DISTRIBUTED SELF-ORGANIZED SOCIETIES IN ACTION

Orestis Akribopoulos
Research Academic Computer Technology Institute
Patras, Greece
akribopo@cti.gr

Dimitrios Amaxilatis
Research Academic Computer Technology Institute
Patras, Greece
amaxilat@cti.gr

Ioannis Chatzigiannakis
Research Academic Computer Technology Institute
Patras, Greece
ichatz@cti.gr

Vasileios Georgitzikis
Research Academic Computer Technology Institute
Patras, Greece
tzikis@cti.gr

Marios Logaras
Research Academic Computer Technology Institute
Patras, Greece
logaras@cti.gr

1. Introduction

In order to properly evaluate and examine the characteristics and performance of the implemented algorithms the need of real world experiments was of paramount importance. Although simulators are extensively used for evaluating the performance of an algorithm with respect of a specific value (e.g. total messages exchanged), they have serious limitation regarding the realism of the simulating conditions. One the other hand testbeds with real hardware nodes have the downside that the results from experiments can not be easily reproduced and controlling the simulation is difficult. For this we have constructed a testbed structure comprising of wireless sensor nodes that can be used to run the developed software in a controlled setting. This can be used to reliably reproduce the same experiments or to test how the algorithms react to external events. The user can introduce external events and control the operation of the nodes in real-time, by sending commands to the nodes (e.g. for starting/stopping/resetting). With these controls we can create a wide variety of scenaria and carefully monitor the operation of different protocols under the same settings.

2. Portable testbed structure

The testbed consists of 38 iSense wireless sensor nodes equipped with an environmental module responsible for reading the light and temperature conditions. In addition the testbed has 3 Arduino Xbee nodes, two of which control a small fan and an LED lamp respectively (actuators). The third Arduino issues wireless commands to

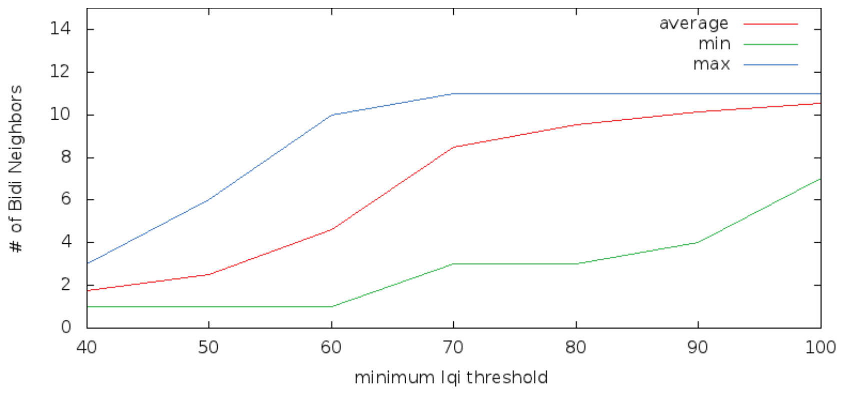

Figure 1: Node Connectivity statistics for different lqi thresholds

the other two using pressure buttons and switches. Three SunSPOTs connected to three Moway robots are also part of the testbed.

The iSense nodes are placed in such way that they create 3 distinctive groups of nodes in the testbed: a large group in the middle, a medium one on the right side and a small one on the left top corner. These groups are connected through bridges so that they can be isolated by disabling one or two nodes. In that way, specific features of the algorithms can be easily showcased creating the proper conditions (i.e. that of an isolated island).

To achieve such setup using this large number of nodes in a confined area, certain limitations in wireless communication between the nodes need to be introduced. In another case, the network topology would be a trivial, fully connected graph. The signal strength of all nodes has been reduced to -18 dB, in order to prevent communication between distant nodes. To ensure this limitation an additional thresholding mechanism is introduced, that leads nodes to discard the messages that do not meet certain criteria (see Section 3.2). Some of the results gathered from tests about the possible LQI thresholds are presented in Fig 1. An extensive trial and error procedure has led to the decision as to where the exact position of the node and antenna angle should be.

