Wireless Sensor Network, **2011, 3, 158-166**
doi:10.4236/wsn.2011.35018 Published Online May 2011 (http://www.SciRP.org/journal/wsn)

In-Motes EYE: A Real Time Application for Automobiles in Wireless Sensor Networks

Dimitrios Georgoulas, Keith Blow
Adaptive Communication Networks Research Group, Aston University, Aston Triangle, United Kingdom
E-mail: *dimitriosgeorgoulas@yahoo.com*
Received February 10, 2011; *accepted March* 29, 2011; *accepted April* 7, 2011

Abstract

Wireless sensor networks have been identified as one of the key technologies for the 21st century. In order to overcome their limitations such as fault tolerance and conservation of energy, we propose a middleware solution, In-Motes. In-Motes stands as a fault tolerant platform for deploying and monitoring applications in real time offers a number of possibilities for the end user giving him in parallel the freedom to experiment with various parameters, in an effort the deployed applications to run in an energy efficient manner inside the network. The proposed scheme is evaluated through the In-Motes EYE application, aiming to test its merits under real time conditions. In-Motes EYE application which is an agent based real time In-Motes application developed for sensing acceleration variations in an environment. The application was tested in a prototype area, road alike, for a period of four months.

Keywords: Wireless Sensor Networks, Middleware, Mobile Agents, In-Motes, Acceleration Measurements, Traffic Cameras

1. The In-Motes Middleware System Architecture

In-Motes [1] is an intelligent agent based middleware for wireless sensor networks (WSNs). In-Motes is based on Agilla [2] and Mate [3] middleware's by allowing users to inject agents inside the network and provides a high level architecture for the given agent community based on federated systems and behavioral rules produced by a parallelism of bacterial strains [4]. The middleware is written on a combination of nesC and Java programming languages and is applied on top of the TinyOS Operating System [5].

The In-Motes middleware can be defined as a mobile code middleware [6] that generates a flexible framework for deploying applications in wireless sensor networks. In-Motes Agent is a small computer program that is the fundamental actor of an In-Motes application which combines one or more instruction capabilities, as published in the instruction set, into a unified and integrated execution model for every node in the wireless sensor network. The In-Motes agent will have a specific agent identifier in order to be distinguished from similar agents that will co-exist locally in a node or globally inside the network at the same time. The In-Motes agents do not embed any level of learning or social capabilities, thus in the descriptive domain of a generic agent definition they pass only as individual mobile processes with pre-defined instructions that act on behalf of an end-user.

The In-Motes agents consists of four different mobile code categories that can co-exist at the same time inside the wireless sensor network according to the user needs and the specification of the deployed application [7].

The Facilitator category consists of In-Motes agents that are responsible for forming a "federation" of the active mobile code in the selected nodes. Their role is twofold, first they set up the communication protocol and secondly they are responsible for collecting all the results from the job In-Motes agents and forwarding them to the base station for analysis. Their life expectancy in the network is tightly bound to the life expectancy of the application. They are able to exchange messages with each other but are unable to take any readings from the nodes that hosted. They have the ability to provide a simple form of decision making based on their level of business. Thus, if a facilitator level of business is above 50% the query will be passed to the next available facilitator. Each facilitator In-Motes agent is divided into small packets of 41 bytes each, upon deployment in order to minimize message loss and deadlocks. They each consume 135 bytes of virtual memory from the In-Motes engine.

The Slave category consists of mobile code that is responsible for capturing the available nodes in the wireless sensor network. By the term capture we mean the ability to assign a predefined number of N nodes under the same facilitator In-Motes agent. Slave agents are practically clones of the facilitator agents and they do not provide any local decision making. After a successful capture of a node they report back to the facilitator agent and then die. Each slave agent being a facilitator clone consumes 135 bytes of virtual memory during its active period.

The Job category consists of mobile code that is responsible for carrying out the user requests to the wireless sensor network. Their role is to collect readings from the sensing devices of the hardware and their specification is tightly bound with the application. Thus, a job In-Motes agent could be reporting temperature, light or acceleration readings to the facilitator agent. They can report only one set of readings at the same time. Job agents are able to be transferred either by cloning or by migrating inside the wireless senor network based on the application and the available memory. Therefore, for large scale, complex applications which needed most of the memory resources, job agents are migrating while for simple applications they are cloning. The difference between cloning and migrating is based on how the code is transferred inside the wireless sensor network. Thus, a Job agent is migrating when the same code is visiting the predefined nodes alters their parameters, but it never stays resident in any of them, while with cloning, a Job agent creates multiple copies of its code that are transferred and stay resident in all the predefined nodes.

According with their status, defined as static or dynamic, In-Motes job agents are able to provide a simple level of local decision making. The term “static” describes In-Motes job agents which perform a single user request measurement and then die while “dynamic” describes In-Motes job agents which perform multiple measurements and respond to changes in user defined parameters. The dynamic In-Motes job agents consume 118 bytes of virtual memory and they migrate inside the system while the static ones 68 bytes and they clone inside the wireless sensor network.

