Passive target tracking: Application with mobile devices using an indoors WSN Future Internet testbed[1]

Karagiannis Marios [2], Chantzis Konstantinos [2], Nikoletseas Sotiris [3] and Rolim José [2]
[2] Centre Universitaire d' Informatique, Geneva, Switzerland. E-mail: {marios.karagiannis, konstantinos.chantzis, jose.rolim}@unige.ch
[3] Research Academic Computer Technology Institute (CTI) and University of Patras, Greece. E-mail: nikole@cti.gr

***Abstract*—In this paper, we describe the implementation of applying and testing the "Lightweight Target Tracking using Passive Traces algorithm" [1] on a FIRE wireless sensors testbed located in the Theoretical Computer Science/Sensors Lab in Geneva, Switzerland. We provide information about the hardware installation and configuration, the changes we did to the algorithm to adapt it to a real testbed as well as the tools we implemented to operate the network and receive feedback from the algorithm's operation. Finally, we discuss the performance evaluation findings of our implementation.**

I. Introduction

In this work we study the problem of tracking an asset that can be mobile or not in a small office environment, where a wireless sensor network is deployed. We take an experimental approach, by building a testbed in the Theoretical Computer Science(TCS)/Sensor Lab at the University of Geneva, using off-the-shelf sensor nodes, mobile robots and other devices. We then implement a known tracking algorithm in this testbed and evaluated its performance and usability under real conditions.

Our decision to apply a lightweight target tracking algorithm in a confined office space has many real-life applications. Such applications could be tracking people in a smart building, with variable resolutions. Emergency services could utilize such a system in cases where people must be located within a building (natural disasters, fire, earthquakes etc.). On the other hand, valuable assets can also be tagged and tracked within a smart building, in order to locate them or even track their usage history. For example, a laptop in such an environment can be used by several groups in different floors of a smart building and later generate presence statistics in each group. Tracking can also be used for security applications, where the absence of presence of a valuable asset, or its movement outside a predefined area can raise an alarm in a central control panel. The same principle can also apply for movement of unauthorized persons within secure areas.

The implementation details we will present use technologies developed in the context of two Seventh Framework Programme EU projects, Wisebed [2] and Hobnet [3].

[1]This work has been partially supported by the IST Programme of the European Union under contract number ICT-2008-224460(WISEBED) and contract number ICT/FIRE-STREP257466 (HOBNET)

The aim of the Wisebed project (June 2008-May 2011)[2] is to provide the framework needed to build multi-level infrastructure of wireless sensor network testbeds that are connected, even though they can be heterogeneous and located in remote locations in respect to each other. Such infrustructure allows large scale research experiments to be scheduled for logical large scale networks that are well organized and dynamic. The Wisebed project has provided technology that allows connection between different vendor hardware in different geographic locations, using Virtual Links [4], an XML scheme for describing wireless sensor networks topologies and data alike, called WiseML [5] which is based on GraphML [6]. Finally, a development framework that supports development on WSN sensors of multiple vendors was introduced, called Wiselib. Wiselib allows unified high level wireless sensor network development and provides a library of algorithms categorized in thematic categories, like localization, geographic routing etc.

The Hobnet project (June 2010-May 2013)[3] is a project that uses existing FIRE technologies and platforms for Future Internet applications focused on automation and energy efficienty for smart/green buildings. The main research areas of Hobnet are the use of IPv6/6LoWPAN infrastructure, 6lowApp standarization, new algorithmic models that fit the objectives of the project and are scalable and energy efficient, rapid development and integration for building management and support for deployment and monitoring of the applications on existing FIRE testbeds.

II. The experimental testbed

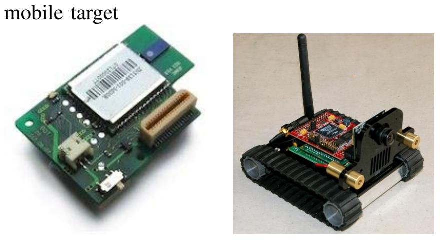
A. Hardware

In the testbed we setup for the needs of our tracking algorithm, we used hardware nodes from coalesenses GmbH, named iSense [7]. The iSense nodes are modular, depending on the needs a specific application. We used the iSense Core Modules for each node in our network. The Core Modules (fig. 2(a)), use 802.15.4 Zigbee compliant radio, providing a 250kbit/s transfer rate and use a 32 Bit RISC Controller running at 16MHz. They feature an integrated ceramic antenna, the transmission power of which can be programmatically controlled. In our testbed, we set the transmission power to -6dBm to avoid over-connectivity due to the limited space.

