BARAKA: A Hybrid Simulator of SANETs

Thomas Halva Labella, Isabel Dietrich and Falko Dressler

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

***Abstract*— We present BARAKA, a new simulator for SANETs. The evaluation of algorithms developed for communication and co-operation in this context is usually accomplished separately. On the one hand, network simulation helps to measure the efficiency of routing or medium access. On the other hand, robot simulators are used to evaluate the physical movements. Using two different simulators might introduce inconsistent results, and might make the transfer on real hardware harder. With the development of methods and techniques for co-operation in Sensor/Actuator Networks (SANETs), the need for integrated evaluation increased. To compensate this demand, we developed BARAKA. This tool provides integrated simulation of communication networks and robotic aspects. Thus, it allows the complete modelling of co-operation issues in SANETs including the performance evaluation of either robot actions or networking aspects while considering mutual impact.**

I. Introduction

Sensor/Actuator Networks (SANETs) are challenging research objects. The field was born from the intersection of research on Wireless Sensor Networks (WSNs) and mobile robotics. The result is an heterogeneous system, made of fixed nodes capable only of sensing the environment (the sensors, called also *motes*), and mobile nodes that are also able to change it (the robots).[1] Akyildiz et al. [1] cite 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 WSNs. Research issues include power management, routing, co-ordination algorithms, design, and many other topics. Many algorithms and methods have been proposed to optimise the efficiency of the networking part as well as of the co-ordination between single nodes. Examples are the optimised navigation of robot systems using a WSN [2] or efficient actuation control in SANETs [3].

All these approaches must be carefully evaluated in order to prove the promised capabilities. Usually, a simulation environment is preferred to a lab setup. Even if advances in electronics allow us to experiment with compact hardware sensors and with robots in real environments, simulation is still useful if we want to test larger networks, or if preliminary experiments with real objects might risk to break them.

When we started our research in this area (we present our work in [4]), we discovered a major problem. There is no comprehensive tool for integrated SANET simulation. In other words, there is no simulator that takes equally care of both the networking of the nodes and the realistic movements of the robots (we present an overview of up-to-date simulation tools in the next section). We think that an integrated simulation is necessary for the following reasons: it allows to study deeper form of interactions between robots and motes; most importantly, it reduces the gap between simulation and reality. An integrated detailed simulator allows the experimenter to develop algorithms in simulation and to immediately use them also on real hardware.

The lack of comprehensive tools led to the development of BARAKA, the simulator we are going to present in this paper. BARAKA was motivated by the need of an integrated SANET simulation environment. We might have used two different simulators, one for the network and one for the robots. This would have most likely introduced inconsistent results between the two simulators. This is because one simulator takes particular care only of some aspects and approximates some others, which indeed might be fundamental for the second simulator. BARAKA now allows us to create realistic physical environments in which we can model the robot systems and their behaviour as well as the communication protocols and corresponding properties in a single simulation setup. Basically, BARAKA is built upon OMNeT++, a well known network simulation tool, and Open Dynamics Engine (ODE), a library used to simulate rigid-body physics.

The main contribution of this paper is to present a new integrated simulation tool for SANETs. BARAKA features the following characteristics:

- integrated simulation of communication networks and robotics
- complete modelling of co-operation issues in SANETs
- performance evaluation of either robot actions or networking aspects considering mutual impact.

The rest of the paper is organised as follows. Section II describes network simulators as well as robot simulators. We describe OMNeT++ and ODE, which we used to build BARAKA, in Sec. III and IV, respectively. In Sec. V, we present our new SANET simulator in more detail. Section VI shows a case study to demonstrate the capabilities of BARAKA using a comprehensive set up. Section VII concludes the paper.

[1]We use the word *agent* in the following when we refer to either the entities of a SANET.

II. Network and Robot Simulators

Network simulators are typically used to study the interactions between entities (such as routers, links or packets) in communication networks. Because of the discrete nature of the simulated entities, network simulations are most efficiently carried out as *discrete event simulations* [5]. Discrete event simulators assume that a system can be represented by a set of state variables. The variables change values only at a countable number of points in time. The simulator maintains a set of future events (such as message arrivals or timer firings) and processes this set one event at a time. It starts with the earliest event in the list and continues with the following ones. The simulation times flows with the time associated to the events. It might not advance if two events are concurrent, or advance with big steps if two events are separated in time.

A large number of network simulation tools are available. Among the more well-known and popular tools, there are the commercial simulators OPNET[2] and Qualnet[3], and the free open source simulators ns-2[4] and OMNeT++.[5]

Many simulators come with a number of ready-to-use protocols, mostly including the common Internet protocols, and often also a selection of protocols for *ad hoc* networks (such as DSR [6] or AODV [7]). The simulators can model wired as well as wireless connections. Support for mobile nodes is available in most simulators in the form of one or more mobility models, such as the Random Waypoint mobility model [8].

