This work-in-progress paper was presented as part of the main technical program at IFIP WMNC'2013

# A Multi-Layer Architecture for Wireless Sensor Network Virtualization

Imran Khan[+], Fatna Belqasmi[*]
[+]Institut Mines-Télécom
Télécom SudParis
Evry, France
imran@ieee.org, noel.crespi@it-sudparis.eu

Roch Glitho[*], Noel Crespi[+]
[*]Dept. CIISE
Concordia University
Montreal, Canada
fbelqasmi@alumni.concordia.ca, glitho@ece.concordia.ca

***Abstract*—Wireless sensor networks (WSNs) have become pervasive and are used for a plethora of applications and services. They are usually deployed with specific applications and services; thereby precluding their re-use when other applications and services are contemplated. This can inevitably lead to the proliferation of redundant WSN deployments. Virtualization is a technology that can aid in tackling this issue. It enables the sharing of resources/infrastructures by multiple independent entities. This position paper proposes a novel multi-layer architecture for WSN virtualization and identifies the research challenges. Related work is also discussed. We illustrate the potential of the architecture by applying it to a scenario in which WSNs are shared for fire monitoring.**

***Keywords-Wireless Sensor Networks; Virtualization; Overlay Networks; Wireless Sensor Network Virtualization***

## I. INTRODUCTION

Wireless Sensor Networks (WSNs) are amalgamations of micro-electro-mechanical systems, wireless communication systems and digital electronics nodes that sense, compute and communicate [1]. They are made up of sensors, sinks and gateway nodes. Virtualization is a technology that presents physical resources logically, and enables their efficient usage and sharing by multiple independent users [2]. The new generations of WSN nodes have more and more resources (e.g. storage, processing) [3]. It now makes sense to consider the efficient usage and sharing of these resources through virtualization. WSN virtualization enables the sharing of a WSN infrastructure by multiple applications [4]. There are two possible approaches to WSN virtualization. The first one is to allow a subset of sensor nodes to execute an application, while at the same time (preferably) another subset of sensor nodes executes a different application [5]. These subsets can vary in size and in number according to the application requirements. The second approach is to exploit the capabilities of the individual sensor nodes and execute multiple application tasks [4], [6] and [7]. Each application task is run by a logically distinct but identical physical sensor node.

This position paper proposes a new multilayer architecture for WSN virtualization and discusses the related research challenges. A real-life fire monitoring application scenario is used for illustration throughout the paper. The rest of the paper is organized as follows. A fire monitoring motivating scenario, the requirements and related work are presented in Section II. In Section III the proposed architecture is presented and its applicability illustrated by the fire monitoring scenario. Research issues are discussed in Section IV and Section V concludes the paper.

## II. FIRE MONITORING MOTIVATING SCENARIO, REQUIREMENTS AND RELATED WORK

### A. Fire Monitoring Motivating Scenario

Consider a city near an area where brush fire eruptions are common and let us assume that the city administration wants to monitor fires using a WSN and a fire contour algorithm [8]. Some private homes in the area already have sensor nodes to detect fire. For this application, the city administration could either deploy sensor nodes all over the city (even in private homes), or only in areas under its jurisdiction (i.e. streets, parks) and re-use the sensor nodes already deployed in private homes. The former is not an efficient approach whereas the latter approach is efficient and will avoid redundant WSN deployments. In the latter approach, at least two applications will share sensor nodes: one, belonging to home owners and the other belonging to the city administration. Without technologies such as virtualization this solution would be 'mission impossible'.

### B. Requirements

The first requirement that can be derived from the scenario is the concurrent execution of tasks from multiple applications by the sensor nodes. We call this WSN node-level virtualization. The second requirement is the ability of WSN nodes to dynamically form a group to perform isolated and transparent execution of application tasks in such a way that each group belongs to a different application. We term this mechanism as network-level WSN virtualization. The third requirement is support for the prioritization of the application tasks. For certain events, this might be crucial. The final requirement is that the proposed solution should be generic and platform-independent.

### C. Related Work

Table I provides a summarized view of the related work in relation to the requirements identified in the previous section. It

978-1-4673-5616-9/13/$31.00 

shows that none of the existing proposals meets all of our requirements.