In each room of our testbed, we have installed an Atom based PC, acting as a controller for the nodes of that room. All the servers use the Atom D550 dual core processor, with 2GB RAM which provides enough power to run the applications needed, but with limited electrical power usage, therefore generating less heat and noise for around the clock usage. In all 5 of our servers, we installed the Ubuntu 10.10 linux distribution. Each server runs the WISEBED Testbed Runtime. All the technical reports describing the various parts of the runtime can be found in the website of the Wisebed EU project [2]. All the nodes are connected to the server of that particular room using 10m USB cables (fig. 2). We chose to setup the network using this configuration for two main reasons.

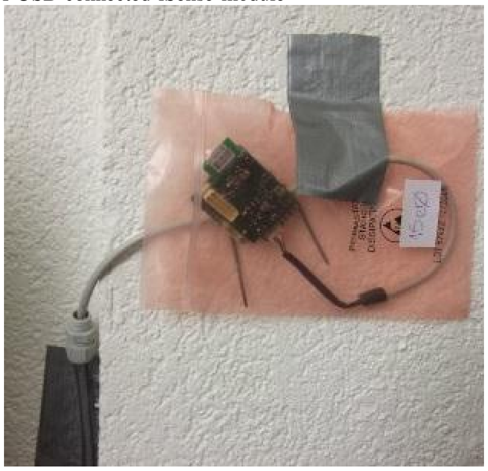
We didn't want to use the wireless medium for anything else than the operation of the tracking algorithm. Control and reporting messages, not related with the algorithm operation itself are being sent and received using the USB cables, thus leaving the wireless medium available for the algorithm. Of course, it was not possible to reserve the wireless medium exclusively, since the same 2.4GHz frequency used for the Zigbee compatible iSense hardware, is being used in 802.11, but eliminating one factor was a good enough reason to use USB cables. Also, while the iSense hardware supports one-to-many wireless firmware flushing, we found that using the USB cables directly connected to the servers, allowed for faster and less error-prone mass firmware flushing for our nodes.

Fig. 1. The mobile target



(a) The iSense core module (b) The Surveyor SRV-1 robot

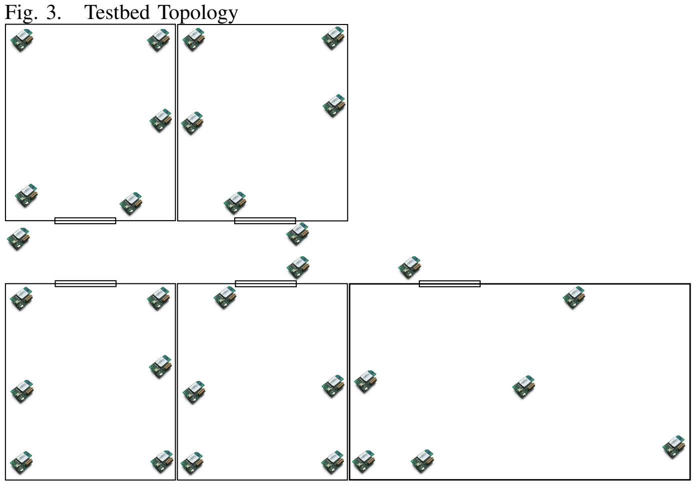
Fig. 2. A USB connected iSense module



B. Topology

The topology of the network is shown in figure 3. We feel that the choice we made to place multiple nodes in each rooms gives us the choice to use multiple tracking resolutions. If the desired resolution is a per-room tracking, one or two nodes would also be sufficient per room, but using more gives us the ability to use different reported locations to fine tune the position in a room, achieving much higher resolution. Details about this mechanism are described in detail in subsection IV-B.