None of the network simulators that we know supports the simulation of realistic node movements, which take into account both the nodes' physical construction and terrain characteristics.

Robot simulators are used to implement control algorithms for existent pieces of hardware. It is usually preferred to start the implementation of any algorithm in simulation because it is safer, and there is no risk to break a robot or to lose its control. Because of this, the target of the simulator is to ease the transfer of any program from simulation to real robots.

Nowadays, every desktop computer is powerful enough to accurately simulate the robot and its environment. A number of simulators have been developed that can simulate the world physics. One of the most used is the commercial simulator Webots.[6] The world physics simulation is based on the ODE library. ODE is not the only solution for rigid body simulation. There are other commercial libraries available, such as Vortex[7] and Havok[8].

The accurate environment simulation is a recent trend in robot simulators. The RoboCup Simulator[9], used for simulated football matches during the RoboCup [9] competitions, began with a two dimensional environment and has recently added the third dimension and the simulation of players' movements.

The focus of robot simulators is an accurate simulation of a robot's behaviour. They usually use the API that is going to be used on the real hardware (e.g., to get the value of the infrared sensors or to set the speed). They do not consider the problem of modelling the networking of the robots. Although there are simulators that run *over* a network, like Player/Stage [10], they do not *simulate* the network.

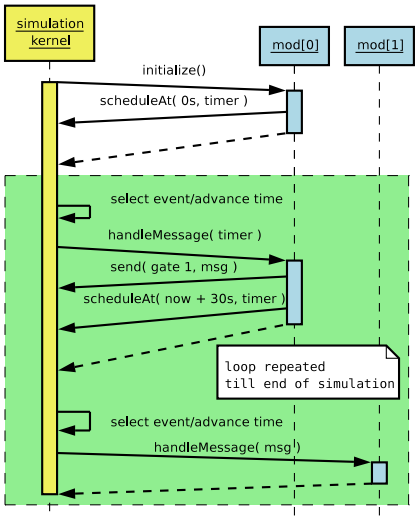

Fig. 1. UML sequence diagram of the OMNeT++ simulation kernel. The left bar represents the simulation kernel, the others two modules. The continuous-line arrows from one bar to another represent normal C++ methods call. The parameters of the calls are those between brackets.

III. OMNeT++

OMNeT++ [11] is a discrete event simulator. It simulates modules, which can send messages to each other through communication channels. Modules connected together form a network. Modules can be compound, made of several sub-modules which form a sub-network between them.

Modules, channels and messages are implemented as C++ objects. Each message represents an event and is stored in the scheduler of OMNeT++(also called the future event list). The simulator, after having initialised the modules, takes the first event in the list and delivers it to its destination. The delivery occurs by calling a method of the module and giving the message as parameter. Modules can send messages to others or to themselves. In this case, they mostly simulate internal timers. A delivery time is associated to each message and determines its position in the scheduler list. The simulation continues till there are no more messages to be delivered or till a specified time limit has been reached.

Let us see, for example, how it is possible to simulate a module, called mod[0], that sends a message every 30 s to another, mod[1]. The situation is summarised in Fig. 1. During the initialisation phase, mod[0] schedules a message for itself at time 0 s, that is, at the beginning of the simulation. This is how module's internal timers can be simulated. When

[2] http://www.opnet.com/
[3] http://www.scalable-networks.com/
[4] http://www.isi.edu/nsnam/ns/
[5] http://www.omnetpp.org/
[6] http://www.cyberbotics.com/
[7] http://www.cm-labs.com/products/vortex/
[8] http://www.havok.com/
[9] http://sserver.sourceforge.net/

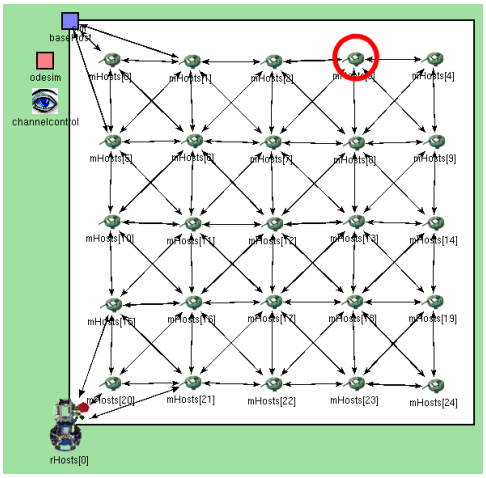

Fig. 2. Snapshot of BARAKA: network view. The icons on a grid represent the position of motes. One robot is at the bottom left corner. The arrows shows the connections, i.e., which are the nodes that can be reached by each agents. Connections are handled by the channelcontrol module of the MF. The circled mote is the target that the robot has to reach in the experiment of Sec. VI.

the simulation starts, the scheduler of OMNeT++ takes this message out of the list, and delivers it to mod[0]. The module is waken up by this message and prepares the message to send to mod[1]. mod[0] sends the message, that is, calls a function of OMNeT++ to store the message in the scheduler. Given the length of the message, the speed of the connection between the modules and the current status of the channel (busy or free), OMNeT++ calculates the delivery time of the message to mod[1]. After having sent the message, mod[0] sets another timer for 30 s later. The control returns to the simulation kernel. It takes the following event in the list, the message to mod[1], and delivers it to its destination. The loop continues till the end of the simulation.