In order to monitor the status of the running experiment, each iSense node is connected via USB to a laptop that hosts the Testbed Runtime. This connection is required in order to collect all the debug output from the nodes so that the visualization can be done on later point. The 37 nodes of the testbed are connected to six 7-port USB hubs which are attached to 2 netbooks, three on each netbook.

All nodes are placed on top of two neoFoam surfaces on designated positions. The neoFoam has been drilled so that the USB cables can be put through. The two surfaces are placed side-by-side creating a combined surface of 200 x 70 cm.

On top of each neoFoam surface and 5 cm above, a plexiglass structure is placed. The plexiglass has steamglass opacity except from the areas where the nodes are. This structure serves three different purposes:

- creates a projection surface for the visualization
- leaves the nodes visible for the observer
- provides a terrain on which the Moway Robots are able to move

Around the combined plexiglass surface, a 2 cm high wall is placed in order to confine the the Moway Robots and prevent them from falling off (see Figure 2).

The two netbooks hosting the Runtime of the Testbed are connected via LAN to a more powerful computer, responsible for the visualization of the experiment. Two video projectors connected to this machine project the visualization on the plexiglass surface. In order to cover the whole 200 x 70 cm surface, the projectors are attached to a metal frame that allows their elevation up to 190 cm from the plexiglass, while at the same time, provides stability for the whole structure (see Figure 3). For logistic and portability reasons, the frame consists of many smaller parts that allow its disassembly.

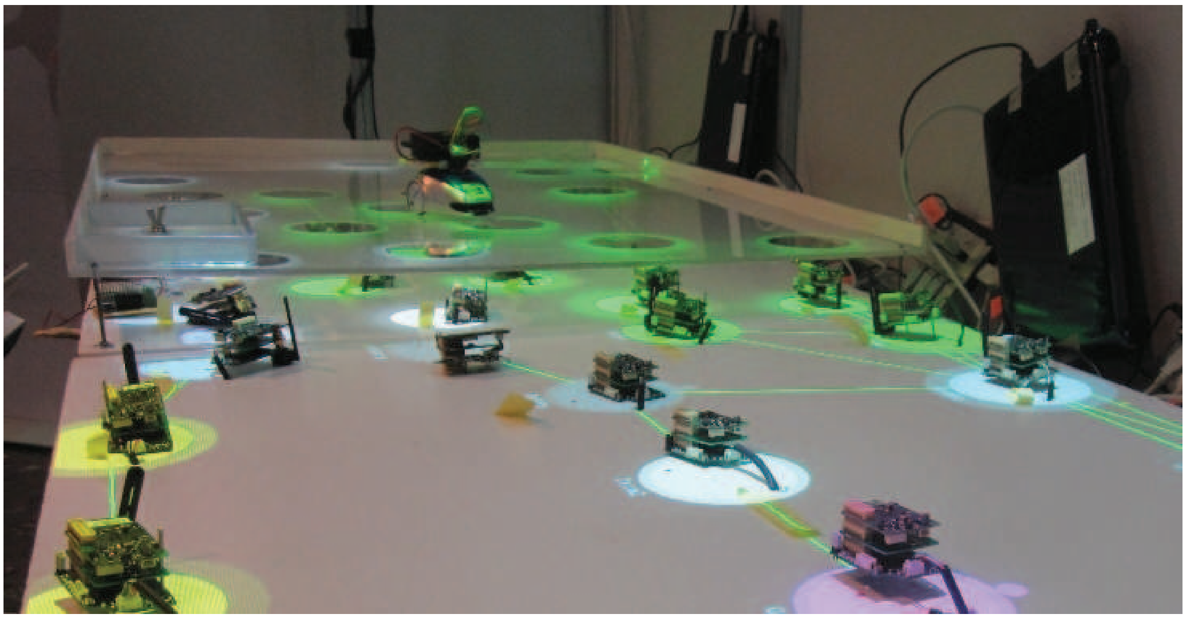

Figure 2: A vertical view of all the three layers of the portable testbed. The sensor nodes are placed on the neoFoam surface. On the background, a Moway robot is moving on the plexiglass surface.