Fig. 3. Testbed Topology



C. Mobility

Target tracking is better exploited with mobility scenarios. Tracking could in theory mean the in-network process of finding the target's last location, whether or not this locations has changed, but having a mobile target makes better sense. In our setup, we used a Surveyor SRV-1 robot [8]. The SRV-1 is a Open Source Wireless Mobile Robot which uses a 1000MIPS 500MHz processor, a digital video camera with resolution and WLAN 802.11b/g networking on a quad-motor tracked mobile robotic base (fig. 2(b)). In out setup, we attached an iSense node with an iSense Rechargeable Battery Module with a security sensor module. This gave us access to a 3 axis accelerometer. We used the accelerometer data to try to figure out when the robot was moving by comparing our accelerometer with a preset threshold on all 3 axis. Although in theory moving with a constant speed would give an acceleration of 0, in practice we could use the vibrations and variations in speed to make this decision with relative ease. Apart from filtering raw accelerometer data, the iSense platform accelerometer can be configured to generate interrupts on movement, direction change or free fall.

By using the moving-or-not data with a timeout after the robot stopped moving, we made could start sending beacons only when we figured that the position of the target has changed. After stop moving, we allowed the target to keep sending beacons for a few seconds, in order for our visualization system to get enough nodes weights to pinpoint the position with greater accuracy (see IV-B).

The topology of the network was designed on top of an already in-use setup of offices and corridors. We wanted to apply a network topology that would come close to real-life use as much as possible, and ended up with an arbitrary mesh topology; this topology assumes a rather weak model, however allowing multi-hop routing with good scalability and robustness in medium and large size networks. We dismissed a completely controlled aproach, such as a regular grid, from the beginning as non-realistic in a real setup.

D. Networking

The iSense platform uses a ZigBee compliant radio [9]. The ZigBee specification operates in the unlicensed 2.4GHz wireless band according to IEEE 802.15.4. It is a frequency agile solution covering 16 channels with power saving mechanisms and utilizes the industry standard AES-128 security scheme. ZigBee provides standards-based, reliable wireless data transfer. The main advantages of ZigBee is that it has been designed for constrained (battery, radio, size, etc.) wireless devices, it supports interoperability, it is flexible/reconfigurable, reliable and robust. In this sense, the ZigBee standard compliance in our hardware, allowed us to focus on the algorithmic part of our implementation as well as higher level issues that arose.

III. The tracking algorithm

The tracking algorithm we applied on the Future Internet testbed, was based on [1].

The key element of this protocol is to passively track a target, with the use of traces that span efficiently throughout the WSN topology. The tracking can be broken up to the following states

1) Target detection: When a sensor node detects the target in its vicinity, it initiates a new trace of type initial. The $start\_time$ of the trace is set to be the time at which the node detected the target. Furthermore, the $start\_intensity$ is set to the value $max\_intensity$, which is the maximum value allowed by the protocol. At any point in time, the value of intensity of that target, can be queried from the node. The intensity is defined by the following simple time dependant formula: $intensity = start\_intensity(current\_timestart\_time)$, where current time is the time indicated by the sensor nodes clock at the time of the query. In other words, when a sensor node detects a target it initiates a trace of maximal intensity which value will diminish (linearly) with time.

2) Trace spread: The trace spreading component is a mechanism by which the sensor network spreads traces across the network in such a way that the tracking agent will be capable of using those traces to track down the target. As explained previously, the idea is to span as well as possible the whole network with a tree of degree two, rooted on the node which has detected the target. When a node ”n” detects the target, it chooses two nodes to which it will propagate the trace to from a candidate list. The candidate list is a subset of ”n” adjacency list comprised of all the nodes that are further away from a repulsion point. The repulsion point is the last (oldest) ancestor node of the path of the trace with grandparent being the oldest possible ancestor, is piggybacked by the trace and updated with every spread. The two nodes from the candidate list to which the trace is propagated to, are chosen according to two heuristics. One being deterministic (the furthest node from the repulsion point) and the other one being randomized. Each of these two chosen nodes will, in turn, choose two nodes, and so on.

