Journal of Network and Innovative Computing
ISSN 2160-2174 Volume 1 (2013) pp. 214 - 223

Mobile agents in wireless sensor networks

Jan Horacek[1], Frantisek Zboril jr.[2] and Zak Jakub[3]

[1]Faculty of Information Technology, Brno University of Technology
Bozetechova 1/2, 612 66 Brno, Czech Republic
ihoracek@fit.vutbr.cz

[2]Faculty of Information Technology, Brno University of Technology
Bozetechova 1/2, 612 66 Brno, Czech Republic
zborilf@fit.vutbr.cz

[3]Faculty of Information Technology, Brno University of Technology
Bozetechova 1/2, 612 66 Brno, Czech Republic
izakjakub@fit.vutbr.cz

***Abstract*—This article describes a novel routing protocol for mobile agents in wireless sensor networks. Protocol was influenced with some specific needs of agents, such as agent mobility. Agent platform should select the node where will be agent placed automatically. Therefore we propose to use vague addressing instead of exact addressing. Agent selects a few parameters such as target area or sensors that are necessary for work and platform will move the agent to best node itself.**

***Keywords*-WSageNt; agent; mobile code; routing protocol; WSN; vague addressing;**

I. Introduction

Wireless sensor networks are used in various environments and we can track efforts to bring mobile agents into such decentralized platform. Each node in the wireless sensor network contains its own micro-controller, its own source of energy (battery). Node can sense surrounding environment and is able to communicate with its neighbors. Required properties of an agent by Wooldridge [10] are to be autonomous, reactive, proactive and to have social abilities. We can see that wireless sensor networks can benefit from mutli-agent aspects. Each node can be autonomous since it has its own micro-controller, it can communicate so the agent placed on such node can have some social abilities and it can react on changes in the environment since it has sensors, which sense outside changes.

There are many multi-agent platforms for wireless sensor networks such as Agilla [1], AFME [2] or HERA [3]. There are many other platforms but the main weakness of all of them is the communication between two nodes, which are not direct neighbors. Such communication is called multi-hop and we have to use some routing protocol to deliver a message between two distant nodes. For example Agilla uses simple geo-routing algorithm that we will describe in next section.

As not many of nowadays agent platforms focus on multi-hop communication, we can state that none of them focuses on agent migration and agent placement in any details. Our WSageNt platform [4],[5] provides services, that will assign the best node for agent's work automatically [13]. WSageNt does not use exact addressing of nodes, which is used by other platforms. Exact addressing can limit agent stay in the network. Reasons why it is so, will be described later. We rather use vague addressing in the form: "Move me into the area, which is 100 meters north-east, where I will need temperature sensor for my work". Such agent migration is demonstrated on Fig. 1. Agent selects desired destination area and our platform chooses the best node for agent stay from the candidates in this area.

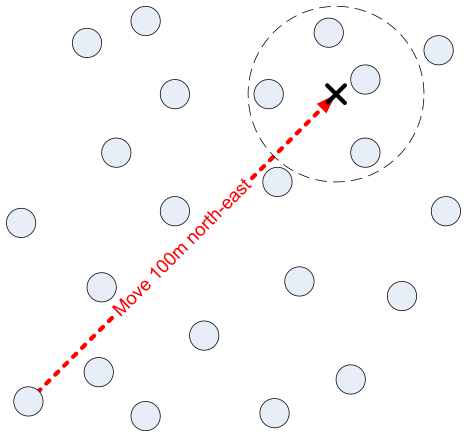

Fig. 1. Demonstration of agent migration with vague addressing.

This paper focuses mainly on the process of assigning the best node for agent stay. The assigning process is divided into two stages:

1) Finding the target area desired by an agent.
2) Selecting the best node in the target area for agent work.

Therefore the first part of this paper is dedicated to process of finding the target area. We will describe geo-routing algorithms and we will sketch out modifications to the existing algorithm that are necessary for our purposes. Second part of this paper describes the process of selecting best candidate in the target area. We use trust between neighbors and we will demonstrate how the misbehavior of some node can be detected. This paper contains also demonstration of how the agent selects the contexts that are important for its work and what influences decision process.

II. Geo-routing algorithms

Geo-routing algorithms work with an idea not to work with an address burned in its radio module, but with a set of

coordinates that identifies a geographic position of node in the network. We usually work with 2 coordinates. For example the Agilla uses static $m \times n$ grid and nodes are addressed with two integers. Agilla platform expects that nodes are aligned into the regular 2D grid and nodes are able to communicate with its top, bottom, right, left and diagonal neighbors. Fig. 2 shows how the packet from a node $[4, 4]$ is retransmitted to a node $[1, 1]$. The packet travels step by step to a neighbor that is closest (geographically) to the destination address.

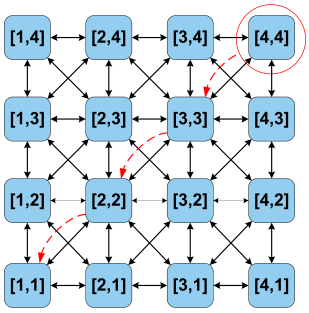

Fig. 2. Greedy variant of geo-routing algorithm used in Agilla.

We have to mention that the multi hop communication can play a significant role for agent stay in the network. Agent should have the ability to be mobile and there is also the possibility that the agent will have to visit many nodes for its work. In the traditional point of view, we have to know exact address of destination to be able to deliver the message. All of the nowadays multi-agent platform uses exact addressing, which can limit the stay of agent in the network. Fig. 3 shows what could happen, when an agent traverses through network for a long time. Lets say that agent's work is measuring of data on the edges of some rectangle. The agent pseudo-code to travel first edge would look like:

```
x, y = 1;
(for ;x<height; x++)
{
  do_something();
  move_to_node(x+1);
}
```

Such simple pseudo-code could have troubles when we use exact addressing of nodes. The operation `move_to_node(x+1)` could fail. One reason could be that destination node is out of battery or it has another problem. Another problem could arise when the network is irregular. We can imagine such situation when nodes are dropped from plane. The algorithm expects that the destination node exists, but such expectation could not be valid. The pseudo-code of the agent has to deal with such situation when we use exact addressing. Therefore the size of code will increase. This could be a problem because the agent has to run on nodes with very limited resources. For an example on MICAz nodes we have only 4kB of RAM that has to be shared between an agent platform and the agent itself.

