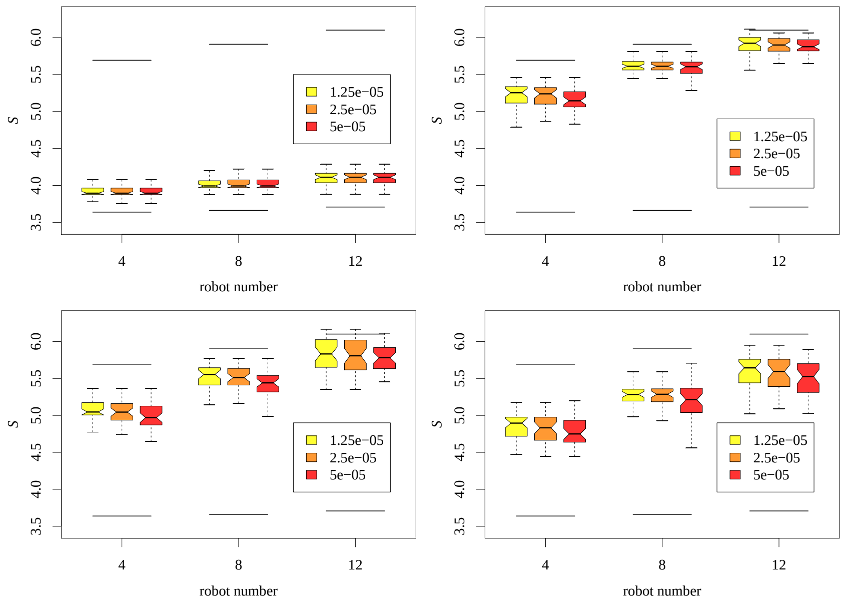


Fig. 2. Evolution of hierarchic social entropy in the SANET during the experiments. The first plot refers to 120 s after the beginning of the experiments (top left plot), the others to 1200 s (top right), 2400 s (bottom left) and 3600 s (bottom right). The three groups of bars in each plot correspond to a group size (on the x axis). Each bar in a triplet refers to a help density (as shown in the legend) and summarises the results of 40 runs. The notches of the bars correspond to the median results. The bars extend to the first to the third quartile of the distribution of results. The whiskers extend till the points that are no more than 1.5 times the inter-quartile distance. The horizontal segments show the approximated upper and lower bounds of the hierarchic social entropy. See the text for discussion.

Figure 2 shows how the hierarchic social entropy developed through time during the experiments. As a reference, we report the median value of the distribution of hierarchic social entropy at 0 s (bottom segments in Fig. 2) and the median of estimated upper bounds (upper segments). The hierarchic social entropy at 0 s was not constant because of the stochastic placement of the motes, thus the median value represents a sort of approximated lower bound. The estimated upper bounds were calculated by keeping fixed the motes, and finding the value of robots' positions and agents' task probabilities that maximised S. S is however complex to maximise, so we searched for the local optimum reached from random initialisation of agents' probabilities and robots' positions. We repeated the process for each motes' initial position. The segments in Fig. 2 are the medians of the local optima we found.

The results in Fig. 2 show that the agents in the SANET specialised themselves. The level of S increased after the beginning of the experiments, meaning that they agents tended either to be near and perform different tasks, or to perform the same tasks but far one from the other. The level of differentiation reaches the maximum value around 1200 s, keeps it till 2400 s and then slowly decreases.

The hierarchic social entropy can increase because the agents are more apart in the environment, because they have different task probabilities, or a combination of both. Given that most of the agents are motes, and that the robots start from outside and move inward the network, the physical distance component can only be reduced. Therefore the hierarchic social entropy increases because of a more heterogeneous distribution of probabilities. We can see that the agents indeed differentiate their activities by looking at the distribution among the agents of the probabilities to perform each task. They look for every robot group size, help density and for tasks T_1, T_2 and T_3 like Fig. 3.[11]

Shortly after the beginning (60 s), there is only one peak. Agents' probabilities were all initialised to the same value, and are now spreading. At 3600 s, three peaks appear. About 30% of the agents had low probability of performing a given task. Less than 30% had probability between 0.4 and 0.5. Finally, there is a small peak also for probability between 0.9 and 1. We are granted that two agents with same probability of performing a task were not likely to be neighbours by the high values reached by S (Fig. 2).