The Fix category consists of mobile code that is used as a debugging tool for the wireless sensor network. Their role is to flush the memory of a single node in case of a problem such as buffer overflow or to flush the memory of the total number of nodes of the network. They are small in size, 25 bytes, and do not provide any local decision making.

The In-Motes architecture is divided in two layers [8]. The first layer consists of the In-Motes agents that were described above. Based on the fact that we could have one or more mobile codes active at the same time on the same node lead us to the need for a second layer that apart from the In-Motes engine would include a manager scheme for regulating issues such as context and reactions. Without this layer the In-Motes agents would have a loose hierarchy that would lead to confusion between their roles and responsibilities inside the wireless sensor network and also the system would consume unnecessary physical and virtual memory. Thus, the second layer consists of a facilitator manager, agent manager, rules manager, operation manager and an instruction manager **Figure 1**.

The In-Motes instruction set is based on those of Agilla and Mate. However, there are many modifications and differences in order to support the facilitator agent’s scheme and the tuple space operations [9].

The In-Motes communication protocol [10] is based on the federation communication scheme. A facilitator In-Motes agent is send to the network in order to capture and create facilitator and slave nodes before any user requests or the actual application is forwarded. The life cycle of a facilitator In-Motes agent is shown in **Figure 2**.

The facilitator agent works by continuously checking whether any of the nodes are available for capture. The user sends a single facilitator into the wireless sensor network, although this is not limited by the In-Motes infrastructure, allowing more than one facilitator to be deployed in large scale applications where the nodes exceed the total number of 20.

Upon arrival at the first available node, the facilitator will insert a facilitator tuple into the tuple space assigning thus the first facilitator node. The capturing procedure takes place when a facilitator agent during its migration registers a capture or a slave reaction to the corresponding node. An alternative is for the facilitator agent to clone rather than migrate and generate a slave agent inside the wireless network.

A counter will be incremented every time a capture reaction takes place; when the counter reaches two, the facilitator agent will migrate again to the next available node assigning this time around a new facilitator tuple and slave reaction and the capturing procedure will repeat.

It is expected that during the lifetime of a wireless sensor network some nodes will eventually die and information will be lost. In-Motes can adapt and dynamically take actions upon unexpected scenarios like the ones mentioned above. If a facilitator node goes down the network will dynamically adapt since the lifecycle of the facilitator agent that we described above never terminates and a new capturing procedure will take place.

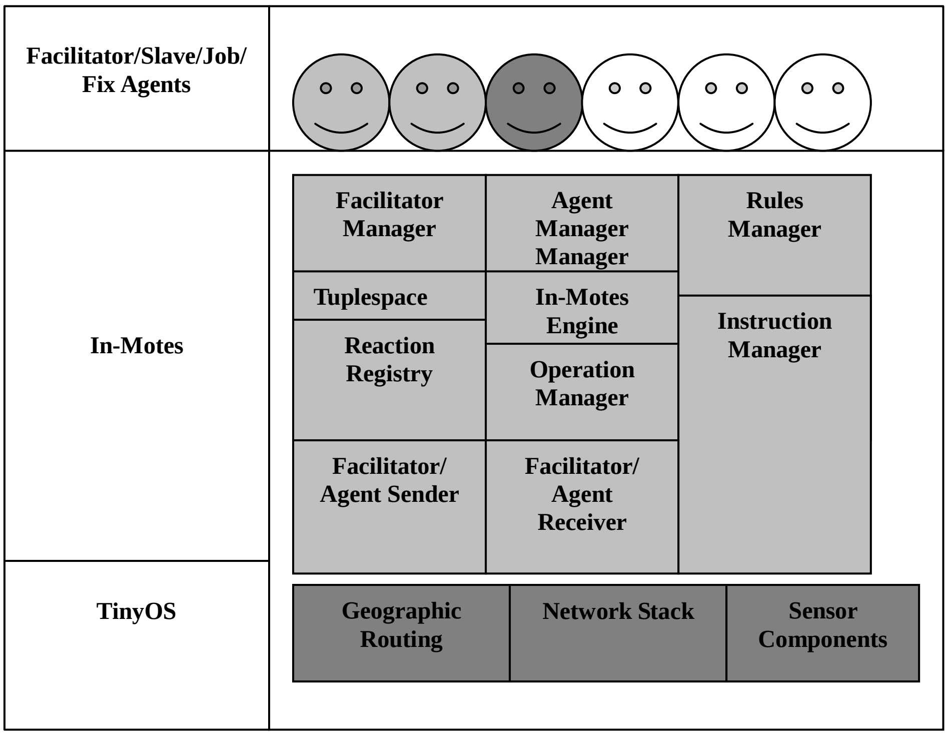

Figure 1. The In-Motes Architecture, the first layer consists of the In-Motes agents, the In-Motes layer sits on top of the TinyOS platform.