The agent code is usually stored in RAM memory (better performance of interpretation) and therefore it could cause

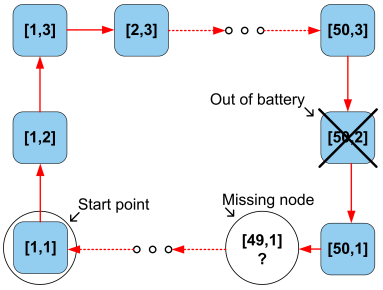

Fig. 3. Demonstration of problems when using exact addressing.

that the agent will not fit into the limited size of memory. We rather implemented such solution directly in the WSageNt platform. The code that deals with such problems is then stored in ROM memory of node, which is considerably larger (128kB at MICAz). Another reason to do so is the fact that this problem solving mechanism would be a redundant part of many agents we send to the network. Our solution also selects the destination node, which should be best for agent's work.

III. Travel to the destination area

The first stage of the assigning process is to travel to the destination area. Our algorithm is based on GPSR algorithm [8], with modified final part. GPSR is similar to the greedy variant of geo-routing algorithm used in Agilla. Agilla platform expects that nodes are placed in regular grid of the size $m \times n$. Main constraints of Agilla platform are:

- Network has to have regular rectangular shape.
- Count of nodes can not be prime number, otherwise we have line topology.
- Each neigbour to the node has to have its coordinates $[x \pm 1, y \pm 1]$.
- Size of the network is static and it is set when we build from the source code.
- There can not be holes in the network.

We will discuss two main constraints of the Agilla platform. Conclusion of first four constraints is that Agilla platform could be used in laboratory environment, but we could have problems to deploy our nodes in real outside environment. We have time to place our nodes in regular 2D grid and we also know what type of agent we will send to the network to test specific example. We can set coordinates to each node at the build time and place them to the expected location. These constrains can limit us in deployment to real environment. We are not able to add one node to the network. We have to place whole line (row or column) and we have to also recompile platform with new $m \times n$ variables and reload all nodes. It can take a serious amount of time to do that.

The biggest issue is last constrain. Greedy variant of geo-routing algorithm has trouble to work properly when there are holes in the network. Aligning nodes to regular 2D grid without holes could be problematic in the outside environment. We have to also consider that some node runs out of battery or it is damaged. This also create a hole in the network. In real applications we see such events frequently. Fig. 4 demonstrates

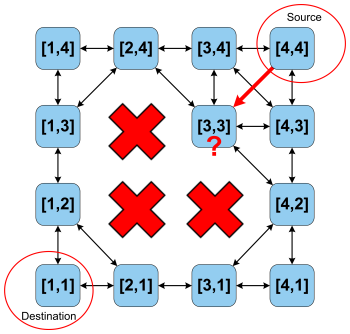

Fig. 4. Problems with greedy variant of geo-routing algorithm used in Agilla.

a problem with greedy variant of geo-routing algorithm, which causes that packet can not be routed.

The node $[4,4]$ wants to deliver some data to the node $[1,1]$. Packet is then transmitted to the node $[3,3]$, which is closest node to the destination. Distances of all nodes to the destination are in the table I. Since we are now located at node $[3,3]$ it has to select a node from its neigbours, which is geographically closest to the destination and which is closer to the destination than node $[3,3]$ itself. There does not exist any eligible neigbour to which we could route a packet. Algorithm is then stuck in local optimum and can not continue to deliver data to destination node.

$y \setminus x$	1	2	3	4
4	3	$\sqrt{10} \simeq 3.162$	$\sqrt{13} \simeq 3.606$	$\sqrt{18} \simeq 4.243$
3	2	-	$\sqrt{8} \simeq 2.828$	$\sqrt{13} \simeq 3.606$
2	1	-	-	$\sqrt{10} \simeq 3.162$
1	0	1	2	3

TABLE I
DISTANCE OF NODES TO THE DESTINATION NODE $[1,1]$.

GPSR algorithm tries to solve such problem of holes (space without nodes on the path to the destination) in the network with the idea to go around them on the edge of hole. GPSR works in two modes:

1) Use greedy variant to route packet.
2) Go around edge of hole.

Algorithm switches from mode 1 to mode 2 in the moment it is stuck in local optimum. It also note coordinates of node (we will label it as M_0) where such switch has been done. It help us to create a virtual trajectory from problematic node to the destination node. With this trajectory we are able to detect a situation that we successfully passed a hole and we can switch back to the first mode. In the second mode we use the right hand rule [9] to go around a hole, which is demonstrated on Fig. 5. Every node that receive a packet to forward will route it to its first neighbour counterclockwise about itself. The hole is always on the right side of the edge (directed from source to destination). If we apply this rule n times we should get back to the starting point and create a face around such hole.

When we are stuck at the node M_0 we select first neighbour counterclockwise about the vector $destination \rightarrow M_0$. Then we continue using right-hand rule until we find target area.

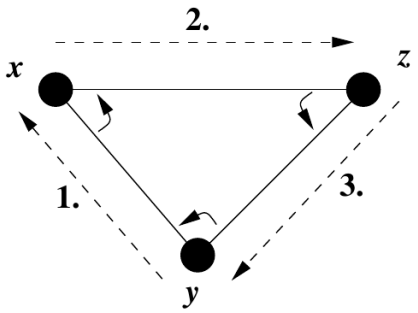

Fig. 5. The right-hand rule *(interior of the triangle)*. x receives a packet from y, and forwards it to its first neighbour counterclockwise about itself, z. [9]

A. Planarized graphs

Authors of GPSR algorithm noticed a problem that can occur in wireless sensor network. There can occur two or more edges that are crossing themselves. Such crossing edges can cause that algorithm does not create a face around the hole. The problem is demonstrated on Fig. 6. On this example we see that algorithm missed to visit a node v and it creates a cycle $x \rightarrow u \rightarrow z \rightarrow w \rightarrow u \rightarrow x$, which is not valid. If we remove these crossing edges $x \leftrightarrow u$ and $w \leftrightarrow z$ algorithm starts to work properly. A graph, where no two edges cross is known as planar [9].