The situation was different for task T_4. The distributions of the probability to perform T_4 looked like Fig. 4. The left peak grew for decreasing help density. The high number of agents that did not perform task T_4 is explained by the relative low help density. The distribution expanded to the right for fixed help density and increasing robot number. This is simply due to the fact that the higher the number of robots, the higher is the probability that a robot answers to a help request. Thus, the higher is the number of successful ends of this task, and the higher the probability that the same robot repeats the task.

The development of the distribution of performing a task is summarised in Fig. 5. We reproduce only the result for twelve robots, but those for four and

[11] Due to the limited space, we do not show all the plots.

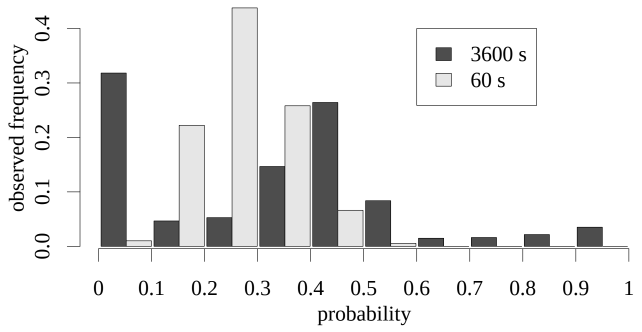


Fig. 3. Distribution of the probabilities to perform task T_2 among the agents (12 robots, help density 50 $10^{-6}s^{-1}$). On the x-axis there are ten intervals of probability. On the y-axis, we report the fraction of agents that have been observed having probability to perform T_2 in the corresponding x range at two snapshots of the system (see the legend).

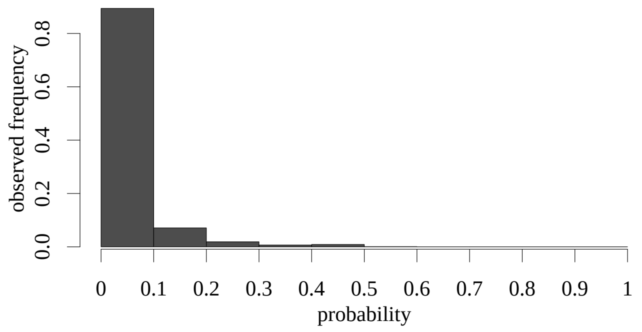


Fig. 4. Distribution of the probabilities to perform task T_4 among the agents (12 robots, help density 50 $10^{-6}s^{-1}$).

eight are similar. What struck our attention is that, while the hierarchic social entropy does not change between 1200 s and 2400 s, the distributions still do. There are new peaks arising (corresponding to the stripes for $P \in (0.4, 0.5]$ that become darker and darker in in Fig. 5) even after 1500 s. If more and more agents get similar probability of performing a task, they become more "alike" and thus the hierarchic social entropy should decrease. This does not occur only in the case the nodes become more "alike" in the task probability space but they are further away in the physical space. That is, if two nodes have the same high probability of performing a task, they are far one from the other in order to keep the same distance in the clustering space and maintain the same hierarchic social entropy.

It is also possible to note that more and more agents set their probability of performing a task next to the central peak value. This is shown by the stripes that become darker and darker beside the main one ($P \in (0.3, 0.4]$ and $P \in (0.5, 0.6]$), starting at about 2300 s. We think that this phenomenon can explain the slow decrease of the hierarchic social entropy at the end of the experiments. At 2300 s in fact, a consistent number of agents has probability in $(0.4 \in 0.5]$, but they are relatively far from each other. When other agents get probability in $(0.3, 0.4]$ and $(0.5, 0.6]$ (and become similar in the task space), they fill the positions between the previous agents. They are therefore also physically nearer, and this is why the hierarchic social entropy decreases.