The authors in [4] discuss SenShare platform, which supports both WSN-node and network-level virtualization. A runtime layer on top each sensor node supports multiple applications. SenShare works on top of embedded Linux OS and only supports TinyOS applications. A network-level overlay is created to group WSN nodes executing similar applications. In [5], WSN nodes form subsets to support applications that monitor dynamic phenomena. Each independent subset executes an application, supporting network-level virtualization. Two illustrative applications are also discussed. Maté [6] presents a pioneering work that supports node level virtualization by means of a tiny virtual machine and a stack-based interpreter. It was designed to work on early generation, resource-constrained sensor nodes and is quite restrictive.

Melete [7] is an extension of Maté and supports both node- and network-level virtualization. At the node level, Melete provides interleaved execution of multiple applications on a sensor node. At a network level, Melete supports the logical grouping of WSN nodes where each group is dedicated to a single application. The sensor nodes can be part of more than one logical group at the same time. VITRO [9] aims to transform application-specific WSNs into large-scale virtual networks supporting multiple applications. VITRO offers node-level virtualization using a hypervisor that controls virtualization-related tasks. Authors in [10] present a self-organizing tree-based approach, as a possible solution to [5], to facilitate the creation, operation and maintenance of dynamic groups that facilitate WSN network level virtualization. The solution ensures that no event remains undetected. MANTIS [11] is an embedded operating system that supports the simultaneous execution of threads on sensor nodes by using context switching. It supports preemptive multithreading by assigning priorities to threads.

TABLE I. SUMMARY OF RELATED WORK

| **Related Work** | **Requirements** | | | |
|---|---|---|---|---|
| | ***Node-Level virtualization*** | ***Network-Level virtualization*** | ***Application Priority*** | ***Platform Independence*** |
| SenShare | Yes | Yes | Yes | No |
| Maté | Yes | No | No | Hardware |
| Melete | Yes | Yes | No | No |
| VITRO | Yes | No | No | No |
| [5] | No | Yes | No | Yes |
| [10] | No | Yes | No | Yes |
| MANTIS | Yes | No | Yes | Software |

## III. PROPOSED ARCHITECTURE

In this section we discuss the architectural principles; the layers, paths and nodes, the interfaces and the protocols. We also illustrate them with a fire monitoring scenario. We assume that all physical sensor nodes can execute concurrent tasks assigned by applications and services. This assumption is not far-fetched because existing sensor kits such as SunSpot [12], operating systems like Contiki [13] and Squawk JVM [14] do support concurrent task execution.

### *A. Architecture Princples*

The first architectural principle is that any new application/service (e.g. city administration application) is deployed as a new overlay on top of the physical WSN. Overlays have several advantages: they are distributed, lack central control and allow resource sharing [15]. These features make them an ideal candidate for WSN virtualization. The second principle is that any given physical sensor node can execute (locally) a task for a given application deployed in the overlay. Any given sensor node may execute several such tasks at any given time. These tasks include gathering sensor data and sending event notifications to the overlay applications.

The third principle is that the overlay-related operations are not necessarily performed by the sensor nodes directly concerned, as they may not have enough capabilities to support the overlay middleware. When that is the case, they will delegate the operations to more powerful sensors and even to other nodes. The fourth and final principle is that within the architecture there are separate paths: data and signaling. The sensor data (e.g. temperature values) is transmitted from sensor nodes to the overlay application using the data path. The control data (e.g. overlay initiation and overlay join request/reply messages) is sent over the signaling path.

### *B. Layers, paths and functional entities*

Figure 1 shows the layers, paths and nodes. There are three layers (physical, virtual sensor and overlay) and two paths (data and signaling). At the physical layer a WSN has two types of sensor nodes. Type A sensor nodes perform overlay management operations for themselves and on behalf of other sensor nodes, whereas type B sensor nodes cannot. In figure 1 sensor Z is a type A node and sensors X and Y are type B nodes. There is another network at the same layer, called the Gates-to-Overlay (GTO) network, consisting of heterogeneous nodes such as powerful sensors, gateways and sink nodes. GTO nodes can communicate with the WSN sensor nodes and help them to join the application overlays. In this architecture, type B sensors have two options for joining the application overlays, either via type A sensor nodes or via GTO nodes. In figure 1, sensor Z can perform overlay management operations for itself and for sensor Y, whereas sensor X uses a GTO node to join the overlay.