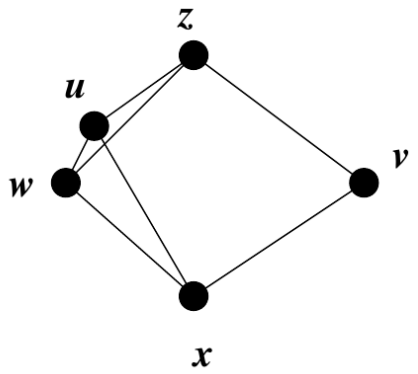

Fig. 6. A network with crossing edges. Starting from x to u, the right-hand rule gives the tour: $x \rightarrow u \rightarrow z \rightarrow w \rightarrow u \rightarrow x$. [9]

Relative Neighbourhood Graph (RNG) [11] is one of the planar graphs used in many disciplines. There exist an edge between u and v if and only if:

$$\forall w \neq u, v : d(u,v) \leq max[d(u,w), d(v,w)] \qquad (1)$$

On Fig. 7 is demonstrated such rule defining if edge exist in RNG. We check if there is no witness w between two nodes u, v and if there is any such node we remove an edge $u \leftrightarrow v$ from the graph. If we remove every edge that does not satisfy rule defined with equation 1 we create a RNG. It is important that such procedure can be done with each node separately in its local neighbourhood table. It is also proved that it does not disconnect the network [9].

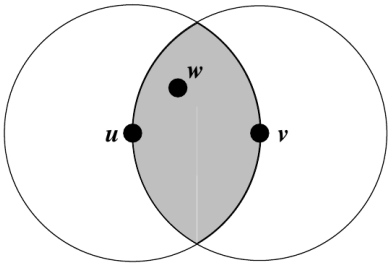

Fig. 7. The RNG graph. Edge $u \leftrightarrow v$ exists if there is no witness w in the shaded area. [9]

Each node can compute if link to some neighbour is part of RNG and it note such information to the neighbourhood table. GPSR algorithm then use all paths in the first mode (greedy mode), while it uses reduced number of paths from RNG to route in second mode.

B. Decision node in the target area

GPSR algorithm uses the exact addressing and therefore it has problem, when destination node does not exist. We modified the final part in the way that the node closest to the destination address is selected as a decision node. The goal of the decision node (DN) is then to determine, which node is the best for an agent stay. The next section IV of this paper will deal with the second stage of the assigning process, which is called the decision process.

We discus a problem of reaching proper decision node now. We will start with basic and most ideal example that is on Fig. 8.

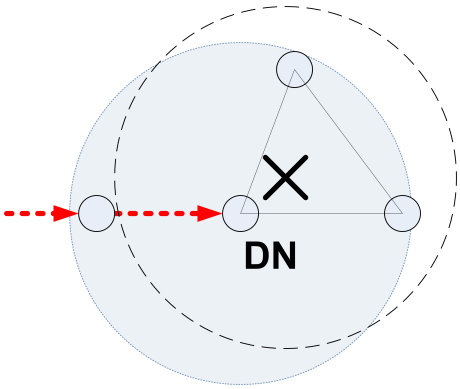

Fig. 8. Ideal situation in final stage of modified GPSR algorithm. We can create a triangle from 3 neighbours. Destination point is inside of this triangle.

Destination that agent wish to visit is marked with cross. Target area is then labelled with dashed circle. Packet reach a node that is inside the target area. Such node pretend itself to be a candidate for decision node DN. Node has to check if it has labelled itself as DN properly. It is labelled properly if and only if:

1) It is closest one to the destination from all nodes in its neighbourhood table.
2) There exist at least two neighbours of labelled DN such as that coordinates of destination are within a triangle created with these 3 nodes.

Every node that mark itself as a candidate for DN has to check these two conditions and pass this test to become DN. If first condition is not met we just forward packet to the neighbour which is closest one to the destination. Such node mark itself as candidate for DN and check these two conditions again.

If second condition is not met it means that destination is located inside some hole. Such situation needs a special handling that we have to discuss.

C. Destination inside a hole

Since we have detected that the destination point is inside some hole there exists a possibility that there is some node from the back side of hole, which is closer to the destination point. We have to go around such hole, create a cycle and select from all possible candidates (for DN) one node, which is closest to the destination point. Because of that our packet contains field with coordinates of best candidate M_{best} that we have found yet.

We switch to work in second mode of GPSR algorithm. We mark coordinates of node M_0, where we have made this switch. Fig. 9 demonstrate a situation when we are in second mode already and we have found first candidate DN1.

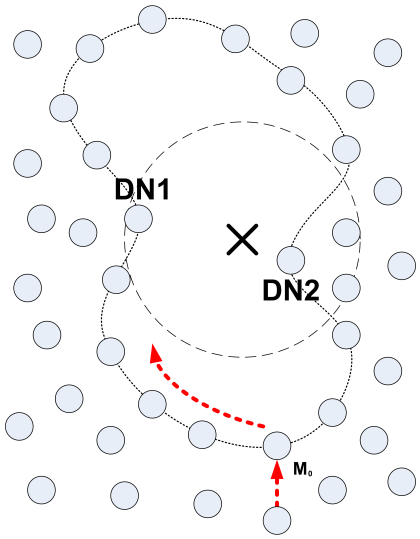

Fig. 9. Selecting DN from all candidates. We do one round to select best candidate. We added third mode, when M_0 informs such node that it has become DN.