3) Spread inhibition mechanism: Whenever a node is about to spread a trace, it queries its neighbours. If one of its neighbours already holds a trace of higher intensity than the trace about to be spread, then the node stops spreading this trace: the spreading is inhibited. More precisely, when a node holds a trace, it computes the intensity of its trace. Say the traces intensity has value x, for $0 \leq x \leq max\_intensity$. If the spreading was to happen, two nodes $c1$ and $c2$ would be chosen, and they would receive a new trace of intensity x minus a spreading penalty penalty, which is a constant parameter of the algorithm. If one of the neighbours of the node considering the possibility of spreading its trace currently holds a trace of intensity at least x penalty, then the spreading is inhibited. Otherwise, the nodes $c1$ and $c2$ are initialised with a new trace of intensity $xpenalty$ and if type spread.

4) Tracking: The tracking process is fairly simple. When a network user wants to locate the target, he initiates a tracking agent. Until the agent finds a trace which can be used to track down the target, the agent does a random walk in the communication graph. Once a node holding a trace has been found, the tracking agent greedily moves towards the node holding the highest intensity trace. (Recall that the intensity of a trace is higher when the trace is close to the root of the tree, and when it is fresh, in the sense that the original trace follows a recent detection). At some point, the agent will eventually reach a node holding a trace of type initial. When this happens, the agent can compute the time at which the sensor holding the initial trace detected the target and send a report to the user. However, it could very well be that another neighbour node has detected the target too, so the agent keeps moving towards nodes with ever increasing trace intensity: the agent tracks more up to date location information on the target. When the maximum intensity is reached the agent has to report back to the user. In our case, it will follow the gradients of the tracking node of the user (that also operates as a target node) performing an inverse tracking scenario.

This tracking algorithm can be used in combination with off-line optimized network design methods such as in [10].

IV. Implementation aspects

A. The reporting pipeline

The reporting pipeline is the procedure that feeds the visualization system with data, when they become available from the network. It consists of 3 components. The first component is the tracker firmware. The tracker firmware, after sending software agents which return with the initial trace position, forwards this position using a debug message to the serial port, via its USB cable. The second component is a plugin for the iSense iShell software written in Java. When data become available from the tracker firmware, it is received, encapsulated in a UDP datagram and forwarded to one or more predetermined IP addresses. Devices in IP addresses run the third component, which is the visualization software. This software was developed in C#, which can be compiled in the Windows OS using the native C# compiler, or Linux based systems using the Mono project compiler. The visualizer has a map of the physical space that the experiment is running, opens a UDP socket on a waiting thread and waits for new data to be forwarded from the Java plugin through this socket. Using this data, it uses a specific algorithm to estimate the target position. The reporting pipeline is shown on figure 4.

Fig. 4. Reporting

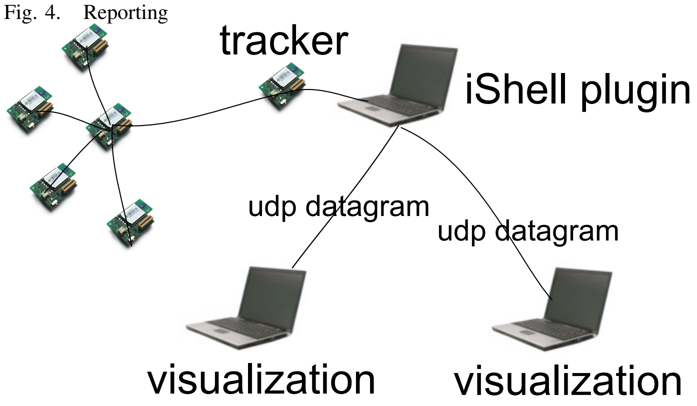
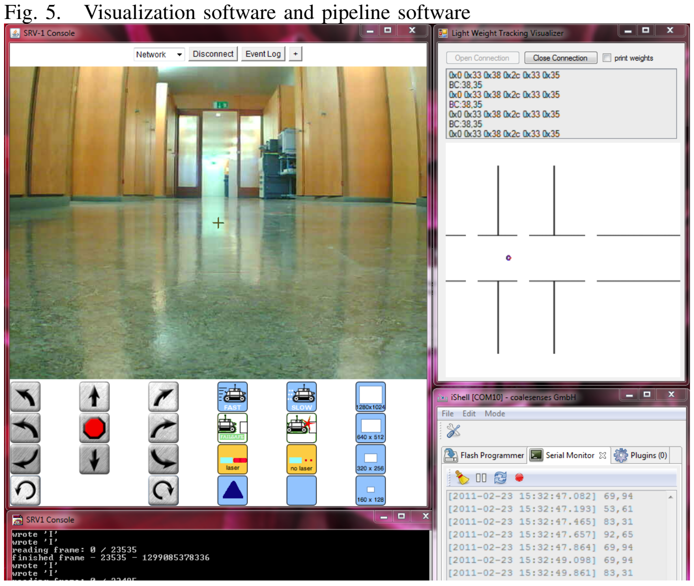