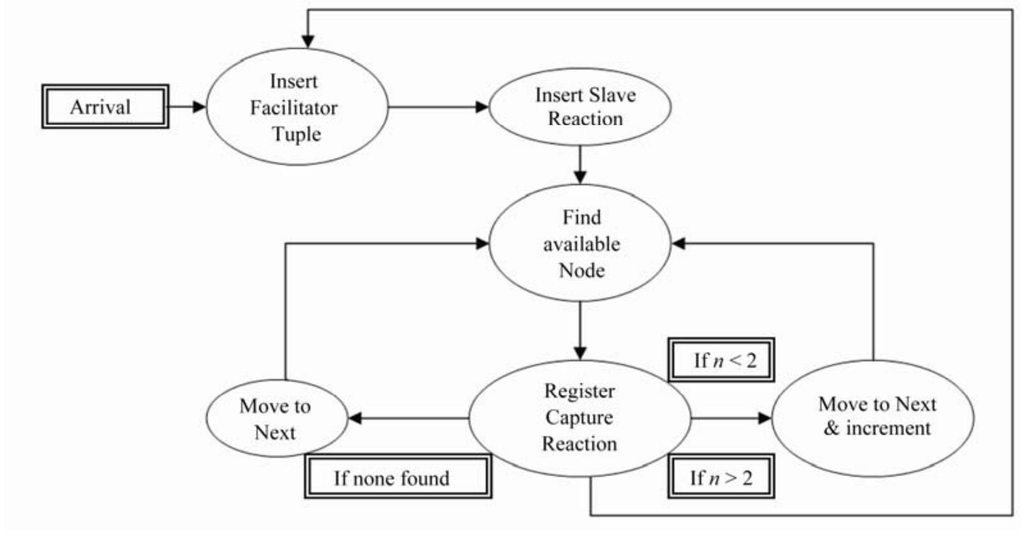

Figure 2. The life cycle of the In-Motes facilitator agent.

2. The In-Motes EYE Application

Last year more than 2 million motorists were caught speeding on camera, raising £120 m a year in revenue for so-called 'Safety Camera Partnerships' comprising police, magistrate councils and road safety groups. Speed cameras have boomed on British roads from a handful a decade ago to 3,300 fixed sites and 3,400 mobile devices today. In October 2006 a massive flaw in a new generation of speed cameras was reported by Daily Mail [11] allowing motorists to avoid speeding fines in some of the busiest UK motorways by simply changing lanes. The Home Office admitted in public that drivers could avoid being caught the by hi-tech 'SPECS' cameras, **Figure 3**,

Figure 3. The SPEC road cameras with the problematic software, taken from http://www.speedcamerasuk.com/SPECS.htm

which calculate a car's average speed over a long distance.

The cameras were designed to catch motorists who simply slow down in front of a camera, case of the Gatso speed cameras, and then drive above the speed limit until they reach the next one. The loophole in the software is located when a motorist changes lanes as it is unable to calculate if the average speed is above the limits due to the fact that the fixed points of measurement need to be in a straight line. Although the software was designed to improve the road safety, by measuring a driver's average speed between two fixed points which can be many miles apart the loophole meant that drivers may actually increase the risk of accidents by continually switching lanes. Since then, an update of the software took place to correct this problem but as Mr. Collins, a Home Office representative stated recently "There are configurations when (a speeding vehicle) would not be picked up, if it's gone from lane one to lane three between cameras."

As we mentioned above Gatso speed cameras, **Figure 4**, are frequently used but their vast flow is there size that makes them visible to a driver and the fact that GPS units inside a car can detect them and warn a speeding careless driver in advance.

Figure 4. The Gatso road cameras that are easily detectable, taken from http://www.speedcamerasuk.com/gatso.htm

The automobile industry is spending every year a worthy budget in embedded sensor technology and in the recent years sensors such as car parking sensors and car crash sensors have been developed and installed in production line vehicles increasing the alert and safety of a driver [12]. Thus, we believe that in the near future a sophisticated wireless sensor system cooperating with speed or acceleration sensors embedded in the car could resolve the speed cameras problems and why not even eliminate them.

With the In-Motes EYE application and our middleware we demonstrated that the above proposal could be feasible if it was funded in a large scale and automobile companies expressed interest. A more sophisticated network is required and more advanced sensors should be developed towards that goal without though major modifications to our middleware specification that we envisage that could adapt relatively easy in a large scale scenario.