DN1 does not met second condition to become proper DN so we have to continue. We have not have any candidate for DN yet, so we set M_{best} to coordinates of DN1. Algorithm continue in second mode until it reach node DN2. It compares stored coordinates of M_{best} to itself and when DN2 is closer to destination it set new value of M_{best}. Node DN2 does not met second condition also. Therefore we continue for searching DN.

We can detect that one cycle is finished when we receive such packet at M_0 again. We operate in third mode since now.

Node M_0 sends a packet to the node M_{best} to inform that it has been selected as proper DN and should initialize

decision process. We switch to third mode (to reach selected DN), which is similar to second. Minor difference is that all candidates for DN compare its coordinates with M_{best} and if they match, they know that they have been selected as proper DN.

D. Left-hand rule application

Left-hand rule works almost the same as the right-hand rule except the fact we select first neighbour clockwise about the vector $destination \rightarrow M_0$ or $Node_{previous} \rightarrow Node_{now}$. The left-hand rule has the same ability to go around the hole as the right-hand rule, but it goes in opposite direction.

We mention one situation when right-hand rule can fail. Then it is useful to try to go in opposite direction with use of left-hand rule. Such situation is generalized version of a problem "spike" that we found in [12] and it is shown on Fig. 10. We send a packet to the left side (node M_1) and we expect that it makes one round around the hole with use of right-hand rule. We expect that packet will be received from the right side at node $M_1^{'}$ (the node established with use of left-hand rule to vector $destination \rightarrow M_0$).

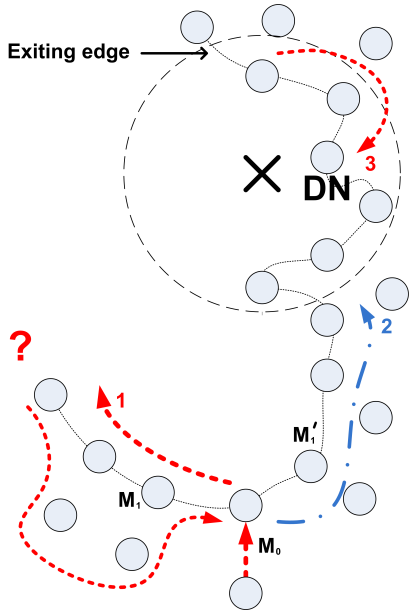

Fig. 10. Right-hand rule do not solve a problem at all times. Sometimes left-hand rule can solve the problem.

We modified the condition when we drop packet due to destination is unreachable. When M_0 receives packet for the second time it does not check if the target was found only (GPSR drops such packet immediately), but it also check if it came from left side (established as M_1 with right-hand rule about $destination \rightarrow M_0$) or from the right side ($M_1^{'}$ with left-hand rule). If it is received from right side, packet is dropped. Otherwise we have to continue.

We define fourth mode, which will be the same as the second except it uses left-hand rule. It is activated when second mode fails to find destination and M_0 receive such information from M_1. If such information is received from other node than M_1 we drop packet. Lets consult the usage of fourth mode in Fig. 10, which is labelled with arrow denoted with 2. We forward packet using left-hand rule and since we reach target area we check if the next skip will bring us out of it. Last node that is inside target area switch algorithm from fourth mode to the final fifth mode. It selects M_{best} node as DN directly.

We found that holes, which causes that we have to switch to fourth mode are large. Go around this edge consumes a lot of energy. Therefore we rather select the best candidate from this side and we do not bother about the rest. Such protuberance of network can be found on the outer edge of the network. If we do not stop algorithm at this moment, it can cause forwarding packets through all nodes at the outer edge of the network.

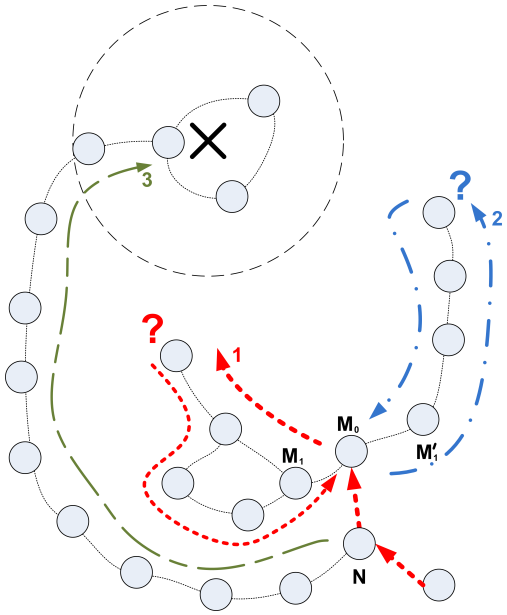

Fig. 11. Sometimes right-hand rule do not solve problem and left-hand rule either. We select second neighbour counterclockwise about the vector $destination \rightarrow M_0$ to become a new M_0.

Extreme example of protuberance of network is shown on Fig. 11. Protuberances are on both sides. From the left side we do not find destination and we are informed about it from M_1. From the right side we do not find destination either and we are informed about it from $M_1^{'}$. We mark the node M_{2_last}, which has used the mode 2 for the last time. It is the node that inform M_0 about the fact that destination is not reachable with use of right-hand rule. It is node $M_1 = M_{2\_last}$ in this example. In this extreme example we select first neighbour counterclockwise about the vector $M_0 \rightarrow M_{2\_last}$ to become a new M_0. Node that becomes a new M_0 is denoted as N. Node N switch back to second mode and tries to reach destination from this location.

Such situation does not happen very often, but we must handle it correctly. Otherwise we drop packets that can be delivered.

IV. Trust evaluation to specific node

In our system, each node evaluates the trust value of their direct neighbors in a few specified contexts. We denote the trust value of node $[a, b]$ to the node $[x, y]$ in the specific

context as $\tau^{context}_{[x,y]}$ and it is evaluated from interactions between nodes. The trust value is evaluated in 4 contexts: data validity of 3 different sensors $\tau^{s1}_{[x,y]}, \tau^{s2}_{[x,y]}, \tau^{s3}_{[x,y]}$ and we also evaluate the link quality between nodes $\tau^{link}_{[x,y]}$. We will follow with description of how the trust values are evaluated.