The virtual sensor layer consists of the virtual sensors that execute either overlay application tasks or overlay management tasks. The virtual sensors of sensor X and sensor Y only execute overlay application tasks, as they are type B nodes. Sensor Z, a type A node, has three virtual sensors, two for the overlay application tasks and one (VSZ2) for the overlay management task. Both sensor Y and sensor Z use VSZ2 to participate in application overlays. The overlay layer consists of multiple application-specific overlays (for simplicity only two overlays are shown). Each application overlay is created

by the end user application and consists of two types of nodes, virtual sensors that run overlay application tasks and virtual sensor/GTO nodes that run overlay management tasks.

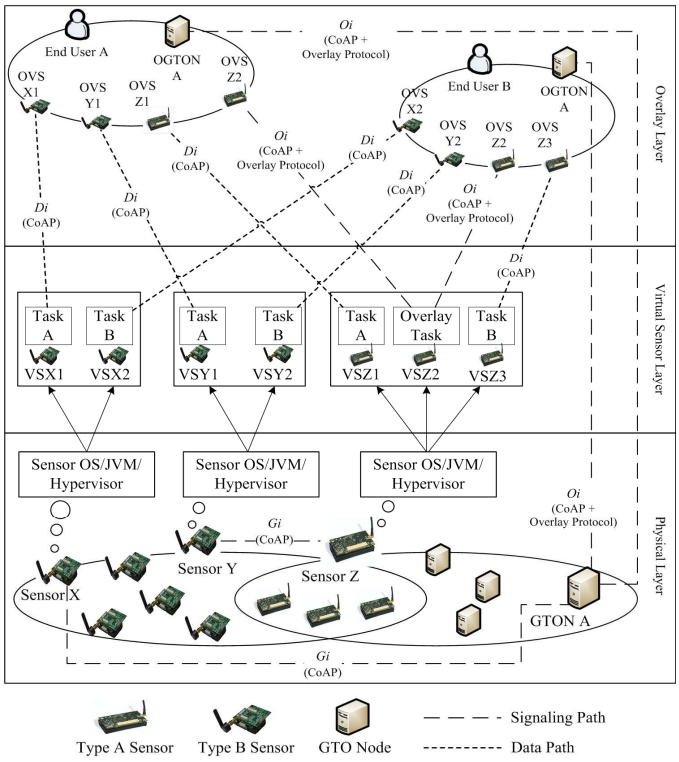


Figure 1. General architecture

In these overlays the boundaries enforced by the physical WSNs disappear, easily allowing data exchange between them. As per the fourth architectural principle, there are separate paths in the architecture between various entities. The interfaces and protocols used at these paths are discussed in the next section.

### C. *Interfaces and Protocols*

In figure 1, the data path uses the data interface (*Di*) provided by all the sensor nodes. This interface supports a lightweight protocol, suitable for resource constrained devices such as type B nodes. CoAP [16] is a candidate protocol for this interface. The interface to the overlay (*Oi*) is used by the signaling path and supports CoAP along with any suitable overlay protocol, e.g. TChord [17], ScatterPastry [18] or JXTA [19]. Both type A and GTO nodes provide this interface. The Gate-to-overlay interface (*Gi*) is provided by all sensors as well as GTO nodes. As type B nodes are not capable of supporting any overlay protocol, they cannot receive specific overlay messages. Type A and GTO nodes can receive such messages and communicate over the *Gi* interface to prepare type B nodes to join an overlay. Using CoAP for the *Gi* interface eliminates the need for type B nodes to support another protocol.

### D. *Illustrative Use Case*

Figure 2 illustrates the application of our architecture to the fire monitoring scenario. The city administration and the home owners deploy the fire detecting sensors in public streets and private homes, respectively. It is possible that some sensors in private homes are type A nodes and some are type B nodes. In figure 2, home 1 and home 3 have type B nodes and home 2 has a type A node. Sensors X and Z use a home gateway and city sensor A in the public street, respectively, to participate in the city admin overlay. Sensor Y participates in the city admin overlay on its own. It is assumed that owners register their sensors with the city admin during their deployment.

The creation of the city admin overlay is a three step process. The first step is overlay pre-configuration, which is performed during offline registration. Data such as sensor types, their capabilities, IDs and addresses for communication are collected in this step. During this step it is determined whether any sensor requires another node for joining the city admin overlay. If so, then that node's relevant information is also collected along with any associated mapping/binding. All this information is stored in a central repository (not shown in fig. 2), which is easily accessible to the city administration.

