Cloud-based cost-efficient application and service provisioning in virtualized wireless sensor networks

Imran Khan

THÉSE DE DOCTORAT
TÉLÉCOM SUDPARIS – INSTITUT MINES-TÉLÉCOM
EN CO-ACCREDITATION
AVEC L'UNIVERSITÉ PIERRE ET MARIE CURIE - PARIS 6

École Doctorale Informatique, Télécommunication et Électronique de Paris

Spécialité
Informatique et Réseaux

Présentée par
Imran KHAN

Cloud-based Cost-efficient Application and Service Provisioning in Virtualized Wireless Sensor Networks

Soutenue le 08/07/2015 devant le jury composé de:

Rapporteurs:

M. Ahmed KARMOUCH	Professeur à Université d'Ottawa, Canada
M. Mika YLIANTTILA	Professeur à Université d'Oulu, Finlande

Examinateurs:

M. Guy PUJOLLE	Professeur à UPMC, Paris 6, France
M. Ernesto DAMIANI	Professeur à Università degli Studi di Milano, Italie
M. Laurent CIAVAGLIA	Chercheur scientifique principal à Alcatel-Lucent, France
Mme. Imen Grida BEN YAHIA	Ingénieur de Recherche à Orange Labs, France

Co-encadrant de Thèse:

M. Roch GLITHO	Professeur Associé à Université Concordia, Canada

Directeur de Thèse:

M. Noël CRESPI	Professeur à Télécom SudParis, France

Thèse numéro : 2015TELE0019

—To my late Father,

Acknowledgements

First of all, I would like to thank Prof. Noël Crespi, my thesis director and supervisor for giving me all the support, encouragement, guidance and freedom one could wish for. It was absolute pleasure to work with him and to gain valuable experience.

It gives me immense pleasure to acknowledge and convey my heart felt appreciation for the assistance and the feedback given by Prof. Roch Glitho, my thesis co-supervisor. He taught me the principles of research, with lots of patience, and his insight, constant motivation and advise helped my work go in the right direction.

I'd like to thank the reviewers of my thesis; Prof. Ahmed Karmouch (University of Ottawa, Canada) and Prof. Mika Ylianttila (University of Oulu, Finland) for their valuable feedback to improve the quality of my work.

I would also like to show my gratitude to my co-authors, Dr. Fatna Belqasmi (Zayed University, UAE), Fatima Zahra Errounda, Rifat Jafrin, Dr. Jagruti Sahoo & Dr. Sami Yangui (Concordia University, Canada) and Son Han (Télécom SudParis, France) for their valuable contribution in our joint research work. Special thanks goes to Dr. Ángel Cuevas (Universidad Carlos III de Madrid, Spain), Dr. Roberto Minerva (Telecom Italia, Italy), Dr. Djamal Zeghlache (Télécom SudParis, France), Dr. Preston Rodrigues and Dr. Paul Polakos (CISCO, USA) for their useful comments and suggestions to improve my work.

I am also grateful to my colleagues at Service Architecture Lab, TSP; they have made available their valuable support in a number of ways. Special thanks goes to secretary of RS2M Department Valerie Mateus; she was always very kind and generous in helping me to solve tedious administrative tasks.

My profound love and respect goes to my Mother – Hurmat Jahan, my Sister – Yasmeen, and my Brother – Javeed, who always boosted me with encouragement to achieve the new milestones in my life. No words can express my gratitude for my landlady, Mme. Evangelina Sousa who graciously hosted me all these years in Evry.

Finally, throughout my research journey at Télécom SudParis, I have met, interacted with and studied the work of number of excellent people. They, on a smaller or a larger scale influenced me and helped me to see things from different perspective. I thank them for their vision, shared knowledge and contributions to make this a better world.

Imran KHAN
Evry
08 July 2015

Abstract

Wireless Sensor Networks (WSNs) are becoming ubiquitous and are used in diverse applications domains. They are the cornerstones of the emerging Internet-of-Things (IoT) paradigm. Traditional deployments of WSNs are domain-specific, with applications usually embedded in the WSN, precluding the re-use of the infrastructure by other applications. This can lead to redundant deployments. Now with the advent of IoT, this approach is less and less viable. A potential solution lies in the sharing of a same WSN by multiple applications and services, even including applications and services that were not envisioned during the WSN deployment. This will allow resource- and cost-efficiency.