In Fig. 6, we show a typical snapshot of the distribution of tasks in the SANET. The plot refers to T_3. It can be seen, that when a node had high probability of performing T_3, its neighbours were likely to have a low one. The routes that were used to send the data to the base host are depicted in Fig. 7. The network was split in two halves: there were few links between the top right triangle and the bottom left triangle. Figures 6 and 7 do not represent the steady state of the SANET. The network reached a dynamic equilibrium, where things continually changed. This is especially true for the routes in Fig. 7, since the routing tables entries were removed after a while, and new discoveries took place.

6 Conclusions

This paper illustrated an architecture for division of labour in SANETs. The agents make use of solutions inspired by ants' behaviour. The control architecture is based on strong inter-layer and inter-agent interactions. The latter are local, meaning that they occur only between agents within a given range, smaller than the experimental area. The local interactions are however enough to induce self-organised division of labour in the SANET.

The degree of division of labour can be still improved. It could be argued that it is possible to achieve the maximum degree of division by pre-programming the nodes to do only one task and to spread them alternatively. This solution however introduces other problems that do not touch our solution. Firstly, the deployment process becomes more complex, since the right nodes must be placed

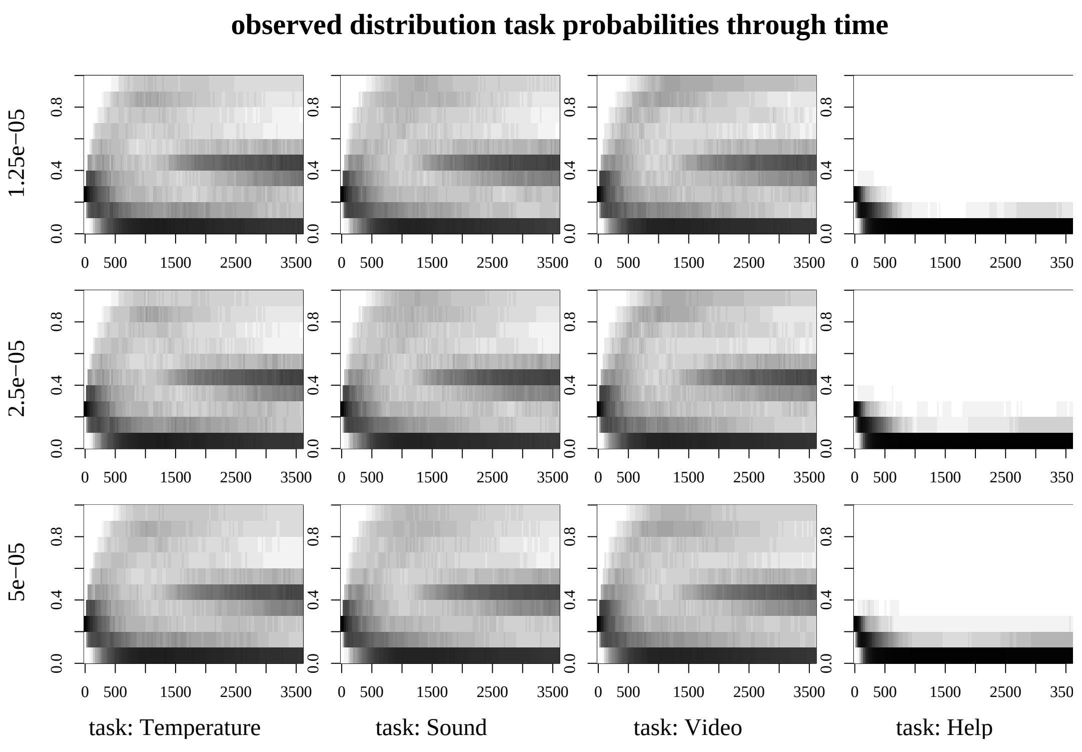


Fig. 5. Dynamics of the distribution of task probabilities (twelve robots). Each plot in the array refers to a combination of tasks (one per column) and help densities (one per row). The task probabilities are on the y axes, the time from the beginning of the experiments on the x axes. The darkness of a cell in position (t, p) is proportional to the number of agents with probability of executing the task equal to p after t seconds from the beginning of the experiments.