A. Sensor context

We make expectation that two close by nodes are placed at the same environment and data sensed from sensor should have similar values [7]. Sensed value at specific $node$ from specific sensor s at the time t is labelled as $val^{s,t}_{node}$. Nodes that we are using have 10-bit ADC converter so the values are in the range $< 0, 1023 >$.

Each node $[x, y]$ periodically broadcast measured data from sensors and neighbors $[a, b]$ that receive such message compare data from their own sensors to that from received message. We evaluate such interaction in each sensor context and the output value will be also in the range $< 0, 1023 >$. We have set minimal distance of nodes to be 1 meter and the divisor in Eq. 2 is then not equal to zero. Maximal 1023 value represents, that sensed values of nodes $[x, y]$ and $[a, b]$ are exactly same. Lower values mean that there is difference in sensed values. Significant difference could be caused with broken sensor. The interaction at the time t is evaluated as a real number with Eq. 2.

$$i^{s,t}_{[x,y]} = 1023 - \frac{abs(val^{s,t}_{[x,y]} - val^{s,t}_{[a,b]})}{MAX(1, distance([x, y], [a, b]))} \quad (2)$$

We take particular interactions and we will compute trust value to all sensor context. We showed how we evaluate each interaction in previous equation. There could be more equations how to transform sequence of evaluated interactions to the trust value.

At the first attempt we have chosen simple mean of all (we have n interactions in the past) interactions in specific context as a trust value:

$$\tau^{s}_{[x,y]} = \mu^{s}_{n} = \frac{1}{n}\sum_{j=1}^{n} i^{s,j}_{[x,y]}$$

We have chosen this function because of it can be converted [6] to sequential algorithm. The main advantage of sequential algorithm is that we don't have to store the interaction history. It is very important in the wireless sensor networks. Each sensor node has limited amount of memory, so storing whole interaction would not be possible. It would also influence the energy consumption of node. Each time we get new interaction we would have to recount the trust value. Widely used sequential variant to count simple mean sequentially:

$$\tau^{s}_{[x,y]} = \mu^{s}_{n} = \frac{1}{n}\sum_{j=1}^{n}(i^{s,j}_{[x,y]})^2 - (\frac{1}{n}\sum_{j=1}^{n} i^{s,j}_{[x,y]})^2$$

As we can see we have to store only 3 variables (two summaries and number of interactions) to count this algorithm sequentially. This aspect to be able to count trust value sequentially is very important and significantly reduce resource consumption. As our first attempt suited our needs we have found out that our model had problems with some situations. When some node start its work it has not any problems usually. All sensors work well at the beginning of its lifetime, bus as time goes on some sensor can failure. There can be various reasons why sensor failure. Nodes are usually placed to outdoor environments, where can occur some water leak, physical damage or a high temperature can damage a sensor.

The problem of using simple mean to compute trust value is that it reflects such damages of sensors slowly. If some damage occurs then trust value of neighbors nodes would be high for a very long time and decrease slowly. We have decided to replace simple mean with exponentially-weighted mean [6]. The equation for exponentially-weighted mean is:

$$\tau^{s}_{[x,y]} = exp\_\mu^{s}_{n} = a^n i^{s,0}_{[x,y]} + \sum_{j=1}^{n} a^{(n-j)}(1-a) i^{s,j}_{[x,y]}$$

As we can see, all interactions are weighted. We will describe sequential variant later. The constant a is the real number $0 < a < 1$ and it affects the weights of interactions. For our needs, the value of a affects how fast it will forget old interactions and learn new inputs. Weights $w_{n,j}$ will be:

$$w_{n,j} = \begin{cases} a^n = a * a^{(n-1)} & \text{if } j = 0, \\ a^{(n-j)}(1-a) & \text{if } 1 \leq j \leq n. \end{cases}$$

In the paper [6] there is a proof that weights are normalized $\sum_{j=0}^{n} w_{n,j} = 1$. We recommend to set strictly $a < 0.5$, because for $a > 0.5$ there can be problems for our purposes. In our situation $w_{n,j} < w_{n,j+1}$ should be satisfied. In the case of $a > 0.5$ there will be:

$$\begin{aligned} a * a^{(n-1)} &> a^{(n-1)}(1-a) \\ w_{n,0} &> w_{n,1} \end{aligned}$$

For some small number of interactions it can cause strange situation in computing trust. For example for 4 samples ($n = 3$) and $a = 0.9$ the weights are in the table II:

j	0	1	2	3
$w_{3,j}$	0.729	0.081	0.09	0.1

TABLE II
WEIGHTS FOR EXPONENTIALLY-WEIGHTED MEAN WITH $n = 3$.

We can see in the table II, that the first interaction ($j = 0$) will have most impact on result. This misbehavior would have less impact on result as the time goes on and we have more and more interactions between nodes. There will become more significant $\sum_{j=1}^{n} a^{(n-j)}$ part of equation. But as we demonstrated the value $a > 0.5$ could cause problems at the early time of trust evaluation.

Exponentially-weighted mean equation can be transformed to its sequential variant [6] that we use at this time, which will look like:

$$\begin{aligned} exp\_\mu^{s}_{0} &:= i^{s,0}_{[x,y]} \\ exp\_\mu^{s}_{n} &:= exp\_\mu^{s}_{n-1} + (1-a) * (i^{s,n}_{[x,y]} - exp\_\mu^{s}_{n-1}) \end{aligned}$$

As we know that values of inputs are in the range $< 0, 1023 >$ then the result will be in the same range also. Then we have to store only one variable $exp\_\mu_n = \tau^{s}_{[x,y]}$, because of we are able to detect if we have any interaction history

already. We set $exp\_\mu_n^s = -1$ at the beginning, which means we do not have any interaction from the past. We have found that value of a set around 0.35 suits our needs best.