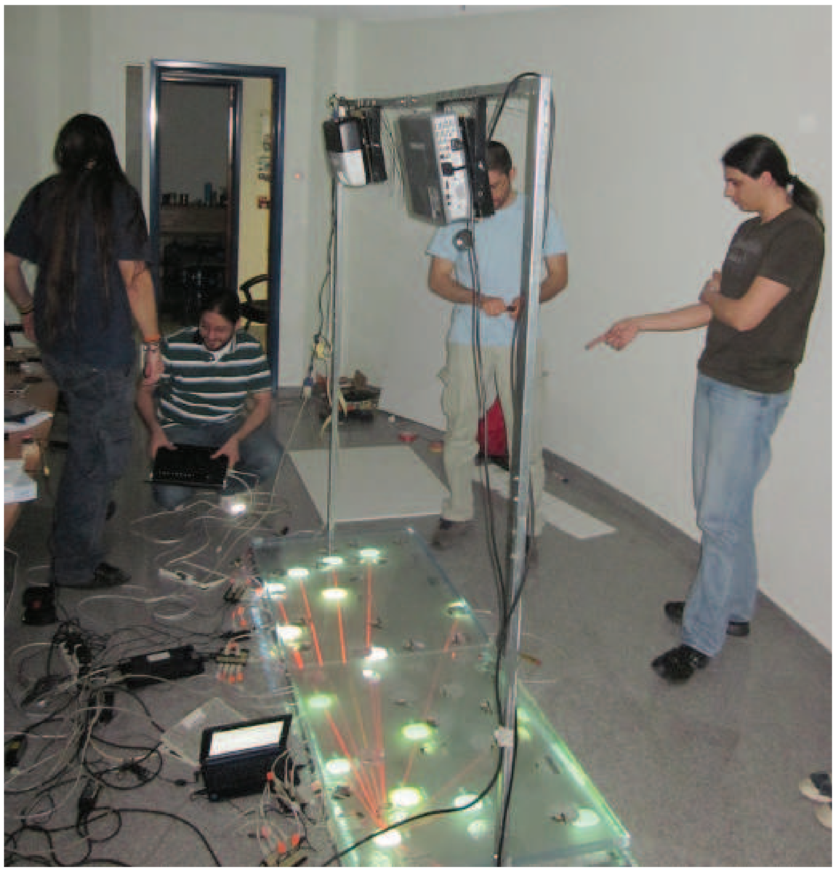

Figure 3: The frame on which the two projectors are mounted. When assembled, keeps the projectors 190 cm above ground.

3. Enabling communication among heterogeneous nodes

A wireless sensor network (WSN) is a network consisting of distributed devices that provide sensing, computing, and communicating features. The last few years systems applied on WSN are becoming more and more noticeable. We have witnessed the emergence of adding new end-user applications to the existing infrastructure. Some of the fundamental problems rising are the hardware, software heterogeneity and the intermittent connectivity.

A first contribution of our work is a careful description of the necessary steps to make a heterogeneous network interoperate and the implementation of a software libray, named mkSense. mkSense enables the communication on a heterogeneous WSN which consists of 4 different hardware platforms (TelosB, SunSPOT, Arduino, iSense). These hardware platforms are the most representative ones, as used by the relevant research community.

Each of the considered sensor devices is equipped with an IEEE std. 802.15.4 - 2003 compliant radio to perform wireless communication. However, each one of them provides only partial implementations of the IEEE 802.15.4, which are not compatible with each other. Therefore the communication between the 4 different devices is not possible out of the box. The differences of the 4 platforms are summarized in the following table, table 1.