Fig. 5. Visualization and pipeline software



```
0x0 0x33 0x38 0x2c 0x33 0x35
BC:38,35
0x0 0x33 0x38 0x2c 0x33 0x35
BC:38,35
0x0 0x33 0x38 0x2c 0x33 0x35
BC:38,35
0x0 0x33 0x38 0x2c 0x33 0x35
```

```
wrote 'I'
wrote 'I'
reading frame: 0 / 23535
finished frame - 23535 - 1299085378336
wrote 'I'
wrote 'I'
```

```
[2011-02-23 15:32:47.082] 69,94
[2011-02-23 15:32:47.193] 53,61
[2011-02-23 15:32:47.465] 83,31
[2011-02-23 15:32:47.657] 92,65
[2011-02-23 15:32:47.864] 69,94
[2011-02-23 15:32:49.098] 69,94
[2011-02-23 15:32:49.861] 83,31
```

To materialize the pipeline presented above, we used the SRV-1 Console application to provide the necessary mobility for the robot carrying the target node, the Coalsenses iShell software to act as middleware between the network and our laptop and our Light Weight Tracking Visualizer software to visualize the reports in real time. This configuration was running on a laptop and a screenshot can be seen on figure 5. The SRV-1 Console provided us with a real-time streaming image of what the robot could see using its front camera, and a graphical user interface for sending movement commands to the robot's motors (top left on figure 5). The streaming image and the movement commands were sent using a 802.11b connection between the laptop and the SRV-1 robot, using an ad-hoc connection setup, bypassing the building's 802.11 infrustructrue. The visualization software can be seen on the top right of figure 5 while the iShell software can be seen on the bottom right of the same figure.

B. The visualization algorithm

One of the key elements of the algorithm implementation on the firmware level of the nodes is to avoid in-network processing and aggregation of the data. The agents spawn by a tracker will walk randomly, find a trace that a target has spread, follow it until they find the initial trace and then use reverse tracking to report the coordinates of the initial trace sensor. So the tracker will only get these coordinates from the system. If what we are after is per-room resolution, we are done here, since sensors are statically deployed in specific rooms. We could also be done by asking the agents to return just the ID of the sensors, since we know where each sensor is deployed. In our demo, we were interested in going a step further and use the data the agents provided to approximate an actual position for out target.

We achieved this using not only the last agents report, but also reports from previous agents. In order to do this, we assigned a counter to each of the sensors deployed in the network. We called this counter the sensor's weight w. Each time an agents returned a pair of coordinates, we increased the counter of the sensor corresponding to these coordinates. At any given time, we concluded that the position of the target was the average of the coordinates taking into account each sensor's weight:

$$x_{target} = \frac{\sum_{n=1}^{s} w_n * x_n}{\sum_{n=1}^{s} w_n}$$

and

$$y_{target} = \frac{\sum_{n=1}^{s} w_n * y_n}{\sum_{n=1}^{s} w_n}$$

where s is the total number of sensors and x_n is the x coordinate of sensor n, y_n is the y coordinate of sensor n and w_n is the weight of sensor n.

Every i seconds we decreased the weights for all sensors until they reached 0 in order to keep them relevant to the current position of the target. Also, we limited the maximum weight a sensor can have to $wmax$. For our experiments, after trial and error, we set $i = 3$ and $wmax = 5$, which we think were appropriate for the speed of our target and the response time of our network agents.