In terms of hardware we have used a set of 5 mica2 sensors with the accompanied MTS310CA sensor boards. One base station, a laptop connected with an MIB510 interface board fixed in an area served as the aggregation point. Two radio controlled cars with an attached mica2 sensor had the role of the moving objects in the environment, **Figure 5**. The last 2 mica2 sensors were occupied by the two facilitators and they were placed in a straight line 2 meters apart communicating with the aggregation point that was 6 meters away and in the line of sight the facilitator nodes. All the hardware was provided by the Crossbow Technology Ltd. All the sensor motes were working under the TinyOS operating system [13] and the application was deployed through our In-Motes Reloaded middleware. Our trial took place in an outdoor environment with a sufficient space for the radio controlled cars to accelerate without any obstacles. The laboratory controlled environment was avoided all together mainly because of the limited space and our assumption of noise interference from the laboratory equipments that seemed to interfere with the wireless

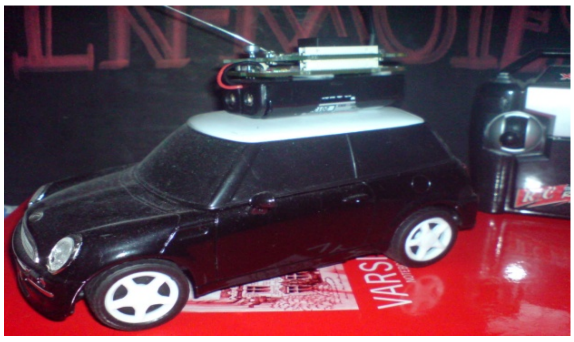

Figure 5. One of the radio controlled cars with the attached mica2 sensor that was used for the In-Motes EYE trial.

sensor network transmissions.

The In-Motes EYE application uses dynamic job In-Motes agents for forwarding all the user queries in the wireless sensor network. Every monitor request that is send from the user contains a critical parameter which is used locally to determine if a sequence of readings should be reported or not. The nodes that are attached to the cars are waking up every two minutes and report to their facilitator node one acceleration reading. When the critical parameter is breached, each facilitator sends one packet containing one acceleration reading per 10 seconds for a period of 2 minutes.

The user also has the freedom to choose a random radio control car and at random intervals to check its acceleration behavior for a period of two minutes. In order to do so, the memory of the node of the selected car firstly must be flushed, by sending a fix In-Motes agent in order the resident job agent with the critical parameter and the according reactions to be erased. Then a new job agent is migrating to the desired location. After the end of the measuring period the job agent stops its execution and dies. **Figure 6** demonstrates part of the In-Motes EYE application code that was deployed to the wireless sensor network during our trials.

We initialize our wireless sensor network by injecting once two facilitator agents to the according nodes that are placed in a straight line and 2 meters apart, with one node attached to each car acting as slave to them. The facilitators captured the nodes and were ready to receive the first job requests in 35s. We send, once, two dynamic job agents, 40 bytes each to the wireless sensor network. The above procedures, as well as the wake up calls and the transmission of data readings are executed by the application without us interfering with the process unless a failure is noticed or as we mentioned above a user desires to monitor the acceleration of a specific car.

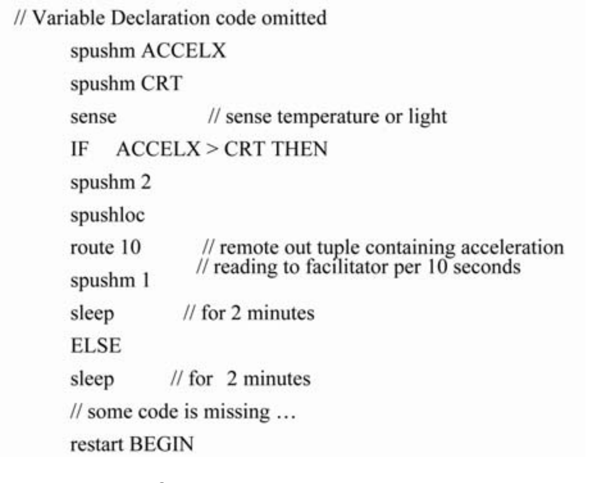
```
// Variable Declaration code omitted
        spushm ACCELX
        spushm CRT
        sense              // sense temperature or light
        IF    ACCELX > CRT THEN
        spushm 2
        spushloc
        route 10           // remote out tuple containing acceleration
                           // reading to facilitator per 10 seconds
        spushm 1
        sleep          // for 2 minutes
        ELSE
        sleep          // for  2 minutes
        // some code is missing …
        restart BEGIN
```

Figure 6. Part of the In-Motes EYE application that was deployed from In-Motes Reloaded middleware.

3. The In-Motes EYE Field Tests

The main location of the In-Motes EYE trial was based on the outdoor environment of a garden. As we mentioned above the two facilitator nodes where placed in a straight line 2 meters apart and they were communicating with the aggregation point that was 6 meters away and in the line of sight the facilitator nodes, no physical obstacles where intervening during the transmissions. We used two radio controlled cars which at random time intervals were accelerating in a square area of the garden, 10 × 30 meters. The motion of the cars was random and it was not following any patterns, **Figure 7**.