Two major developments have led to this potential solution. One is the advancements in hardware and software in WSN domain. As WSNs' nodes are becoming more and more powerful, it is getting more and more pertinent to research how multiple applications could share the very same WSN deployments. The second development is the Cloud Computing paradigm that promotes resource- and cost-efficiency by applying the virtualization concept to the available physical resources. As an enabler technology, virtualization can decouple WSN infrastructure from the applications running on the infrastructure. This thesis focuses on the cloud-based cost-efficient application and service provisioning in virtualized WSNs. It makes the following contributions.

First, an extensive state-of-the-art review is presented that introduces the basics of WSN virtualization and motivates its pertinence with carefully selected scenarios. Existing works are presented in detail and critically evaluated using a set of requirements derived from the scenarios. The pertinent research projects are also reviewed. Several research issues are also discussed with hints on how they could be tackled. This contribution substantially improves current state-of-the-art reviews in terms of the scope, motivation, details, and future research issues.

The second contribution consists of two parts: the first part is a novel multilayer WSN virtualization architecture that allows the provisioning of multiple applications and services over the same WSN deployment. It is applicable to new generation/powerful as well as resource-constrained WSN nodes. The proposed architecture provides platform independence and uses separate interfaces for data and control messages. It is implemented and evaluated using a scenario-based proof-of-concept prototype using Java SunSpot kit. The second part of this contribution is the extended architecture that allows virtualized WSN infrastructure to interact with a WSN Platform-as-a-Service (PaaS) at a higher level of abstraction. Through these enhancements a WSN PaaS can provision WSN applications and services to the end-users as Software-as-a-Service (SaaS). The enhancements are based on the identified fundamental differences between virtualized WSN Infrastructure-as-a-Service (IaaS) and traditional IaaS. Early results are presented based on the implantation of en-

hanced architecture using Java SunSpot kit.

The third contribution is a novel data annotation architecture for semantic applications in virtualized WSNs. It allows in-network data annotation and uses overlays as the cornerstone. We use domain-independent base ontology to annotate the sensor data, which is then forwarded to the PaaS where domain-specific ontologies exist. A proof-of-concept prototype, based on a scenario, is developed and implemented using Java SunSpot, AdvanticSys Kits and Google App Engine.

The fourth and final contribution is the enhancement to the proposed data annotation architecture on two fronts. One is the extension to the proposed architecture to support ontology creation, distribution and management. The second front is a heuristic-based genetic algorithm used for the selection of capable nodes for storing the base ontology. The ontology management extension is implemented and evaluated using a proof-of-concept prototype using Java SunSpot kit, while the simulation results of the algorithm are presented.

Keywords: Wireless Sensor Networks; Internet-of-Things (IoT); Virtualization; WSN Virtualization; Cloud Computing; Semantic Web; Data Annotation; Overlays

Résumé

Des Réseaux de Capteurs Sans Fil (RdCSF) deviennent omniprésents et sont utilisés dans diverses applications domaines. Ils sont les pierres angulaires de l'émergence de l'Internet des Objets (IdO) paradigme. Déploiements traditionnels de rcsfs sont spécifiques au domaine, avec des applications généralement incrustés dans le RdCSF, excluant la ré-utilisation de l'infrastructure par d'autres applications. Cela peut conduire à des déploiements redondants. Maintenant, avec l'avènement de l'IdO, cette approche est de moins en moins viables. Une solution possible réside dans le partage d'une même WSN par de multiples applications et de services, y compris même les applications et les services qui ne sont pas envisagées lors du déploiement de WSN. Cela permettra de ressources et de coût-efficacité.