V. Performance measurements

We have tested our testbed setup using 4 different scenaria.

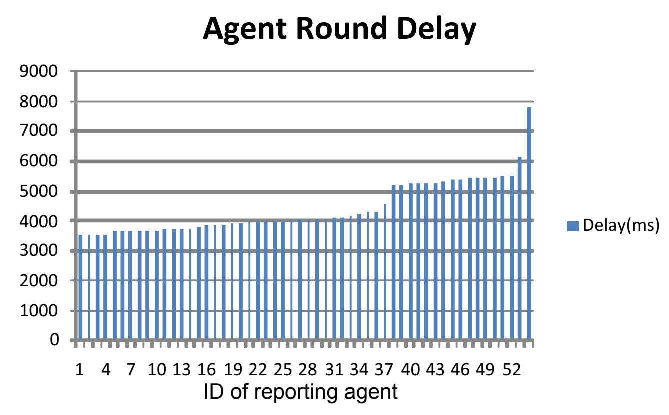
	Tracker	Target	Proximity
Scenario 1	Stationary	Stationary	Same Room
Scenario 2	Stationary	Stationary	Different Room
Scenario 3	Stationary	Mobile	Variable
Scenario 4	Mobile	Mobile	Variable

In all 4 scenaria, the target module was on an SRV-1 robot, controlled remotely via wi-fi, while the tracker was a laptop with an iSense module hooked up on one of its USB ports, running the software setup described in IV-A. In scenario 3, where the tracker was mobile as well, a human was physically carrying the laptop from room to room, walking normally and asynchronously with the movement of the target. The tracker was also waiting randomly in each room before moving to the next one. For easier comparison, in all scenaria and while the target was active, we recorded the agents spawned from the tracker and returned to it for a duration of 5 minutes.

A. *Scenario 1*

In this scenario, both the tracker and the target were stationary in one room. In figure 6 we can see that the return time is in about 66% of the times between 3 seconds and 5 seconds, while only 2 out of 55 agents took more than 6 seconds to return to the tracker. Due to the static nature of the experiment and the close proximity of the target and the tracker, these results are very stable and predictable.

Fig. 6. Scenario 1 - Agents Return Times(ms)



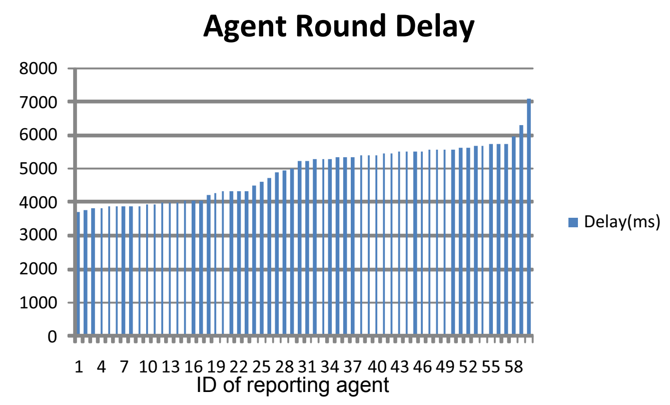
B. *Scenario 2*

In this scenario, both the tracker and the target were stationary but this time, they are positioned in different rooms. In figure 7 we can see that the return time is in almost all the samples between 3 seconds and 6 seconds, while only 2 out of 60 agents took more than 6 seconds to return to the tracker. The slight increase increase in agents return time is due to the increased number of hops the tracker agent had to travel in order to find the initial trace of the target which is further increase by the same amount for the reverse tracking (while looking for the tracker to report back). Due to the static nature of the experiment and the close proximity of the target and the tracker, these results are also very stable and predictable.