B. Communication context

Besides of fact we have 3 contexts for 3 different sensors, we have also the context of communication $\tau_{[x,y]}^{link}$ through the radio. One thing that influence the link quality is the distance to the neighbor node. But there could be obstacles in the path of signal from node x to node y. Our nodes support to detect strength of signal when it receive message. Such value is named RSSI (Received Signal Strength Indication) and in the TinyOs it is represented with 16-bit number. We work with different types of nodes platforms. Radio modules of different platform could provide such RSSI values in different range. For example MICAz platform returns values in the range `<FF D2, 00 2B>` hexadecimally, while Iris platform that returns RSSI value in the range `<00 00, 00 1C>` hexadecimally. We normalize such values to the interval $< 0, 255 >$. Best link quality is represented with the highest value 255 and worst link quality with a value 0. It is better to normalize values, because of our WSageNt platform can be loaded to different type of nodes and the meaning of values remains the same.

As we presume that nodes are placed to the environment statically and they do not move in the time, we should take into account that there could be mobile obstacles that has impact on RSSI value during the time. We should also note that value of RSSI is not precise enough and it oscillate around some mean value. Thus the RSSI value at the first time can be different to the second time measurement even if the node placement is statical and so do obstacles on the path. There can occur a very long time without a communication between two nodes and because of that one node should initiate some communication after some period of time. This is done for purpose to find out if there has not come any mobile obstacle. Each measurement starts with broadcast to its neighbors to answer back and it will listen to echoing messages. Therefore our WSageNt platform store information about their neighbors from sequence of 10 measurements of RSSI value. Simple mean value from the sequence is computed for and it is attached into the neighbors table. We just note that we restricted maximal number of nodes in the neighbors table to 20. TinyOs has to have all variables to be defined statically. The size of neighbors table has to be set manually, but for our purposes is maximum of 20 neighbors enough. The size of table can be changed in the source code.

We label the mean value of last measurement sequence as $rssi\_\mu_{last}$. RSSI value is not only one parameter we use for establishing $\tau_{[x,y]}^{link}$. We take into account latency of the packet transmission in milliseconds. The exponentially-weighted mean $latency\_exp\_\mu$ of the latency is counted only on successfully transmitted packets.

Other parameter is the level how many of packets has been transmitted successfully. Packet loss is pretty common in wireless sensor networks. Input values are then 1, which means that packet has been transmitted successfully and 0 that there is a packet loss. Exponentially-weighted mean with $a < 0.5$ prefer new values over the old one. Thus it is not well suited for this purpose, because of output values would oscillate too much. We rather set the $a > 0.5$, but first n samples are counted via simple mean. After n samples we start count the packet loss level with exponentially-weighted mean algorithm. For value $a = 0.9$ there is $w_{n,0} < \frac{1}{n}$ for $n \geq 34$. We have set value of $n = 35$, which provides that start phase of counting level of packet loss should be stabilized. We label the level of packet loss with $loss$ and its range is $< 0, 1 >$.

The trust to the link level $\tau_{[x,y]}^{link}$ is then computed using Eq. 3.

$$\tau_{[x,y]}^{link} = rssi\_\mu_{last} * \frac{1}{latency\_exp\_\mu} * loss \tag{3}$$

If we consider that $latency\_exp\_\mu > 0$ we conclude that value of trust to link is $\tau_{[x,y]}^{link} \in < 0, 255 >$. We just note that the time between initiating broadcast sequence to get $rssi\_\mu_{last}$ should be considerably long. Otherwise it would consume energy rapidly.

V. Selecting best node for mobile agent

In some cases there is important for mobile agent to stay at selected node for a quite long time. Such stay of agent at node can consume a lot of energy. Our model is able to reflect such needs of agent also and select node with most of energy. We label the remaining energy of node $[x, y]$ at time t as $Energy_{[x,y]}^t$. For future computation we will need to set:

$$E_{[x,y]} = \frac{Energy_{[x,y]}^{now}}{Energy_{[x,y]}^0} \tag{4}$$

The value $E_{[x,y]}$ represents ratio of left energy to the energy at the beginning. The range of $E_{[x,y]} \in < 0, 1 >$.

We have set distance $d_{[x,y]}$ of node $[x, y]$ from desired location of agent previously. Now we set a few other parameters that will occur in final Eq. 5. First one is the average number of neighbors nodes and we label it as a avg_nb. It is important parameter, because the higher the number of neighbor is the more resources it takes to obtain all information about nodes. The second parameter is the number of hops from node $[a, b]$ to node $[x, y]$. It means number of retransmission needed to route packet from $[a, b]$ to $[x, y]$ and we label it as $hops_{[x,y]}$. The final Eq. 5 returns value $h_{[x,y]}$ which represent suitability of node $[x, y]$ for specific agent.

Each agent select set of properties, which will influence the decision process of agent placement. He has ability to select what sensors are crucial for its work or if it runs on target node for a long time and needs a lot of energy. Each agent initiate its movement with request for placement to the target area. This request also contains a vector of flags (or attributes) $\beta_f \in \{0, 1\}$, where $f \in \{E, link, s1, s2, s3\}$. Flag set to 1 means crucial aspect in the process of placement decision. For example if an agent needs sensor 1 and sensor 3 and does not bother about other contexts, it will set flag $\beta_{s1} = 1$ and $\beta_{s3} = 1$ and other flags remain set to zero.

$$h_{[x,y]} = \frac{1}{MAX(0.1,d_{[x,y]})} * \frac{1}{hops_{[x,y]}^{avg\_nb}} * * \left(E_{[x,y]}\right)^{\beta_E} * \left(\frac{\tau_{[x,y]}^{link}}{255}\right)^{\beta_{link}} * * \left(\frac{\tau_{[x,y]}^{s1}}{1023}\right)^{\beta_{s1}} * \left(\frac{\tau_{[x,y]}^{s2}}{1023}\right)^{\beta_{s2}} * \left(\frac{\tau_{[x,y]}^{s3}}{1023}\right)^{\beta_{s3}} \quad (5)$$

Using Eq. 5 we compute the h_{node} value for all nodes and then we try to find the $node$ with maximal value h_{node}. In real implementation it would be impossible to maintain trust values for all nodes in the network. Notice the $\frac{1}{MAX(0.1,d_{[x,y]})} * \frac{1}{hops_{[x,y]}^{avg\_nb}}$ part of the equation. As the distance $d_{[x,y]}$ of node $[x,y]$ from target area increases, the number of $hops_{[x,y]}$ increases also. These two parameters will limit maximal values of $h_{[x,y]}$ of distant nodes. We select one of the neighbor nodes (or the decision node itself) in most cases, so we decided that real implementation compute h_{node} value only for neighbors of decision node.