Platform	**Processor**	**RAM (KB)**	**Language**	**Radio**	**Payload size**	**Addressing Mode**		**Incompatibilities**
						16-bit	**64-bit**	
Arduino	ATmega328	16	Wiring (C++)	XBee Series 1	100 bytes	YES	YES	Extra MaxStream Headers
SunSPOT	ARM920T	512	J2ME	CC2420	113 bytes	NO	YES	Extra Headers (LowPan)
TelosB	MSP430	10	nesC	CC2420	128 bytes	YES	NO	Auto Ack is Disabled
iSense	JN5139	96	C++	JN5139	116 bytes	YES	YES	

Table 1: Comparison of the different Platforms.

IEEE 802.15.4 provides two different addressing modes, the 16-bit addressing and the 64-bit. The SunSPOT radio stack supports only the 64-bit addressing mode, while TelosB supports only the 16-bit. Radio stacks of XBee and iSense provide both the 64-bit and the 16-bit addressing modes.

The first step on our Heterogeneous Sensor Network was to set all the devices to the 16-bit addressing mode. XBee was set on the 802.15.4 Mac mode with auto-ACKs. We also implemented a new radio stack on SunSPOT which supports the 16-bit addressing mode.

Based on the LowPAN specification, the Sun SPOT library provides routing, meshing and fragmentation using the LowPAN, on the network layer. LowPAN adds some extra headers on the 802.15.4 packets. In particular, after the 802.15.4 headers, two extra bytes are added by the LowPAN which define whether the packet is LowPAN compliant, whether it is fragmented, whether it is meshed etc.

Our network stack does not support fragmentation and mesh routing. So on each radio stack, two constant bytes at the beginning of the payload of each packet had to be added, in order to define that each packet is not fragmented or meshed. In this way, the LowPAN on the network layer of the SunSPOT is bypassed and the communication between the 4 different sensor nodes is possible.

The customized radio stack for SunSPOT was implemented in Java J2ME, while the library, which enables the communication on iSense and Arduino, was implemented in C++. TelosB motes were running the TinyOS version 2.1.0, so the component for the Telosb was written in nesC.

Moreover, mkSense provides a Java API for communication with XBee modules which are directly connected to PC via a Serial to USB regulator. The objective of this API is to provide a flexible and simple to use API to interact with all of the above hardware platforms.

4. Actuators and controller

For the testbed, we use 3 Arduino (a controller and two actuators) which communicate with each other via the iSense network, using mkSense to connect to the iSense modules. When the controller needs to send a message to one of the actuators, it sends it to its neighboring iSense using mkSense, and then the message is propagated through the iSense network, the last of which propagates it to the Arduino which acts as the actuator.

For the actuators, we decided to have an Arduino controlling a fan, and an Arduino controlling a set of LEDs. Both actuators have 3 modes of operation, off, slow, and fast. When the actuators get no commands from the controller, they are set to off. When an actuator gets a command from the controller, it reads the command, and acts accordingly, by blinking the lights slow or fast, or starting the fan in a slow or fast setting.

For the LEDs, we used an Arduino and 2 ShiftBrite LED modules, which are modules with an RGB LED and a shift register. This allows us to individually control each led by connecting them in series and then sending a value over the data lines and shifting that value once for each LED in the series. When the Arduino gets a command from the controller, it reads the input to decide wether it should set the speed to slow or fast. Then, it flashes the LEDs in a red and blue pattern, with the appropriate speed.

For the fan, we used an arduino and connected 2 relays to drive the fan. Just like the LED actuator, when the Arduino doesn't get commands from the controller, it turns off the fan. When it starts receiving commands from the controller, it reads the incoming data, and turns on the fan and sets it to the correct speed as commanded by the controller. The first relay controls the fan by setting it to on or off. This is achieved by connecting it so as to act as a switch for the GND pin of the fan. Therefore, if the relay is open, the circuit powering the fan is open, and the fan is off. When the relay is activated, the circuit closes and we the fan turns on. The second relay controls the speed of the fan by controlling the voltage we're giving to the fan. In the on state, the relay connects the fan's Vcc to the 9v coming directly from our AC/DC converter to the Arduino, which sets the fan to fast. In the off state, the relay connects the fan's Vcc to the 5v coming out of the Arduino's voltage regulator, which sets the fan to a slower setting. Therefore, when we want to turn on the fan to the fast setting, we activate both relays. If we want to set it to slow, we activate the first one and deactivate the second one, and when we want to turn off the fan, we deactivate the first relay.