C. *Scenario 3*

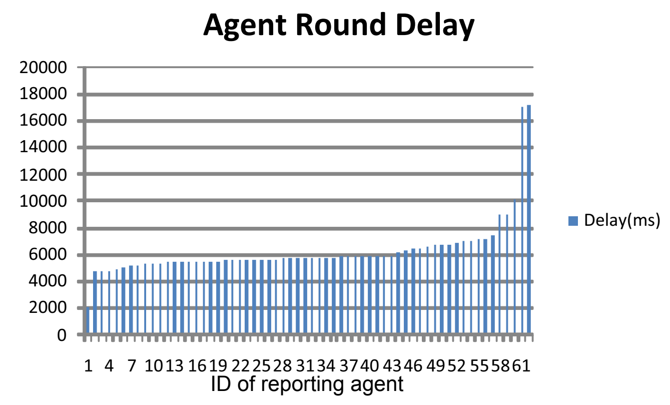
In this scenario, the tracker is stationary while the target is mobile. We have used a specific mobility pattern that covered all the network, moving the target module from room to room using the SRV-1 robot. In figure 10 you can see this mobility pattern. The target begun in room 4, at point S, moved to room 5, then to room 1, then to room 3, then to room 4 and finally stopped in room 2, at point E.

Fig. 7. Scenario 2 - Agents Return Times(ms)



In figure 8 we can see that the return time is in almost all the samples between 4 seconds and 8 seconds, while a small percentage of agents took more than 8 seconds to return to the tracker. Due to the mobility of the target, an increase in return times was to be expected, since old traces were overlaped by new traces in different locations, due to this mobility, causing the tracker agent to travel further away to locate the initial traces and report back. Even with target mobility, we believe the results are stable with a very small number of exceptions.

Fig. 8. Scenario 3 - Agents Return Times(ms)



D. *Scenario 4*

In this scenario, both the tracker and target are mobile. We used the same mobility pattern for the target as in the previous scenario that covered all the network, moving the target module from room to room using the SRV-1 robot. At the same time, we used a different mobility pattern for the tracker. This mobility pattern, which can be seen in figure 11, also covers all of the network, but in a different order and with different mobility speed, since the tracker was carried by a human with walking speed and random waiting time in each room. The tracker begun in room 4, at point S, moved to room 3, then to room 1, then to room 2, then to room 5 and finally back in room 4, at point E.

In figure 9 we can see that the return times for the agents were between 3 seconds and 12 seconds in most samples. A still small, but slightly larger than the previous scenario, percentage of agents took more than 12 seconds to return to the tracker. This is the most complicated scenario, with asynchronous mobility in both the tracker and the target, and higher return times were to be expected. When both the target and the tracker are mobile, the reverse tracking, after the agents has found the initial trace of the target, is equally affected by the tracker's mobility, in the same way the tracker time to find the initial trace is affected due to the mobility of the target. Even with mobility in both the tracker and the target, we believe the results are satisfactory and do not exceed practical uses, when near real-time tracking is needed as a requirement.

Fig. 9. Scenario 4 - Agents Return Times(ms)

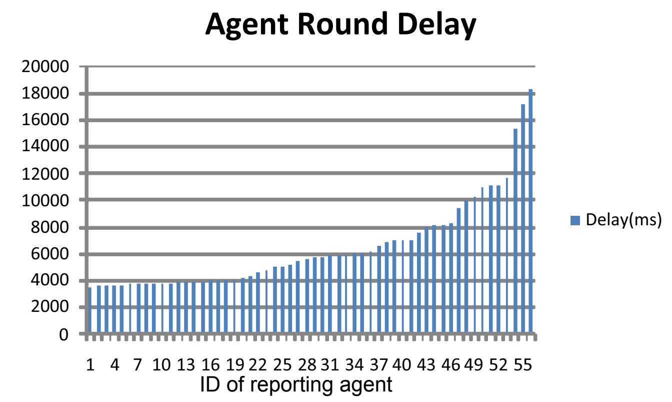
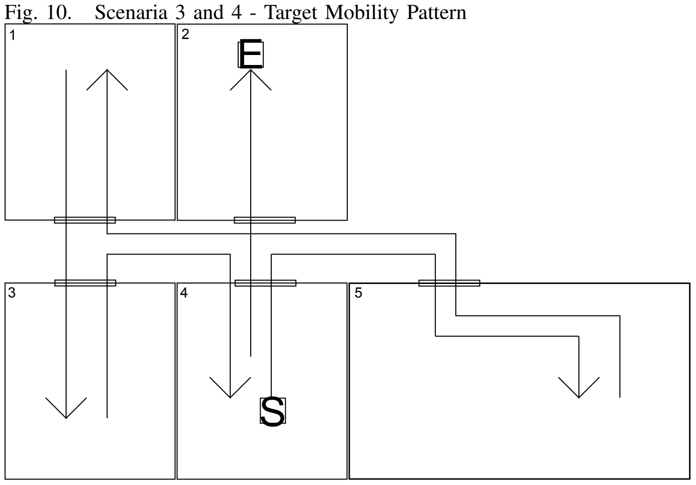