The second step is the activation of the overlay. The city admin application connects to the repository and retrieves a list of sensors, along with all the details, to include them in its overlay. An overlay invitation message is sent to the type A and/or GTO nodes (Home gateway, VSY2 and city sensor A in fig. 2) over the *Oi* interface. These nodes reply by sending overlay join requests to perform overlay management operations. The city admin then sends invitation message to the virtual sensors that will be executing the city admin task (VSX2, VSY3 and VSZ2 in fig. 2). It is assumed that the virtual sensors already have the task code.

VSY2 poses no joining issue as its physical sensor is a type A node, so it easily joins the city admin overlay as a logical node (OVSY2) and sets up its data path with it. For VSX2 and VSZ2, the overlay invitation message is received by home gateway and city sensor A, respectively, on their *Oi* interfaces. These nodes then send the overlay join message on behalf of VSX2 and VSZ2. The city admin creates logical nodes in the overlay (OVSX2 and OVSZ2) and sends the relevant IDs to VSX2 and VSZ2 so they can to send their data (e.g. event notifications) to the OVSX2 and OVSZ2. VSX2 and VSZ2 receive this data on their *Gi* interfaces from home gateway and city sensor A respectively, and set up their respective data paths with OVSX2 and OVSZ2 using the *Di* interface.

The third and final step is the execution of the end user application, which is fire monitoring in this use case. Whenever fire is detected by a physical sensor (e.g. sensor X), its virtual sensor (VSX2) sends the gathered data to the OVSX2 in the city admin overlay using the *Di* interface. Inside the city admin overlay OVSX2 initiates the fire contour computation based on the algorithm used by the city admin. It is now able to share the received fire event data with its neighboring overlay nodes. In the absence of this type of overlay, the exchange of fire event data is not possible as each sensor node is in its own private domain.

## IV. RESEARCH CHALLENGES

The first challenge is providing a discovery and publication framework. Such a framework will be used by the different

actors, including the resource constrained devices, to publish and discover on the fly. The approach used in the previously discussed use case (i.e. offline and static registration) has too many limitations. A dynamic publication and discovery mechanism that factors in the limitations of the resource constrained devices is required.

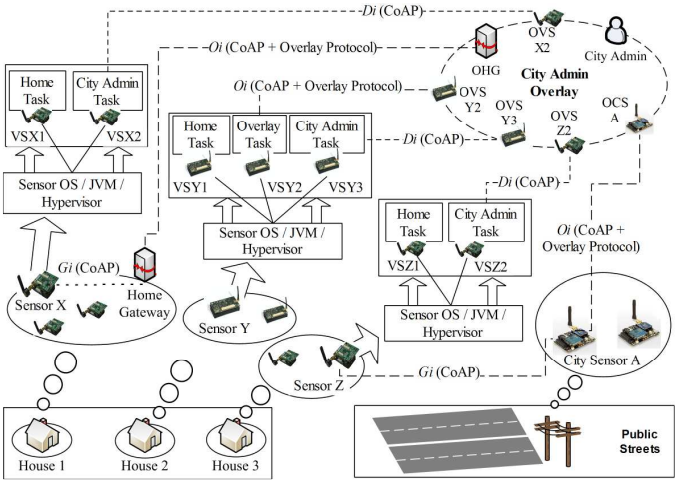


Figure 2. Fire monitoring problem

The second challenge is the signaling framework. There are several signaling frameworks, but they usually target resource reservation (RSVP) and session management (e.g. SIP) and may not be suitable for our needs. In addition, the framework should be adequate for resource-constrained environments. A potential direction is the design of a signaling framework that uses CoAP as its underlying protocol.

Yet another challenge is the protocols for data paths. CoAP is an emerging protocol targeting resource constrained devices and is an attractive option. However, CoAP presents many issues that have not yet been solved. In addition, the use of CoAP in an overlay environment remains to be investigated.

The fourth challenge is finding an efficient mechanism to disseminate the application task to the sensors. Some solutions are provided in [6], [7] and [13], but none is suited for the requirements of WSN virtualization. A proposed solution must provide the flexibility of updating the application task and the modification of parameters at runtime for adaptive sampling.

A fifth challenge is the protocols to be used in the overlays, especially as these protocols should be middleware-independent whenever possible.