We used the accelerometer sensor which exists on the MTS310CA interface boards, a MEMS surface micro-machined 2-axis, ±2 g device that can be used for tilt detection, movement, vibration, and/or seismic measurement [14]. According with the manufacturer it is advised that for accurate measurements, after every trial the accelerometer needs to be recalibrated for every sensor in both axes. During our trials no recalibration of the devices took places as it was beyond the scope of the experimental procedure.

Since the voltage response for the accelerometer is linear with respect to the measured acceleration the motes ADC value can be translated into meaningful engineering acceleration units following the below linear equation:

Reading (m/s^2) = 1.0 (Cal_pos_1g-ADC)/Scale factor

where:

Scale factor = Cal_pos_1g - Cal_neg_1g / 2
Cal_pos_1g = 500
Cal_neg_1g = 400
ADC = output value from Mote’s ADC measurement

The critical parameter for the In-Motes EYE application was set to be 1.082 m/s^2, a value that was selected as it was the average acceleration reading reported from both of the radio controlled cars when they moved randomly in the environment.

Figure 8 is a representative graph that was produced when one of the radio controlled cars was accelerating above the critical parameter. The facilitator node that was assigned to that vehicle was reporting readings back to the end user for a period of 2 minutes by sending 1 packet containing one acceleration value per 10 seconds. **Figure 8** does not represent continuous values of acceleration rather than values of acceleration when the critical parameter was breached, thus a single reading is reported that it only goes back to the previous transmission which happens every 10 seconds. Many of the graphs were produced by the In-Motes Reloaded Oscilloscope application allowing us to observe the acceleration of the cars in real time.

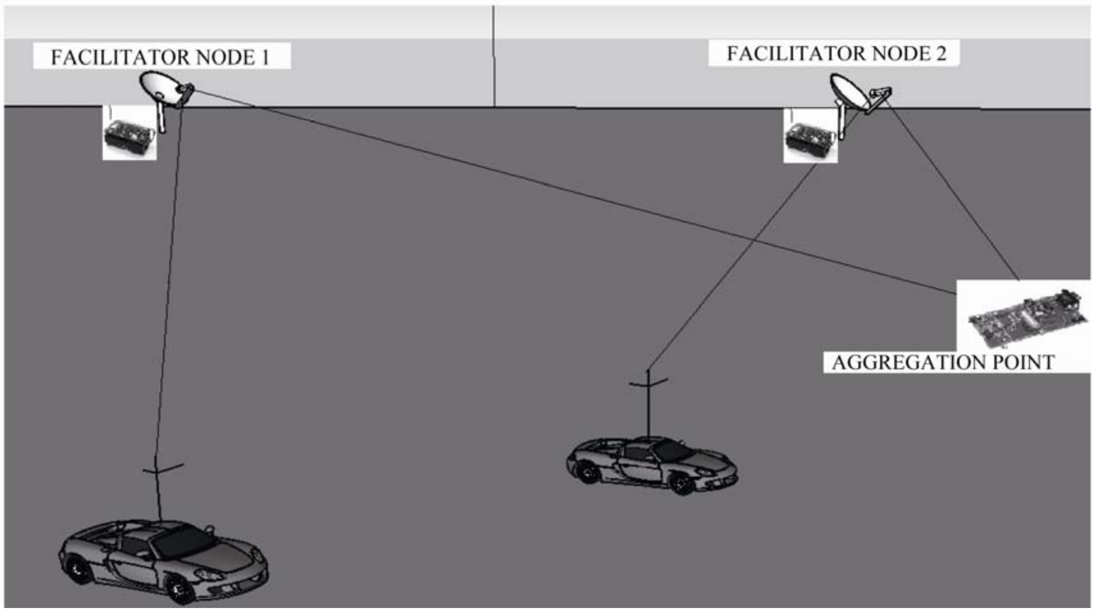

Figure 7. A representation of the In-Motes EYE trial environment and its actors.