OMNeT++ is a very general simulation tool. There are plenty of extensions that have been created to simulate, among other things, communication networks. We used the "Mobility Framework (MF)"[10] for our simulation of SANETs. MF provides a structure to simulate communicating mobile nodes and additional modules that ease the implementation of a reliable simulation.

A special module, the channelcontrol module, connects hosts that could theoretically communicate with each other, as shown in Fig. 2. After every movement of any host, the channelcontrol updates its connections according to the new positions.

Each host is simulated as a compound module. It contains five other modules (Fig. 3). Three of them simulate the standard networking layers: application layer, networking layer and physical transmission device (the NIC). Any message received by the host goes upward from the NIC, through the network layer to the application layer. The inverse path is followed by a message generated by the application layer and

[10]http://mobility-fw.sourceforge.net/

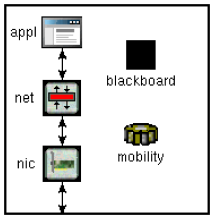

Fig. 3. Implementation of a mobile host in the Mobility Framework. Each host is made of 5 modules: application layer, network layer, NIC, blackboard and mobility.

directed to other hosts.

There is an additional module (mobility) to communicate the new position of the host to channelcontrol, and a blackboard for cross-layer communication.

IV. Open Dynamics Engine

The Open Dynamics Engine (ODE)[11] is a library used to simulate rigid bodies. It provides primitives to define a body by its mass, momentum of inertia, initial position and velocity. Different bodies can be attached to each other through a number of joints: free, extensible, hinges, ball&sockets, and so on. Each body can also have more geometries attached to it, which are used to give a shape to the body. The library also offers primitives to apply forces and torques to the body (as a motor does on the wheels of a car).

The library offers two very important functions. The first one takes the state of the environment at time t and computes the new state at time $t + \Delta$, where Δ is a user defined parameter. This function integrates the equation of motion and returns the solution at time $t + \Delta$. The second function checks whether any two objects are colliding. If it is the case, the libraries takes some measures in order to avoid the penetration of the bodies at the next integration step. Collision detection is also used to compute friction at the contact points. In this way, if we apply a torque to the hinges that connect four spheres to a parallelepiped, and the spheres touch the ground, we can simulate the wheels of a car on a road.

It is up to the program using ODE to set up the objects correctly and to iteratively call the two functions to advance the simulation. Between two calls to the integration step, the program can perform whatever task it needs to do. It can change, for instance, the torque applied to some robots' wheel in order to avoid an obstacle.

An example can help to understand better how ODE works. Our laboratory has a number of Robertino robots[12] (Fig. 4), that we want to simulate later together with a WSN. Robertino is a three-wheeled omnidirectional robot, with six infrared sensors around the body and an omnidirectional camera. The robot uses Swedish wheels, which have a strong grip in the direction of the rotation of the motor, but a very low friction along the axis of the motor. Three such wheels grant the

[11]http://ode.org
[12]http://www.openrobertino.org/

Fig. 4. Picture of a Robertino robot.

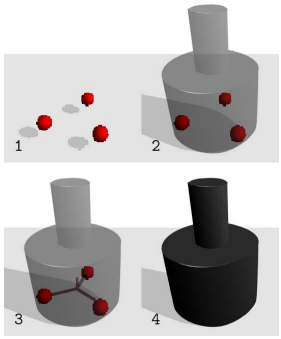

Fig. 5. This sequence shows how the Robertino robots can be built and simulated in ODE: 1) three spheres simulate the omnidirectional wheels; 2) the main body of the robot is approximated by two cylinders; 3) three hinges connect the wheels to the robot's main body along radial axes; 4) the simulated robot is ready to move.

robot the capability to reach every configuration (position and rotation) on a plane.

Figure 5 illustrates how we simulate Robertino with ODE. The main body of the robot is approximated by two cylinders (two geometries). The cylinders are connected to the body placed in the centre of mass of the robot. Weight of the body and dimensions of the geometries respect those of the real Robertino. Wheels are simulated with three spheres. The spheres are connected to the main body through three hinge joints. The hinges are free to turn around the radial axes. The motors of the robot are simulated by applying torques to the spheres along the rotational axes.