For the controller, we have programmed an Arduino with 2 switches and a button. The first switch controls the target, which is either the LEDs actuator or the fan actuator. The second switch controls the speed, can be either slow or fast, and last but not least, the button engages the command, and the controller starts sending control commands to the appropriate actuator through the network of iSense nodes for as long as the button is pressed.

5. Controlling the testbed operation

A web-based user interface that allows full control of the testbed has been implemented. It provides a visual representation of the actual testbed and is designed to run on devices with touch-screen interface (i.e. iPad, see 4). The server that hosts the web-page requires an Apache web server with PHP and Javascript support. In addition, the server needs to communicate with the machine hosting the Testbed Runtime, either via LAN or the Internet.

The web page shows an html image map representing the topology of the testbed on which every node is clickable. When a node is clicked or touched, a pop-up menu appears that allows the control of the specific node. The user at this point, can enable, disable or reset the specific node, choosing different icons, for different functions. Additionally, the user is notified as soon as the command takes place or an error occurs.

Apart from the node-wise control, the global control of the testbed is also possible. The user can initiate the flashing of different applications (End-to-end communication, Group Key establishment, Mobile Asset Tracking, Clustering, Aggregation) as well as enable, disable or reset all nodes at once.

Each time the user clicks or touches the an icon, Javascript "onClick" events trigger shell scripts that are executed directly on the Testbed's Runtime.

To operate the Moway Robots, web services are used. In this case, web service methods are invoked that broadcast commands to the Robots using an XBee module. Apparently, for this function the XBee module needs to be in range with the Robots.

6. Real-time Visualisation and Plots

6.1 Real-time Plots

To better describe what is actually happening in the testbed and how the nodes react to changes of their environment we created an application that based on the debug output of the sensors generates real time statistics and presents them using charts. The debug output of the sensors is provided by Testbed Runtime and the generated information is continously added to a text tracefile. This tracefile is parsed and statistics for the events generated, messages exchanged, the number of clusters formed and the controller application success rate are plotted using JFreeChart.

Such information is extracted during parsing by searching for keywords that indicate the type of event that

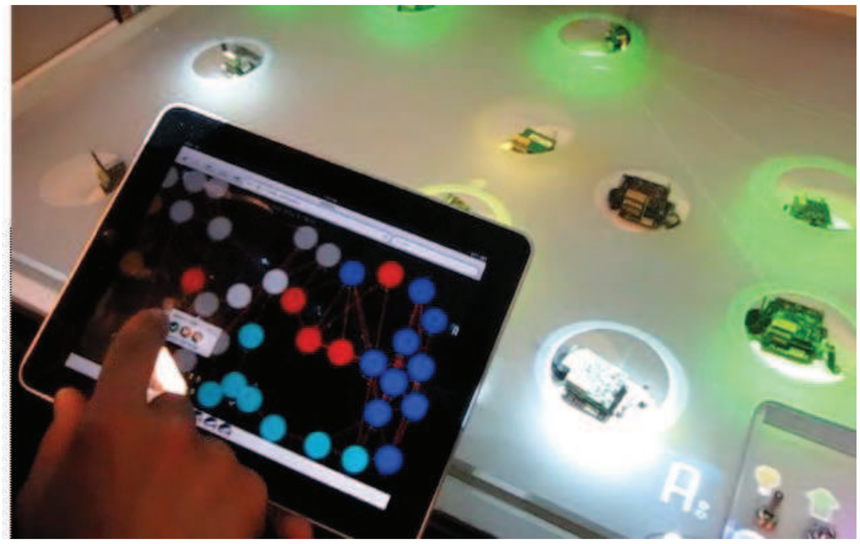

Figure 4: An iPad is used to control the testbed either as a whole or each node separately.

happend. In order to parse the output correctly all algorithms had to use a template for all debug messages. For starters the most obvious would be for the algorithms to add a prefix (the algorithm's coded name) to all debug messages. Also some special characters we inserted in the prefix to better indicate the category of the debug message (ie 'S' for sending a message or 'B' for bidi links of the ND module. Then to define the contents of the debug message we use a series of tokens seperated by ';' just to help the tokenization proccess.