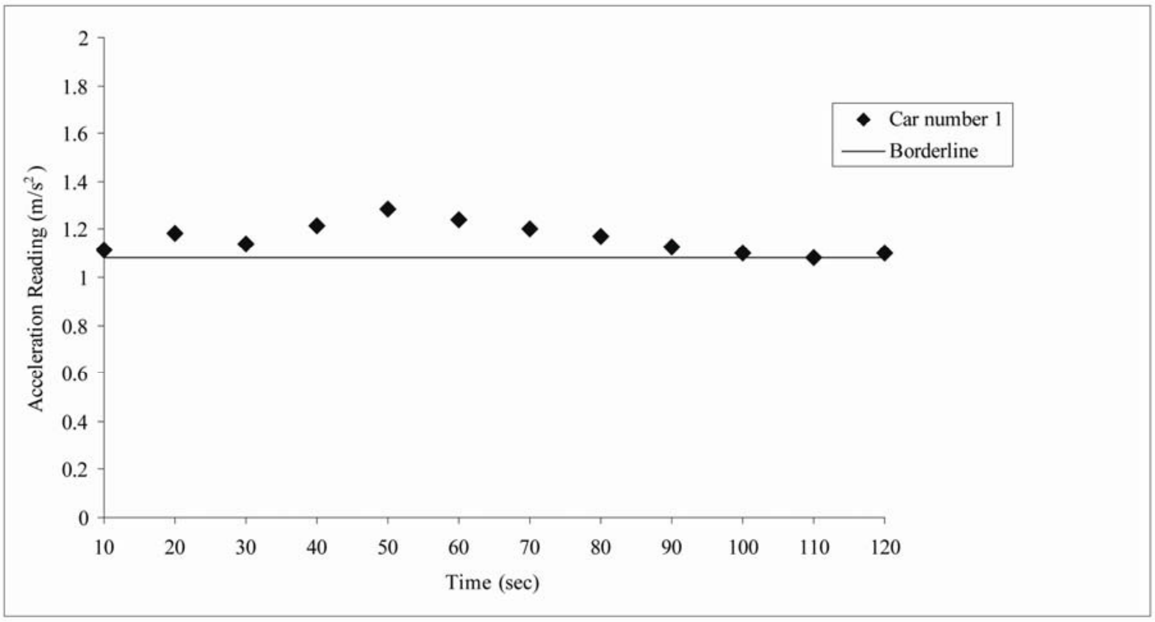

Figure 8. Car 1 acceleration readings reported to the end user when the critical parameter was breached.

As we mentioned earlier the application was allowing a user to monitor the acceleration pattern of a moving car even if the critical parameter was not breached by simply injecting a new job agent to the vehicles sensor. **Figure 9** presents the graph that was produced in the scenario where the user was monitoring the acceleration pattern of car 1 for a period of two minutes while car 2 was accelerating at the same time breaching the critical parameter of the application. The values for car 1 were the current single values of acceleration that were recorded every 10 seconds for the given period while the values for car 2 are as before values of acceleration when the critical parameter was breached. The sensors for each car in this case were working under two different dynamic job agents.

4. The In-Motes EYE Analysis

Overall, we spend one month running different scenarios with the radio controlled cars and changing experimental parameters such as the epoch time per trial circle helping us to understand better the nature of the wireless sensor network we deployed and evaluate better the engine of the In-Motes Reloaded middleware. Packet delivery did not affect by the distance between the aggregation point and the sensors rather from facts such as the duration per measuring period and buffer overflows at the facilitator node end.

Figure 10 presents the total number of packets that were delivered from a facilitator node monitoring one car that breached the acceleration critical parameter for different epochs.

From the above graph it is obvious that the packet delivery performance was affected as the measuring period was increased. Packet losses were observed mainly due to two reasons. Firstly, increasing the activity of a slave node of a radio controlled car it meant that we were increasing the battery power consumption of the mote.

For mica2 motes this increase usually affects the performance of the hardware resulting to delays in obtaining

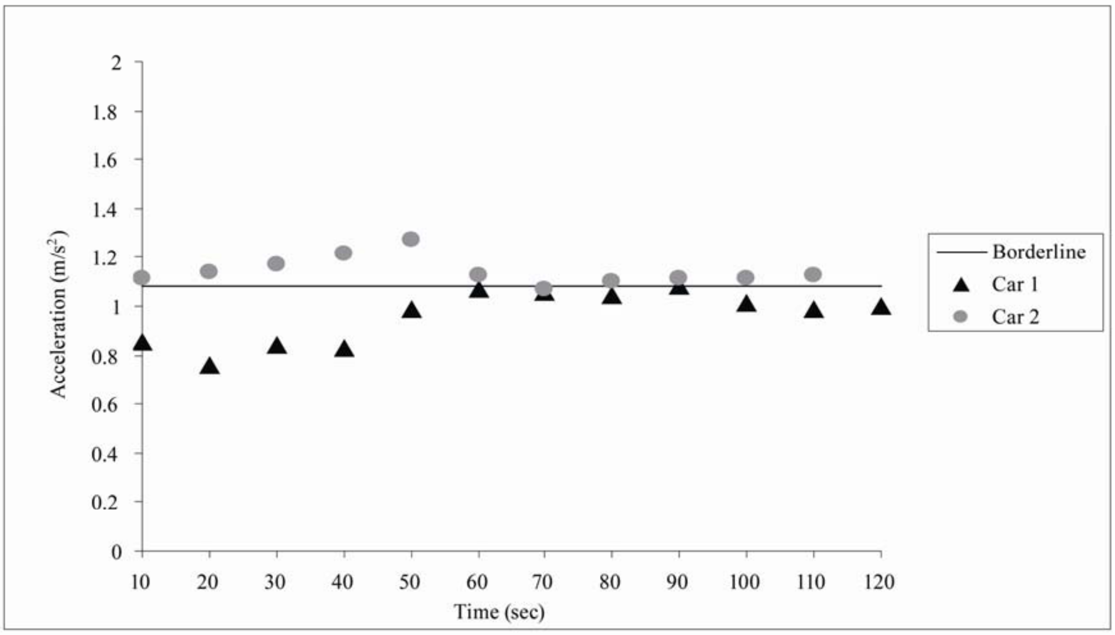

Figure 9. Car 1 is under surveillance by the user while Car 2 accelerates breaching the critical parameter.