ODE allows to specify two friction coefficients for each geometry. We set very low friction between wheels and ground in the direction of the radial axes and high friction in the perpendicular direction, that is, the direction in which the wheel turns. This can effectively simulate a Swedish wheel.

V. BARAKA

BARAKA is the name that we gave to our simulator. It is the result of the integration of ODE into OMNeT++. The integration takes two steps: first, to integrate the collision detection/integration step loop in the OMNeT++ flow; second, to create modules that simulate the robots and the motes both in their physical and networking aspects. These modules are used by the agents' programs to control the behaviours of the agents in the simulated world.

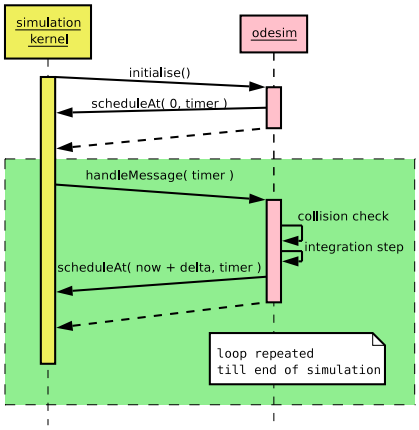

Fig. 6. UML sequence diagram of the integration between OMNeT++ and ODE. The left bar refers to the simulation kernel of OMNeT++. The right bar refers to the OMNeT++ module which implements the real world simulation with ODE.

The ODE loop takes place in an OMNeT++ module called odesim (it is represented as a square in the top left corner of Fig. 2, above channelcontrol). It has no connection to any other module in the simulation. odesim neither receives messages from nor sends messages to the others. During its initialisation, it sets up a timer in the OMNeT++ scheduler. When waken up, odesim performs the collision detection and the integration step of ODE. It then sets up the same timer for Δ seconds later. odesim behaves like mod[0] in Fig. 1, only without the sending of a message (Fig. 6).

We defined a set of interface classes. They allow to modularise the whole simulation and to separate the objects in charge of the simulation from the objects in charge of the agent control. `RealWorldObject` (Fig. 7) formalises the API common to each simulated agents. It includes, for instance, the methods to send messages. `Robot` adds the methods typical for robots (setting the speed of the wheels, getting the sensor readings, etc.). The interface `Controller` specifies the methods that the agents' controllers have to implement. They include most notably a method that is called at each control cycle and a method to handle incoming messages. These interfaces allows a more painless switch from simulation to real hardware. It will be not necessary to rewrite the controllers of the agents, but only the classes that implement the interfaces.

Two classes take care of simulating the agents. They are the ones handled by OMNeT++ simulation kernel: `SimulatedMote` and `SimulatedRobot`. They implement respectively the interfaces described by `RealWorldObject` and `Robot`. We used the multiple inheritance mechanism to allow these classes to simulate both the networking behaviour and the physical-world behaviour.

On the one side, they inherit from `ODEObject`. This class is used as gateway to ODE library and odesim. It allows to create the bodies, geometries and joints related to one object, to get and set the speeds, and so on. When `SimulatedMote` and `SimulatedRobot` are initialised at the beginning of

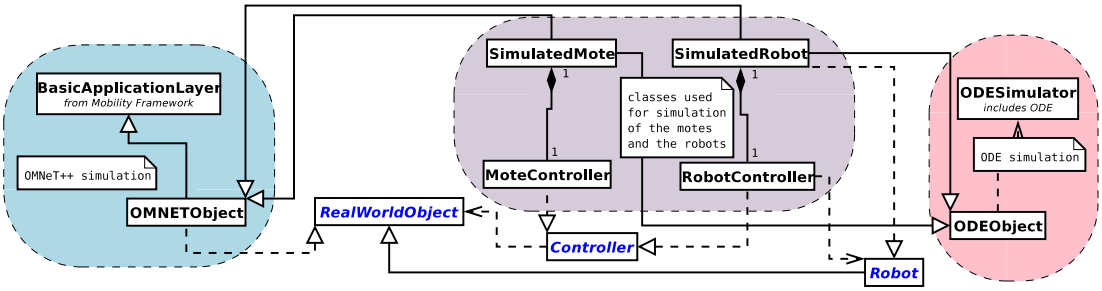

Fig. 7. UML class diagram of the most relevant classes of BARAKA. See the text for the description.