The major templates for the messages are:

- Send Message Parsing : `ModulePrefix;source;type;destination;`
- Event Parsing : `ModulePrefix;...;`
- Clusters Parsing : `CLP;node_id;role_in_cluster;cluster_id;`
- Controller Application Parsing : `{FLS:lamp/FLS:fan/FLR:fan/FLR:lamp};...;`

After parsing is complete all data are added to a number of XYSeries (a data type provided by JFreeChart), ploted with XYLine charts and displayed in the application's relevant JFrame. An example of how statistics are presented is available in Fig 5.

6.2 Visualisation

The Visualisation application (Viz app) is used to show all events happening between the sensors on the table surface using the two projectors. The Viz app is implemented using Processing and requires a computer with two video outputs. To identify the events the viz app parses line by line the same file the Real-Time plots do. The file is generated from the Testbed Runtime. A screenshot from the application is available in Fig 6 and Figs 2,3&4 are pictures of the table on various stages with the Viz app working.

Using the Viz app we are capable of displaying a *circle* over every `iSense` and `Arduino` node. This circle is used to indicate the events of the specific node. By drawing *lines* between the circles we can paint all bidirectional links between the nodes. The bidirectional links are reported from the ND module. Also over all bidirectional links we do paint the *packets* exchanged and use an animation in order to move the packet from the one node to the other. Viz app parses the same debug messages as the Real-Time plots application and uses the source and destination addresses. Also for the ND beacons and all broadcast messages we implemented a *pulse* event generated from the center of the node sending the message. Also we use the Viz App to display information about the CL module, by parsing the CLP events described above to paint nodes of the same cluster using the same *color* and add a special *marker* to all cluster leaders. For the AG module the *values* generated and gathered are also displayied on the side of the node.