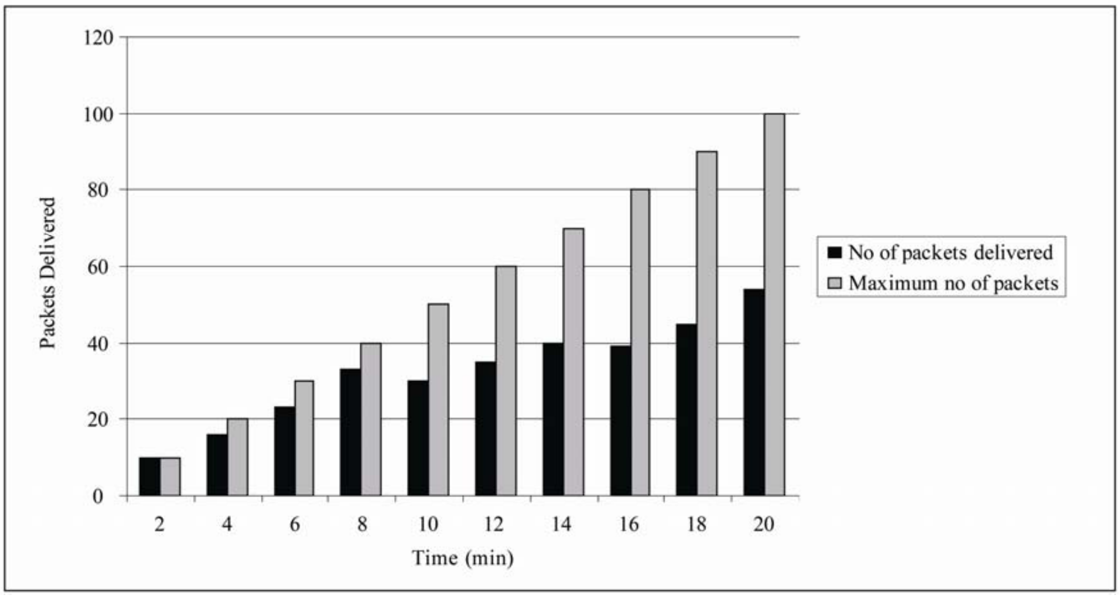

Figure 10. Packet delivery performance of the middleware running the In- Motes EYE for different measuring periods.

a sensing value and even stalls the operation of a node as the battery levels are exhausting. Secondly, the facilitator node when the traffic is heavy it will drop some packets as its sending buffer will overflow. Although, those problems affected the performance of the middleware we observed a success rate that was above 50% in all the trials. That can be explained due to the modifications that we applied to the core engine of the middleware eliminating in most cases race conditions, re-transmissions and overuse of memory resources both virtual and physical ones.

Overall, the following type of errors for the duration of the trial were observed and recorded:

- Stall of the whole network: 5

Describes the condition where the whole network was inactive and no readings were received at the user end. Reasons behind this behavior could be identified due to the below error types. The remote action that was taken was to send a fix In-Motes agent to flash the memory of all the network nodes and reinstall the application.

- AgentSender Fail sending agent: 12

Describes the condition where one or more nodes were not responding to user requests. The mobile code transmitted from the facilitator node never reached its destination. The middleware was providing this information and the actions that were taken were: Either send a fix In-Motes agent to flash the node's memory and then send the user request again with a new In-Motes job agent or physically visit the node turn it off and on again. The problem was visualized from the user as the problematic node was blinking its red LED.

- Null Readings: 100

Describes the condition where the facilitator node was sending back to the end user a null reading. No actions were taken place.

- Facilitator Buffer_Overflow: 5

Describes the condition where the facilitator node could not handle all the receiving traffic resulting in stalling its operation. The red LED was blinking and a pop up window was informing the user about the error. A fix In-Motes agent was send from the user in order the internal memory of the node to be flashed.

- Node Buffer_Overflow: 6

Describes the condition where one or more nodes of the network could not handle any queries and wasn't reporting back to the facilitator node although it had accepted and stored a new instruction (AgentSender Success sending agent). The red LED was blinking and a pop up window was informing the user about the error. A fix In-Motes agent was send from the user in order the internal memory of the node to be flashed.

- AgentReceiver Fail receiving agent: 15

Describes the condition where one or more nodes of the network could not handle any queries and wasn't reporting back to the facilitator node. Although the destined node had accepted (AgentSender Success sending agent) the new In-Motes job agent the new instruction was never matched with any template in the tuplespace so the new tuple that was added did not trigger any reaction from the node. The red LED was blinking and the In-Motes engine was informing the user about the error. A fix In-Motes agent was send from the user in order the internal memory of the node to be flashed.