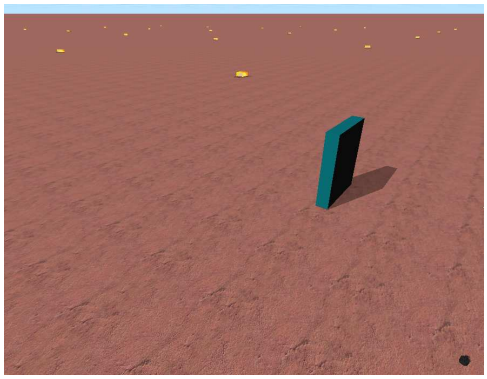

Fig. 8. Snapshot of BARAKA: three-dimensional world view. This picture shows a view of the three dimensional world associated with the network view of Fig. 2. The real dimension of the motes, usually few centimetres, were increased to make them visible. For comparison, the robot in the bottom right corner in the simulated world view is 42 cm tall. The picture shows also one obstacle, a wall, placed between the robot and the first mote it has to reach.

the simulation, they create the objects in the physical world. Motes are simply simulated as light cubes (5 cm side, 50 g mass). Robots are created as shown in Fig. 5, respecting the real masses and sizes.[13] Figure 8 shows the resulting simulated world. During the simulation, the `SimulatedRobot` applies the torques to the object's wheel according to the commands given by the controller. If, for instance, the controller decides that the robot has to move forward, the controller calls the method in `SimulatedRobot` to set the speed of the wheels. The implementation in `SimulatedRobot` calculates which is the required rotational speeds of the wheels. It then calls via `ODEObject` functions of the ODE to set the desired rotational speed by applying a torque on the hinges. `SimulatedRobot`'s work ends here. During the next integration step, odesim will let the wheels rotate. Given the friction with the ground, the rotation of the wheels will result also in a forward translation of the wheel and its connected bodies, i.e., the robot. On request from the controller, `SimulatedRobot` uses information obtained by odesim in order to simulate the robot's sensors. `SimulatedRobot` can get, for instance, a list of nearby objects and use it to calculate the value of the infrared sensors to return to the controller.

[13] `http://www.openrobertino.org/hw/dimensions/overview.html`

The other inheritance branch comes from `OMNETObject`. This class deals with everything that has to do with the networking of the node. It is derived from the application layer class specified by the MF. If the controller wants to send a message, `SimulatedMote` and `SimulatedRobot` take the message from the controller, wrap it into a OMNeT++ message and put it in the scheduler of the simulation kernel. When an agent receives a message from others, the message passes through the NIC, the network layer till the application layer, that is, an instance of either `SimulatedMote` or `SimulatedRobot`. The message is then forwarded to the controller to be processed.

The controllers of the motes and the robots are implemented respectively by the classes `MoteController` and `RobotController`. They both implement the interface `Controller`. Each instance of `SimulatedMote` (`SimulatedRobot`) contains one instance of `MoteController` (`RobotController`) and simulates one mote (robot).

Figure 9 summarises how the classes work together. It depicts the typical working flow of the simulation of a robot.

The network connections to other agents are kept up to date by the mobility module related to each simulated agent. The mobility node regularly queries odesim for the position of the agents, that is, of the objects created by `SimulatedMote` and `SimulatedRobot`.

The advantage of BARAKA w.r.t. OMNeT++ and ODE taken individually is that BARAKA is better when one has to focus on both the networking and the physics of the system at the same time. In [4], for instance, we study a system where the robots' controllers (implemented in the application layers) heavily interact with the network layers of the nodes. If we had run two different set of experiments, one with a network simulator and one with a robot simulator, we would not have been able to understand and exploit the effects of physics on communication and the other way round.

VI. Case Study

We now describe an experiment we ran in order to test our simulator. The following set-up might seem too complex and some design decision that we took might seem unmotivated. This is due to the fact that what we discuss now is in fact only a part of a bigger scenario that we described and analysed in