The final challenge is developing a viable business model for WSN virtualization. While the use case discussed in this paper does not provide the classical separation between WSN infrastructure providers and WSN service providers, in a realistic business model there may be other players as well, e.g. GTO node providers, when these nodes do not belong to WSN.

## V. CONCLUSION

This position paper has proposed a three-layer architecture for WSN virtualization and has discussed the related challenges. The next step of our research will be a proof of concept prototype that demonstrates its feasibility. After that we will tackle the research issues we have identified: the publication/discovery framework, the signaling framework, the protocols for the data path, the framework for disseminating the applications tasks to the sensors and finally the middleware-independent protocols for the overlays.

## REFERENCES


[1] Akyildiz, Ian F., et al. "Wireless sensor networks: a survey." *Computer networks* 38.4 (2002): 393-422.

[2] S. Loveland, et.al, "Leveraging virtualization to optimize high-availability system configurations", *IBM Systems Journal*, vol. 47, no. 4, 2008. 591-604.

[3] Andréu, Javier, Jaime Viúdez, and Juan Holgado. "An ambient assisted-living architecture based on wireless sensor networks." *3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008*. Springer Berlin/Heidelberg, 2009. 239-248.

[4] Leontiadis, Ilias, et al. "SenShare: transforming sensor networks into multi-application sensing infrastructures." *Wireless Sensor Networks* (2012): 65-81.

[5] Jayasumana, Anura P., Qi Han, and Tissa H. Illangasekare. "Virtual sensor networks-A resource efficient approach for concurrent applications." *Information Technology, 2007. ITNG'07. Fourth International Conference on*. IEEE, 2007. 111-115.

[6] Levis, Philip, and David Culler. "Maté: A tiny virtual machine for sensor networks." *ACM Sigplan Notices*, Vol. 37. No. 10. ACM, 2002. 85-95.

[7] Yu, Yang, et al. "Supporting concurrent applications in wireless sensor networks." *Proceedings of the 4th international conference on Embedded networked sensor systems*. ACM, 2006. 139-152.

[8] Bhattacharya, Amiya, Meddage S. Fernando, and Partha Dasgupta. "Community Sensor Grids: Virtualization for sharing across domains." *Proceedings of the First Workshop on Virtualization in Mobile Computing*. ACM, 2008. 49-54.

[9] Navarro, Monica, et al. "VITRO architecture: Bringing Virtualization to WSN world." *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*. IEEE, 2011. 831-836.

[10] Bandara, H. M. N., Anura P. Jayasumana, and Tissa H. Illangasekare. "Cluster tree based self organization of virtual sensor networks." *GLOBECOM Workshops, 2008 IEEE*. IEEE, 2008. 1-6.

[11] Bhatti, Shah, et al. "MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms." *Mobile Networks and Applications* 10.4 (2005): 563-579.

[12] Smith, Randall B. "SPOTWorld and the Sun SPOT." *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*. IEEE, 2007. 565-566.

[13] Dunkels, Adam, Bjorn Gronvall, and Thiemo Voigt. "Contiki-a lightweight and flexible operating system for tiny networked sensors." *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*. IEEE, 2004. 455-462.

[14] Simon, Doug, et al. "Java™ on the bare metal of wireless sensor devices: the squawk Java virtual machine." *Proceedings of the 2nd international conference on Virtual execution environments*. ACM, 2006. 78-88.

[15] Lua, Eng Keong, et al. "A survey and comparison of peer-to-peer overlay network schemes." *IEEE Communications Surveys and Tutorials* 7.2 (2005): 72-93.

[16] Shelby, et al., "Constraint Application Protocol (CoAP)", *IETF, Internet-Draft,* draft-ietf-core-coap-13.txt (work in progress), 2012

[17] Ali, Muneeb, and Koen Langendoen. "A case for peer-to-peer network overlays in sensor networks." *International Workshop on Wireless Sensor Network Architecture (WWSNA'07)*, 2007. 56-61.

[18] Al-Mamou, AA-B., and Houda Labiod. "ScatterPastry: An overlay routing using a DHT over wireless sensor networks." *Intelligent Pervasive Computing, 2007. IPC. The 2007 International Conference on*. IEEE, 2007. 274-279.

[19] Gong, Li. "JXTA: A network programming environment." *Internet Computing, IEEE* 5.3 (2001): 88-95.