VI. Decision process

Decision process should select the best node in the specified area to emplace mobile agent. We should select from the decision node $[a,b]$ and all its neighbors.

In previous section we showed how the node $[a,b]$ is able to compute the $h_{[x,y]}$ value of its neighbors. The problem is how to compute value of $h_{[a,b]}$ itself. We presume that the node $[a,b]$ is able to communicate somehow, so we do not bother about the $\tau_{[a,b]}^{link}$. Such precondition should be satisfied, otherwise the node $[a,b]$ would be removed from neighbor table of its neighbors and could not be selected as a decision node. The problem could occur, when the ability to communicate change dramatically during the decision process. We omit such problem in this article since its occurrence is very rare and the solution would exceed the scope of this paper.

On the other hand particular sensor could be broken on the decision node $[a,b]$. As a result the node $[a,b]$ has reduced trust to all neighbors, which sensors work properly. Fig. 12 shows how the target area looks like. We can see also that all nodes outside of the target area have $hops_{node} \geq 2$. It demonstrates how the trust value is reduced when a sensor of node $[a,b]$ failure. As a demonstration all nodes have constant input at the specified sensor. The input value is labelled below the name of each node. Failure of a sensor is simulated with reading the value 0 from a sensor.

We presume that nodes are started at similar time and have the same period of broadcast. As the evaluation of interactions are done in almost the same time, the trust value of node $[a,b]$ to the node $[x,y]$ is almost identical to the trust value of node $[x,y]$ to the node $[a,b]$. There are just slight differences at that values.

A. Voting mechanism

From the Fig. 12 the node $[a,b]$ is in strange situation. It does not trust to anybody and nobody trust to it. If we try to set $h_{[a,b]}$ as a average trust of all neighbor nodes to the node $[a,b]$ it will not solve anything. All nodes would have the value h_{node} too low and similar to decide, which node is best for the agent.

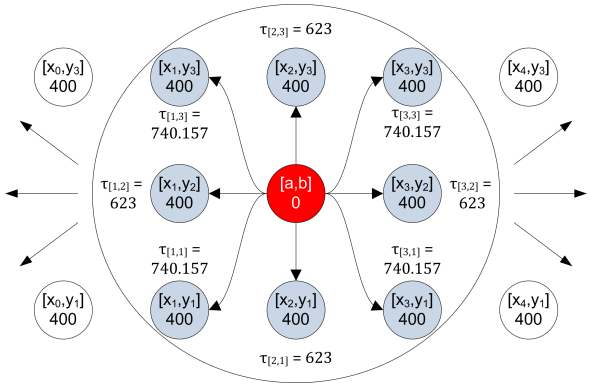

Fig. 12. Trust value of decision node $[a,b]$ to its neighborhood. Decision node has the sensor failure.

We rather implemented a vote mechanism that solve such situation well. The idea is not to compute $h_{[a,b]}$ itself at the decision node $[a,b]$. We rather make votes and listen if other nodes select the decision node as the best candidate.

Decision process is initiated from decision node. As we mentioned already we vote over the nodes, which are in direct neighborhood or we select decision node itself. All the nodes who vote, have to know who are the candidates to vote. On Fig. 12, there is a node $[x_4,y_3]$, which is a neighbor to the node $[x_3,y_3]$ (it is one of the node, who vote). As the node $[x_3,y_3]$ vote, it should filter off all neighbors, which are more than 2 hops away from decision node, including the node $[x_4,y_3]$. Therefore the decision node start the process with broadcast message. This message include the list of candidates (all nodes in the neighborhood of decision node and the decision node itself), so all voting nodes know, how to filter its neighbors.

All nodes select its best candidate (from neighbors, who are in list) and send its proposal to the decision node. Decision node counts all answers and selects the node with most proposals as the winner. Winner of the decision process is informed that it should emplace the agent and it start the transfer of the agent.

For better demonstration how the voting mechanism work we have prepared two examples. First one demonstrates the situation, when a sensor fails at the decision node itself. Fig. 13 represents such situation. For easier reading of Fig. 13 and 14 we simplify the evaluating function of its neighbors to $h_{[x,y]} = \tau_{[x,y]}^{s1}$.

Fig. 13 demonstrates the situation, when decision node has broken sensor. It is represented with reading 0 value from a sensor. All nodes in the neigbourhood of $[a,b]$, has reduced trust to the $[a,b]$ in the comparison to other neighbors. For example for the node $[x_1,y_3]$ the $h_{[x,y]}$ will be:

$$[x_1,y_3] \rightarrow \begin{cases} h_{[x_2,y_3]} = 1023 - \frac{abs(405-435)}{1} = 993.0 \\ h_{[x_1,y_2]} = 1023 - \frac{abs(390-435)}{1} = 978.0 \\ h_{[a,b]} = 1023 - \frac{abs(0-435)}{\sqrt{2}} \simeq 715.409 \end{cases}$$

It is clear, that the best candidate of node $[x_1,y_3]$ is $[x_2,y_3]$.

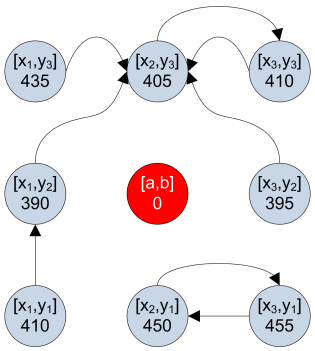

Fig. 13. Demonstration of vote mechanism. Decision node has sensor failure.