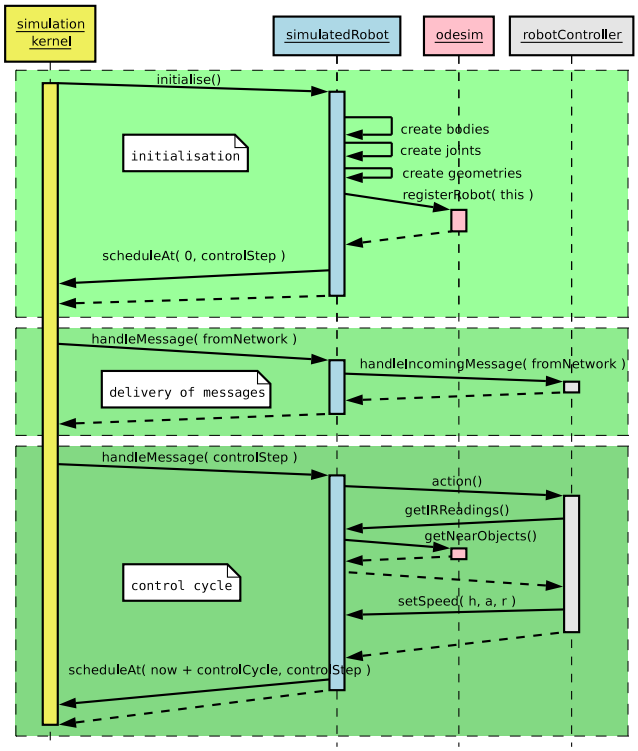

Fig. 9. UML sequence diagram that summarises how a robot is simulated in BARAKA. During the initialisation, simulatedRobot (an instance of `SimulatedRobot`), creates the body in the world handled by odesim. The simulation kernel delivers messages to simulatedRobot, which forwards it to robotController, the controller of the robot. simulatedRobot sets a periodic timer to simulate the robot's control cycle. During the control cycle, the controller might call methods of `SimulatedRobot` to obtain, e.g, the value of the IR sensors. `simulatedRobot` calculates the value using information coming from odesim.

another work [4]. There is unfortunately not enough space to justify our choices here. We think however that it is not so important now to give a sound justification of our set up. Our purpose is to test BARAKA and to give some examples of what we can do with it. The following set-up, albeit complex, can effectively test both the networking and the physical behaviour of a SANET at the same time.

In this experiment, both networking and physical simulation of the environment are important. We simulate a SANET with 25 motes placed on a grid in a square environment of side $500\ \mathrm{m}$, as shown in Fig. 2 and Fig. 8. One mote (highlighted by a circle in Fig. 2) broadcasts a message requesting for a robot's intervention. There is one robot in the bottom left corner that listens for incoming requests. When it receives one, it answers and drive to the requesting mote, avoiding to collide against other objects.

The environment size is smaller than the one commonly used for SANETs, because it is the trade-off between different criteria: on the one hand, we need a large area to have a realistic simulation of the system; on the other hand, larger area increases the simulation time, since the speed of the robot is fixed. The size of our environment is however sufficient to test BARAKA's features, which is our main concern in this

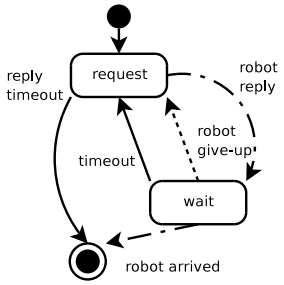

Fig. 10. Motes' behaviour for help-request task. The dash-dotted arrow represents a transition that occurs thanks to an incoming packet, in this case a robot that answers the mote's request or that signals its arrival. The dotted arrow stands for an incoming packet from the robot which gave up the task. Continuous-line arrows are internal events which the mote evaluates at each control step. See the text for the description of the states.

paper.

The robot is too far away to communicate with the mote, thus the network requires a routing mechanism. Our routing algorithm is explained in [4]. It is able to set up several routes (in the network topology) to one destination. The MAC protocol is the IEEE 802.11 as implemented by the modules provided by MF.

The robot needs to know the path (in the environment) to its destination. The robot does not require a map of the environment since it can obtain enough information from the routing table of its network layer. The topology of the WSN can be seen as an approximation of the topology of the environment. The robot can exploit this feature instead of trying to build a map of the environment. Assuming that the robot can know the direction of a nearby mote[14], it can travel in the SANET in the same way in which network packets do.

A. Agents' Controllers

After the mote has broadcast its help request, its controller works as depicted in Fig. 10:

request
: The mote waits for any robots to reply. In our case, only one robot can reply. If the mote does not receive a positive answer within $30\ \mathrm{s}$, it broadcasts the request again. It repeats this for a maximum of 3 times and then gives up.

wait
: The mote waits for the robot that was assigned the task. During this period, the robot is travelling through the network to reach its destination. It regularly sends messages to the mote. Messages are both used as 'keep alive' and to update the robot's route. They contain also the expected maximum time the robot requires to travel one hop. If the mote does not receive a message from the robot again within this time, the mote 'drops' the robot, broadcasts the request again and returns to **request**. Upon

[14] 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.

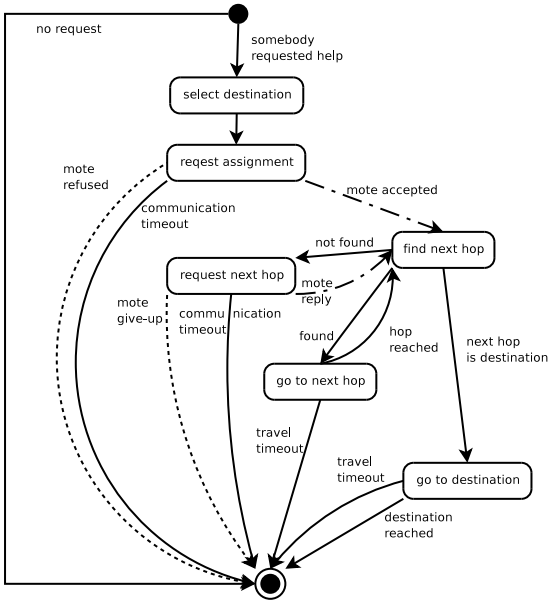

Fig. 11. Robots' behaviour for help-request task. The meaning of dash-dotted, dotted and continuous-line arrows is as in Fig. 10. See the text for the description of the states.

arrival, the robot sends a message to signal it is on the place, and the mote considers the request fulfilled.