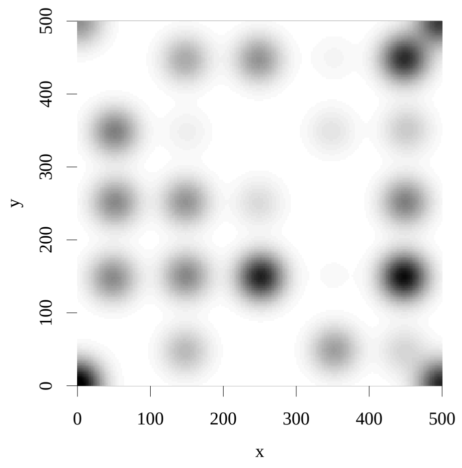


Fig. 6. Distribution of task T_3 among the agents. Each agent is represented by a fuzzy circle. The picture refers to a run with 8 robots, help density 25 $10^{-6}s^{-1}$ and depicts the state at 2490 s after the beginning. The darker the circle, the higher is the probability that an agent performs T_3.

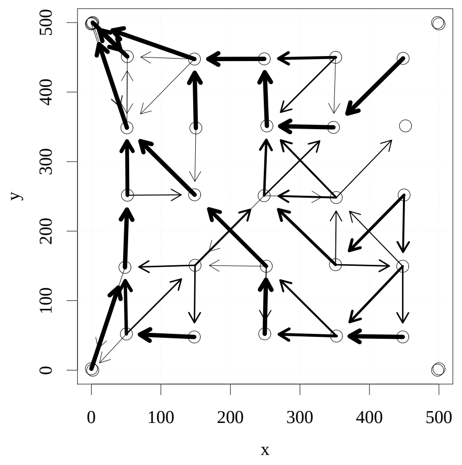


Fig. 7. Routes to deliver the output of T_3 to the base host (in upper left corner). The situation is the same as in Fig. 6. The arrows show for every node the known next hops. The thickness of the arrows is proportional to the probability of choosing a node as next hop.

in the right place to cover the environment. Secondly, the SANET can not adapt to changes in the environment. Nodes require to be reprogrammed for a new situation. Reprogramming might be driven by the base host, which sends the new programs to the nodes. The drawback is that the network has to sustain a bigger load. Another solution would be to use the robots to go and reprogramme individual nodes. The drawback here is that a number of resources, the mobile robots, are taken away from other tasks only for the maintenance of the network. Algorithms similar to those used by our SANET have shown to adapt well to changes in the environment [3,14,16]. Thirdly, one needs to know in advance the characteristics of the environment and the number of motes and robots to find the optimal division of labour. This *a priori* knowledge might not be correct or difficult to retrieve.[12] This knowledge is not required by our architecture.

Future work will test other adaptation rules both at the application layer and at the network layer in order to improve the division of labour. We will also test how our system behaves under dynamic environments. To improve the response time to changes in the environment, we think it will be important to base the packet filtering mechanism also on the content of the messages. Suppose that two neighbouring nodes are measuring the temperature of the environment, but they measure two very different values. It is important that both values are reported, since it might mean that a fire broke out.

Acknowledgements

This work was funded by the Deutscher Akademischer Austausch Dienst (DAAD), grant number 331 4 03 003.