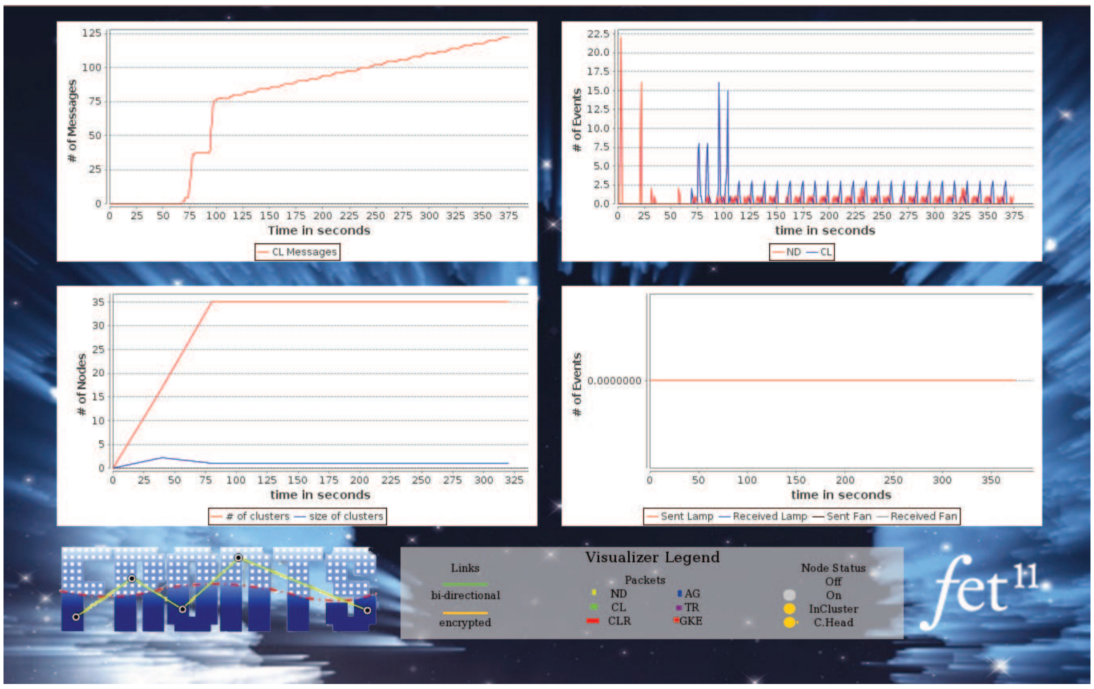

Figure 5: Screenshot of the Real Time Plots application used for monitoring the testbed

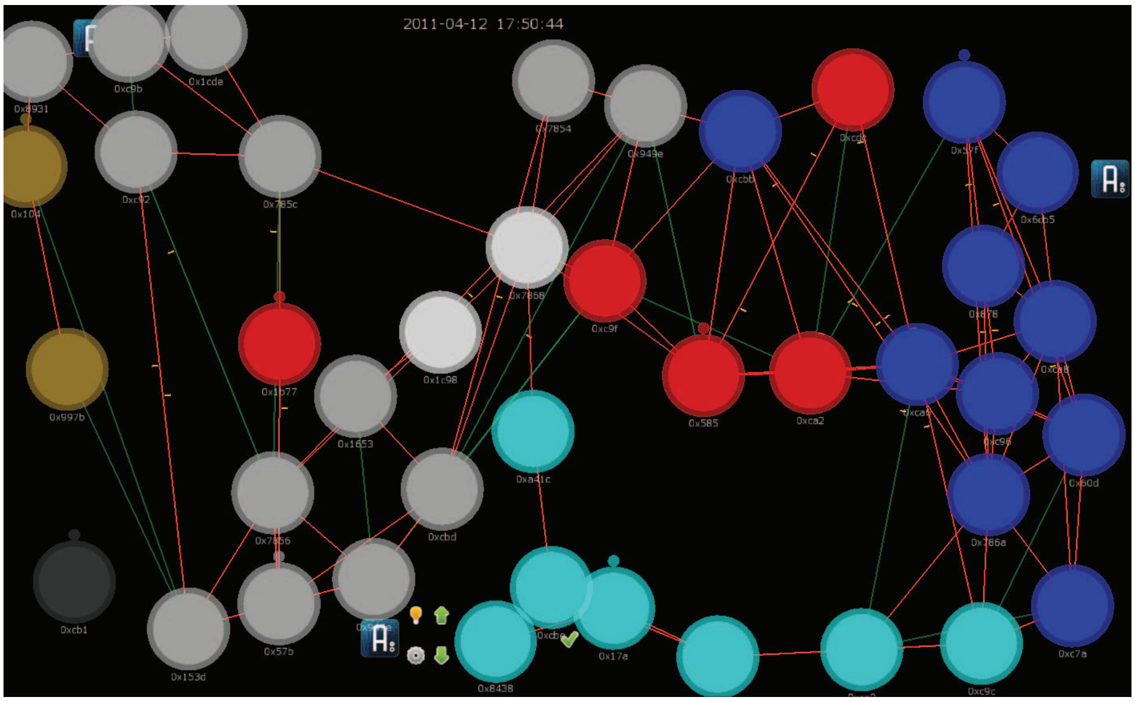

Figure 6: Screenshot of the Visualisation application used to paint on the table surface what is happening

Except for the `iSense` nodes we also have 3 `Arduinos` and some `Moway` robots. As we do not monitor those nodes from Testbed Runtime we use the events of the `iSense` nodes to extract these events. The `Arduino` nodes send and receive messages from the `iSense` nodes, and those debug messages of the iSense nodes indicate the `Arduino` events. Also for the communication with the `Moway` robots we do not need to actually display the send receive events, but we do display the message *queues* that are generated and need to be sent using the Delay Tolerant System and the `Robots`.