The situation is similar, when other voting nodes select their best candidates. They rather select other candidate than the decision node $[a, b]$, because of its reduced trust value.

Lets consider a situation when sensor of decision node reads proper value. Fig. 14 demonstrate a situation when the decision node reads similar value from sensor as its neighbors.

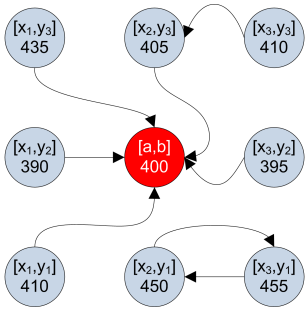

Fig. 14. Demonstration of vote mechanism. Sensor of decision node works properly.

In this example for the node $[x_1, y_3]$ the $h_{[x,y]}$ will be:

$$[x_1, y_3] \rightarrow \begin{cases} h_{[x_2,y_3]} & = & 1023 - \frac{abs(405-435)}{1} = 993.0 \\ h_{[x_1,y_2]} & = & 1023 - \frac{abs(390-435)}{1} = 978.0 \\ h_{[a,b]} & = & 1023 - \frac{abs(400-435)}{\sqrt{2}} \simeq 998.251 \end{cases}$$

The node $[a, b]$ is selected as the best candidate of the node $[x_1, y_3]$. Similar situation happens with most of other voting nodes.

VII. Conclusion

We demonstrated that Agilla platform has a few drawbacks when we want to use our nodes in the real environment. We have focused on greedy variant of geo-routing algorithm that Agilla uses. We have described our modifications to the original GPSR algorithm, which is superior in the comparison to greedy variant. Proposed modifications allowed us to use vague addressing and to reach target area.

There were demonstrated that our voting mechanism suits our needs to distinguish, if the decision node has broken sensor or not. We have demonstrated such behavior on simplified function of $h_{[x,y]} = \tau^{s1}_{[x,y]}$. We have mentioned in section III that the decision node is nearest node to the preferred location of agent. If we look at Eq. 5, we can conclude that if the decision node works properly, it should be selected as a winner of decision process in most cases. It is done because of the $\frac{1}{d_{[x,y]}}$ part of Eq. 5. We demonstrated on example that problems with the decision node that can influence agent stay should cause a relocating of agent to some other node.

Acknowledgement

This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070) and by grant BUT FIT-S-11-1.

References

[1] Fok, C.-L., Roman, G.-C., and Lu, C. Mobile Agent Middleware for Sensor Networks: An Application Case Study. In Proceedings of the 4th international Symposium on Information Processing in Sensor Networks (IPSN05), April 2005.

[2] C. Muldoon, G.M.P. O'Hare, R. Collier, M.J. O'Grady, Agent Factory Micro Edition: A Framework for Ambient Applications, LNCS vol. 3993/2006, p. 727-734, Springer, 2006, ISBN 978-3-540-34383-7.

[3] Tapia at el.: HERA: Hardware-Embedded Reactive Agents Platform, In: Highlights in Practical Applications of Agents and Multiagent Systems, Springer Berlin, 2011, ISBN 978-3-642-19916-5.

[4] F. Zboril jr, J. Horacek, P. Spacil: Intelligent Agent Platform and Control Language for Wireless Sensor Networks, In: Proceedings of 3rd EMS, Athens, GR, IEEE CS, 2009, ISBN 978-0-7695-3886-0.

[5] F. Zboril, P. Spacil: Automata for Agent Low Level Language Interpretation, In: Proceedings of UKSim 2009, Cambridge, GB, IEEE CS, 2009, s. 6, ISBN 978-0-7695-3593-7.

[6] Tony F.: Incremental calculation of weighted mean and variance, University of Cambridge, 2009.

[7] Probst M., Kasera S.: Statistical Trust Establishment in Wireless Sensor Networks, In: ICPADS '07 Proceedings of the 13th International Conference on Parallel and Distributed Systems, IEEE Computer Society Washington, USA, 2007, ISBN: 978-1-4244-1889-3.

[8] Karp B., Kung H.: GPSR: greedy perimeter stateless routing for wireless networks, In: MobiCom '00 Proceedings of the 6th annual international conference on Mobile computing and networking, ACM New York, USA, 2000, ISBN: ISBN:1-58113-197-6.

[9] Karp B.: Geographic Routing for Wireless Networks, PhD thesis, Harward University, October 2000.

[10] Wooldridge, M: An Introduction to MultiAgent Systems, 2nd Edition, 2009, ISBN 978-0-470-51946-2.

[11] TOUSSAINT, G. T.: The relative neighborhood graph of a finite planar set,. Pattern Recognition 12, 4 (1980), 261268.

[12] del Mar Alvarez-Rohena, M., Eberz, C.: Implementation and Analysis of GPSR: Greedy Perimeter Stateless Routing for Wireless Networks, IEEE, 2010.

[13] Horacek, J. and Zboril, F.: Mobile code placement in wireless sensor networks, In: Intelligent Systems Design and Applications (ISDA), 172-177, Nov. 2012, ISSN 2164-7143.

Jan Horacek (born 1985, Jihlava) is a Ph.D. student at Brno University of Technology, Czech Republic. He obtained his Master's degree In 2009 at Faculty of Information Technology on University in Information Technology. His research is aimed to wireless sensor networks, artificial intelligent agents and routing protocols.

Frantisek Zboril (born 1974, Olomouc) is an assistant professor at Brno University of Technology, Czech Republic. He obtained his Ph.D. In 2004 at Faculty of Information Technology of this University in Information Technology. His major interests includes artificial agents, their application in the area of modelling of distributed systems and applications for wireless sensor networks.

Jakub Zak (born 1985, Dacice) is leading author of this paper. This author obtained his Master's degree in 2009 at Faculty of Information Technology. Author is now student of Ph.D. study program on Faculty of Information Technology at Brno University of Technology, Czech Republic. He is studying his final year of his Ph.D. studies. Among his interests belongs all related to agent oriented technologies and modelling systems. Additionally author is interested in methodologies that deal with agent systems creating.