Fig. 10. Scenaria 3 and 4 - Target Mobility Pattern

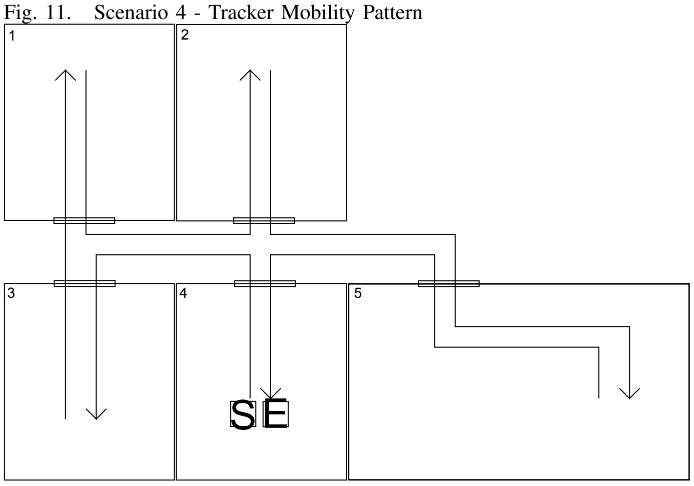


VI. Conclusions and Future Work

In this paper, we described the implementation details when applying and testing the "Lightweight Target Tracking using Passive Traces" algorithm [1] on a Future Internet wireless sensors testbed located in the Theoretical Computer Science/Sensors Lab in Geneva, Switzerland. We provided details on the testbed hardware and installation as well as the tools we implemented to monitor the network's function and behavior. Comparing the results we got from the running of the algorithm on the testbed we setup in our lab, initially we had mixed feelings. The algorithm has only been tested previously on a theoretical level, using simulation software that made some assumptions about connectivity and mobility models. We found out that some of these assumptions were not realistic, so specific additions and tweaks had to be made on the algorithm itself as well as other levels of the implementation, like the connectivity layer.

Our results prove that the application of the algorithm is feasible even with a network setup comprised of 20-30 nodes in a small office environment. Even with mobility on just the target or the target and the tracker modules, we showed that the reports that the tracker receives from the network are within reasonable time limits, that can be used for near real-time tracking and visualization. Static results showed even more promise, in case the requirements do not require mobile tracking. We believe that in future work, we can optimize these reporting times even more, coming closer to near instant real-time tracking using the same setup and the same algorithmic principles. We also plan to introduce multiple trackers and targets on the same network, which can be tracked independently, with minimum delays.

Fig. 11. Scenario 4 - Tracker Mobility Pattern



References

[1] A. Marculescu, S. E. Nikoletseas, O. Powell, and J. D. P. Rolim, "Efficient tracking of moving targets by passively handling traces in sensor networks," *GLOBECOM*, pp. 271–276, 2008.
[2] http://www.wisebed.eu/index.php/documents.
[3] http://www.hobnet-project.eu/.
[4] "Wisebed deliverable 2.2, section 7," http://www.wisebed.eu/images/stories/deliverables/d2.2.pdf.
[5] "Wisebed deliverable 4.1, section 3," http://www.wisebed.eu/images/stories/deliverables/d4.1.pdf.
[6] "The graphml file format," http://graphml.graphdrawing.org/.
[7] "coalesenses gmbh," Website, http://www.coalesenses.com/.
[8] "Surveyor corporation," Website, http://www.surveyor.com/.
[9] "Zigbee alliance," Website, http://www.zigbee.org/.
[10] S. Nikoletseas and P. Spirakis, "Efficient sensor network design for continuous monitoring of moving objects," *Theor. Comput. Sci.*, vol. 402, pp. 56–66, July 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1387358.1387506