- Misplacement of sensor/Drop/Other: 20

Describes the condition where a sensor accidentally was misplaced, dropped, or switched off during its operation.

Failures that solved instantly by our middleware the moment they were noticed, by simply flashing the sensors and reinstalling remotely the application. Stalls of the 2 sensors of the radio controlled cars we were using were not observed frequently mainly cause of the middleware ability to handle and allocate better the memory resources of the system than before. The facilitator scheme and In-Motes Reloaded communication protocol worked as expected and we did not observe any store and forward delays.

5. Conclusions

In this paper we have demonstrated how In-Motes can be used as a flexible platform to deploy dynamic applications in wireless sensor networks. Also with the use of the facilitator and job agents we showed that multiple applications can simultaneously share a network. We demonstrated the In-Motes EYE application in a real dynamic environment and we proved that mobile agents and tuple space/facilitator based communication can be used to program a WSN and increase its flexibility. Network optimization strategies included agent design, error correction and an energy saving scheme for the motes. This study has allowed us to improve the In-Motes architecture in order to provide a better platform for developing applications in WSN's. Our successful implementations of the In-Motes EYE application together with the steady performance of the new version of the middleware compared with the previous version, lead us to envisage a near future where the wireless sensor technology could establish a framework that will overcome various limitations that those networks inhabit.

6. Acknowledgements

The research is funded by EPSRC and Aston University. D. Georgoulas would like to acknowledge the support and valuable advice of Dr. David Holding, Aston University, Mr. C. L Fok of Washington University in St Louis, for helping us understand the Agilla programming core and Mr. M. Smith, technical support engineer of Crossbow technology, for technical advice.

7. References

[1] D. Georgoulas and K. Blow, "In-Motes: An Intelligent Agent Based Middleware for Wireless Sensor Networks," *Proceedings of the* 5th *WSEAS International Conference on Application of Electrical Engineering*, Prague, 12-14 March 2006, pp. 225-231.

[2] C.-L Fok, G.-C. Roman and C. Y. Lu, "Rapid Development and Flexible Deployment of Adaptive Network Applications," *Proceedings of the* 25th *IEEE International Conference on Distributed Computing Systems*, Columbus, 10 June 2005, pp. 653-662. doi:10.1109/ICDCS.2005.63

[3] P. Levis and D. Culler, "Maté: A Tiny Virtual Machine for Sensor Networks," *Proceedings of the* 10th *International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, 6-10 October 2002, pp. 85-95. doi:10.1145/605397.605407

[4] C. M. RoadKnight and I. W. Marshall, "Future Network Management—A Bacterium Inspired Solution," *Proceedings of the* 2nd *International Symposium Engineering and Intelligent Systems*, Tel Aviv, March 2000, pp. 123-129

[5] P. Levis, "TinyOS programming," Stanford University, USA, 2006 http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf [13/03/2007]

[6] J. M. Bradshaw, "An introduction to Software Agents," In: B. M. Jeffrey Ed., *Software Agents*, MIT Press, Cambridge, 1997,

[7] D. Georgoulas and K. Blow, "In-Motes Bins: A Real Time Application for Environmental Monitoring in Wireless Sensor Networks," *Proceedings of the* 9th *IEEE/IFIP International Conference on Mobile Wireless Communications Networks*, Cork, 19-21 September 2007, pp. 21-25. doi:10.1109/ICMWCN.2007.4668173

[8] D. Georgoulas and K. Blow, "Intelligent Mobile Agent Middleware for Wireless Sensor Networks: A Real Time Application Case Study," 2008 4*th Advanced International Conference on Telecommunications*, Athens, 8-13 June 2008, pp. 95-100. doi:10.1109/AICT.2008.51

[9] G. Cabri, L. Leonardi and F. Zambonelli, "Reactive Tuple Spaces for Mobile Agent Coordination," *Proceedings of the* 2nd *International Workshop on Mobile Agents*, Stuttgart, September 1998, pp. 237-248.

[10] D. Georgoulas and K. Blow, "Making Motes Intelligent: An Agent Based Approach to Wireless Sensor Networks," *In WSEAS on Communications Journal*, Vol. 5, No. 3, 2006, pp. 515-522.

[11] Ray Ramsey, "Drivers Can Avoid Speeding Tickets...by Changing Lanes," 2006. http://www.dailymail.co.uk/news/article-410539/Drivers-avoid-speeding-tickets--changing-lanes.html

[12] BMW United Kingdom Research, "Passive Safety in Vehicles," 2010. http://www.bmw.co.uk/bmwuk/bmwcorporate/sales/responsible/passive_safety/

[13] Crossbow, "Wireless Systems for Environmental Monitoring," 2007. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/Smart_Dust_AppNote.pdf

[14] Crossbow, "Crossbow Users Manual," *Crossbow Technology*, 2007, pp. 1-45.