References

1. Akyildiz, I., Kasimoglu, I.: Wireless sensor and actor networks: research challenges. Ad Hoc Networks **2** (2004) 351–367
2. Cardei, M., Wu, J.: Coverage in Wireless Sensor Networks. In Ilyas, M., ed.: Handbook of Sensor Networks. CRC Press, West Palm Beach, FL (2004)
3. Labella, T., Dorigo, M., Deneubourg, J.L.: Division of labor in a group of robots inspired by ants' foraging behavior. ACM Transactions on Autonomous and Adaptive Systems **1** (2006) 4–25
4. Labella, T., Dorigo, M., Deneubourg, J.L.: Efficiency and task allocation in prey retrieval. In Ijspeert, A., Murata, M., Wakamiya, N., eds.: Biologically Inspired Approaches to Advanced Information Technology: First International Workshop, BioADIT 2004. Volume 3141 of Lecture Notes in Computer Science., Springer Verlag, Heidelberg, Germany (2004)
5. Dressler, F.: Self-Organization in Ad Hoc Networks: Overview and Classification. Technical Report 02/06, University of Erlangen, Dept. of Computer Science 7 (2006)

[12] Additionally, if we already knew the environment, it would make little sense to use a WSN to sense it.

6. Gerkey, B., Matarić, M.: A market-based formulation of sensor-actuator network coordination. In Sukhatme, G., Balch, T., eds.: Proceedings fo the AAAI Sping Symposium on Intelligent Embedded and Distributed Systems, AAAI Press, San Jose, CA (2002) 21–26
7. Younis, M., Akkaya, K., Kunjithapatham, A.: Optimization of Task Allocation in a Cluster-Based Sensor Network. In: Proceedings of the Eighth IEEE Symposium on Computers and Communications (ISCC 2003), IEEE Computer Society, Los Alamitos, CA (2003) 329–334
8. Batalin, M., Sukhatme, G.: Using a sensor network for distributed multi-robot task allocation. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2004). Volume 1., IEEE Press, New York, NY (2004) 158–164
9. Low, K., Leow, W., Ang, M.: Autonomic mobile sensor network with self-coordinated task allocation and execution. IEEE Transactions on Systems, Man and Cybernetics, Part C **36** (2006) 315–327
10. Bonabeau, E., Theraulaz, G., Deneubourg, J.L.: Quantitative study of the fixed threshold model for the regulation of division of labor in insect societies. Proceedings of the Royal Society of London, Series B-Biological Sciences **263** (1996) 1565–1569
11. Jesi, G., Montresor, A., Babaoglu, O.: Proximity-aware superpeer overlay technologies. In Keller, A., Martin-Flatin, J.P., eds.: Proceedings of SelfMan'06. Volume 3996 of Lecture Notes in Computer Science., Springer Verlag, Heidelberg, Germany (2006) 43–57
12. Labella, T., Dietrich, I., Dressler, F.: BARAKA: A hybrid simulator of sensor/actuator networks. In: Proceedings of the Second IEEE/Create-Net/ICST International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE 2007), Bangalore, India (2007) In press.
13. Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: the use of simulation in evolutionary robotics. In Moran, F., Moreno, A., Merelo, J., Chacon, P., eds.: Advances in Artificial Life: Proceedings of the Third European Conference on Artificial Life. Volume 929 of Lecture Notes in Computer Science., Springer Verlag, Heidelberg, Germany (1995) 704–720
14. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York (1999)
15. Abolhasan, M., Wysocki, T., Dutkiewicz, E.: A review of routing protocols for mobile ad hoc networks. Ad Hoc Networks **2** (2004) 1–22
16. Di Caro, G., Ducatelle, F., Gambardella, L.: AntHocNet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks. European Transactions on Telecommunications, Special Issue on Self-organization in Mobile Networking **16** (2005) 443–455
17. Perkins, C., Royer, E.: Ad hoc On-Demand Distance Vector Routing. In: 2nd IEEE Workshop on Mobile Computing Systems and Applications, IEEE Computer Society, Los Alamitos, CA (1999) 90–100
18. Balch, T.: Hierarchic social entropy: An information theoretic measure of robot group diversity. Autonomous Robots **8** (2000) 209–238