The robots starts acting after the arrival of the mote's request. The controller of the robot works as shown in Fig. 11:

select destination
The robot probabilistically chooses one host among the set *HD* of motes that are waiting for help. In this case there is however only one mote requesting, and thus it is chosen with probability 1.

request assignment
The robot informs the destination that it is willing to take on the request. The mote decides whether the robot can continue or not. The mote may reject because the request was already fulfilled, or because another robot is working on it. If the robot does not receive an answer after $30\ \mathrm{s}$, it sends the request again for a maximum of 3 times, then it gives up.

find next hop
The controller looks into the routing table of the network layer to select the next hop of the route to its destination. The next hop is chosen randomly. The probability $\mathcal{P}_{nd}$ of a neighbour n to be selected as next hop to go to d is given by:

$$\mathcal{P}_{nd} = \frac{r(n,d)^2}{\sum\limits_{i \in N(d)} r(i,d)} \ ,$$

$$r(n,d) = \begin{cases} H & \text{if } n = d, \\ \frac{1}{h} & \text{otherwise.} \end{cases} \ ,$$

where $N(d)$ is the set of the robot's neighbours that know a way to d, h is the distance measured in number of hops, and H is a high value constant. This functions selects the next hop in the same way as the network layer does to route data packets (see [4]).

request next hop
If there is no entry in the routing table about destination d, the robot sends a message to the destination and waits for a reply. This message is used to start the route discovery process at the network layer. As in **request assignment**, the robot waits $30\ \mathrm{s}$ before sending another request, for a maximum of 3 times, then it gives up.

go to next hop
The robot proceeds towards the next hop. It periodically checks the IR sensors to see whether it is going to collide with an obstacle, and avoids it if necessary. When the robot is at $10\ \mathrm{m}$ from the mote, it invalidates the hop's entry in the routing table, and sends a message to the destination, in order to start a new route discovery process. When it reaches the distance $4\ \mathrm{m}$, it considers the hop reached and searches for a new one. If the previous message did not get lost, the routing table should already contain the information to find the new hop immediately. If the robot has been trying to reach the hop for more than $300\ \mathrm{s}$, it gives up.

go to destination
In this state, the robot behaves mostly as in **go to next hop**, only the robot does not need to send a message to the destination when it is at $10\ \mathrm{m}$ from it. When the robot is at less that $10\ \mathrm{m}$ from destination, it signals the mote that it has arrived.

It should be clear now to the reader that to simulate this set-up we need a good simulation of both the network and the physical world. The former is required to accurately simulate the communication between robots and motes, the latter to allow the robot to move. Although this set-up might seem to the reader somehow crafty (we recall that this is due to the fact that is only a part of our work in [4]), it is a good test bed for our simulator.

The messages exchanged between the robot and the mote consist of three Boolean fields: `ra`, `ma` and `a`. The robot sets `ra` (it stands for "`r`obot `a`cknowledgement") to `true` when it asks to be assigned the help request, or when it sends messages to find the next hops in the route. The field is set to `false` when the robot gives up. The mote sets `ma` ("`m`ote `a`cknowledgement") to `true` to inform the robot that it is in charge of the request. The mote sets it to `false` if the mote gives up on the task. Finally, the robot sets `a` ("`a`rrived") to true to inform the mote that it arrived at the destination.

B. Measurements

There are several things that can be measured with BARAKA. In fact, BARAKA can be used to estimate or measure whatever observable might interest the researcher, as long as the measurements can be expressed in the form of C/C++ code to insert in the source.

For instance, one might be interested in the trajectory of the mobile nodes. It is easy to modify the mobility modules of

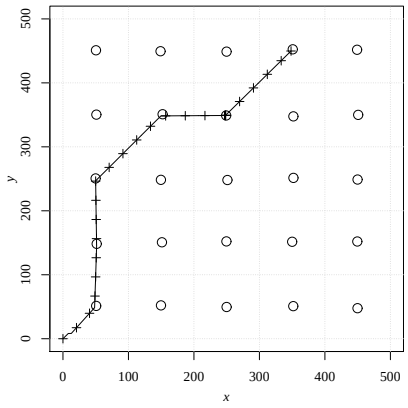

Fig. 12. Trajectory that the robot followed to reach its destination. Ticks mark the position every 30 s. Note the avoiding manoeuvre at the bottom left corner: the straight line is interrupted by an obstacle.

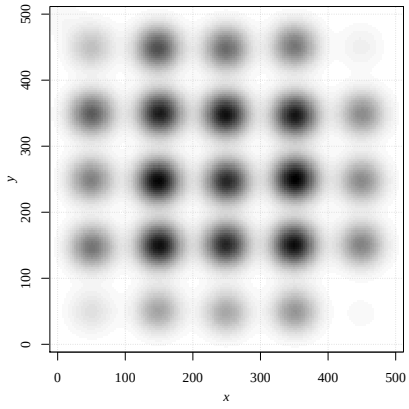

Fig. 13. Number of packets received by each node during the experiment. Each node is represented by a fuzzy circle. The darker the circle, the more packets the node received.

each node to log the position. The result is shown in Fig. 12.[15] As an additional example, we analyse in [4] the degree of division of labour between the agents. The mean time to travel to the next node and the time needed to avoid an obstacle might be other interesting measures that can be easily collected by our simulator.

It is at the same time possible to perform measurements on the network. If one wanted to see which parts of the network are under load, one could plot the number of packets received by each node, as we do in Fig. 13. It is also easy to measure the time it takes to discover a route,[16] the end-to-end delay,[17] or to see the routes that packets travel (see [4]).

VII. Conclusions

We described BARAKA, the simulation tool that we developed for a comprehensive analysis of SANETs. This simulator is more advanced than the ones currently available in the sense that it can accurately simulate the networking and the physical world of the system. This goes at the cost of some more computation time. The overhead is however negligible: in [4] we simulated up to 25 motes and 12 robots, and the speedup with reality was still high. The advantage of BARAKA is that it allows a researcher to do experiments on new forms of robot/mote and application/network layer interactions. It also reduces the work necessary to port the algorithms on the real hardware.

However, the simulator still lacks a validation in more complex scenarios, and our future work will for sure go in this direction. Our preceding experience with simulation both of networks and of robots make us confident in a successful validation: when designing BARAKA, we used our experience to improve its reliability.

[15] Some movies from this experiment, can be seen at `http://www7.informatik.uni-erlangen.de/~labella/comsware07.html`

[16] In our case: minimum 1 ms, first quartile 1.7 ms, median 10.6 ms, third quartile 73.9 ms, maximum 7.5 s.

[17] Minimum 0.7 ms, first quartile 2.1 ms, median 8.7 ms, third quartile 20.8 ms, maximum 0.25 s.

Acknowledgements

Thomas Halva Labella would like to thank the DAAD (Deutscher Akademischer Austausch Dienst), grant number 331 4 03 003, for the fellowship that funded this work.

References

[1] I. Akyildiz and I. Kasimoglu, "Wireless sensor and actor networks: research challenges," *Ad Hoc Networks*, vol. 2, no. 4, pp. 351–367, 2004.

[2] M. Batalin and G. Sukhatme, "Using a sensor network for distributed multi-robot task allocation," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2004)*, vol. 1. IEEE Press, New York, NY, 2004, pp. 158–164.

[3] F. Dressler, "Network-centric Actuation Control in Sensor/Actuator Networks based on Bio-inspired Technologies," in *3rd IEEE International Conference on Mobile Ad Hoc and Sensor Systems (IEEE MASS 2006): 2nd International Workshop on Localized Communication and Topology Protocols for Ad hoc Networks (LOCAN 2006)*, Vancouver, Canada, October 2006.

[4] T. Labella and F. Dressler, "A bio-inspired architecture for division of labour in SANETs," in *Proceedings of the First IEEE/ACM International Conference on Bio Inspired Models of Network, Information and Computing Systems (BIONETICS 2006)*, Cavalese, Italy, Dec. 11–13, 2006, In press.

[5] A. M. Law and W. D. Kelton, *Simulation modeling and analysis*, 3rd ed. Boston: McGraw-Hill, 2000.

[6] D. Johnson, D. Maltz, and Y.-C. Hu, "The dynamic source routing protocol for mobile ad hoc networks (DSR)," Internet Draft, 2004. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt

[7] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on demand distance vector (AODV) routing," IETF RFC 3561, 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3561.txt

[8] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Communications and Mobile Computing: Special Issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, vol. 2, no. 5, pp. 483–502, 2002, p6-3 camp2002survey.pdf.

[9] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara, "Robocup: a challenge AI problem," *AI Magazine*, vol. 18, no. 1, 1997.

[10] B. Gerkey, R. Vaughan, and A. Howard, "The Player/Stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the International Conference on Advanced Robotics (ICAR 2003)*,, Coimbra, Portugal, June 30–July 3 2003, pp. 317–323.

[11] A. Varga, "The OMNeT++ discrete event simulation system," in *Proceedings of the 15th European Simulation Multiconference (ESM'2001)*. European Council for Modelling and Simulation, Nottingham, UK, May 2001.