Deux principaux développement ont conduit à cette solution potentielle. L'un est le progrès dans le matériel et les logiciels de RdCSF domaine. Comme les rcsfs' noeuds sont de plus en plus puissantes, il devient de plus en plus pertinente pour la recherche comment plusieurs applications pourraient partager le même RdCSF déploiements. La deuxième évolution est le paradigme de Cloud Computing qui favorise la ressource- et le rapport coût-efficacité en appliquant le concept de virtualisation à la ressources physiques disponibles. À titre d'outil habilitant la technologie, la virtualisation peut découpler RdCSF infrastructure à partir d'applications exécutées sur l'infrastructure. Cette thèse se concentre sur le nuage-based application rentable et service provisioning dans les environnements virtualisés les rcsfs. Il fait les contributions suivantes.

Tout d'abord, un vaste état de la revue d'art est présenté qui présente les principes de base de RdCSF la virtualisation et sa pertinence avec précaution motive les scénarios sélectionnés. Ouvrages existants sont présentés en détail et évaluées de façon critique en utilisant un ensemble d'exigences dérivées de ces scénarios. Les projets de recherche pertinents sont également examinés. Plusieurs questions de recherche sont également discutés avec des astuces sur la manière dont ils pourraient être abordés. Cette contribution améliore sensiblement l'état actuel des critiques d'art en termes de la portée, de la motivation, détails, et des questions de recherche futures.

La deuxième contribution se compose de deux parties: la première partie est une nouvelle architecture de virtualisation RdCSF multicouche permet le provisionnement de plusieurs applications et services sur le même RdCSF déploiement. Il est applicable à nouvelle génération/puissants aussi bien que des ressources limitées et noeuds RdCSF L'architecture proposée prévoit l'indépendance de la plate-forme et utilise des interfaces séparées pour les messages de contrôle et de données. Il est mis en oeuvre et évalués par l'élaboration d'un scénario à base de prototype de validation de principe à l'aide de Java kit SunSpot. La deuxième partie de cette contribution est l'architecture étendu qui permet d'interagir d'infrastructure RdCSF virtualisé avec un RdCSF PaaS à un plus haut niveau d'abstraction.

Grâce à ces améliorations un RdCSF FQA peuvent provisionner RdCSF applications et services pour les utilisateurs finaux comme Software-as-a-Service (SaaS). Les améliorations sont basées sur les différences fondamentales entre virtualisé et traditionnel RdCSF IaaS IaaS. Les premiers résultats sont présentés en fonction de l'implantation de l'architecture évoluée à l'aide de Java kit SunSpot.

La troisième contribution est une nouvelle architecture d'annotation des données pour les applications sémantiques dans les environnements virtualisés les rcsfs. Il permet l'annotation des données dans le réseau et utilise des superpositions comme étant la pierre angulaire. Nous utilisons une ontologie de base de domaine indépendant pour annoter les données du capteur, qui est ensuite transmis à la plate-forme en tant que service (PaaS) où spécifique domaine ontologies existent. Un prototype de validation de principe, basé sur un scénario, est élaboré et mis en oeuvre à l'aide de kits AdvanticSys, Java SunSpot, et Google App Engine.

La quatrième et dernière contribution est la mise en valeur de l'architecture d'annotation de données proposée sur deux fronts. L'un est l'extension à l'architecture proposée pour soutenir la création de l'ontologie, la distribution et la gestion. Le second front est un algorithme génétique sur la base heuristique utilisé pour la sélection de noeuds capables de stocker l'ontologie de base. L'extension de la gestion de l'ontologie est mis en œuvre et évaluée en utilisant un prototype de validation de concept en utilisant le kit Java SunSpot. Bien que les résultats de simulation de l'algorithme sont présentés.

Mots-clés: Réseaux de capteurs sans fil; Internet-de-Objets (IdO); Virtualisation; Virtualisation de Réseaux de Capteurs Sans Fil; Cloud Computing; Web Sémantique; Annotation des Données; Réseaux de Superpositions

List of Publications

This thesis is based on the following original research publications.

I. Imran Khan, Fatna Belqasmi, Roch Glitho, Noël Crespi, Monique Morrow, Paul Polakos, "Wireless Sensor Network Virtualization: A Survey", *Communications Surveys & Tutorials, IEEE, vol.PP, no.99, pp.1,1 doi: 10.1109/COMST.2015.2412971*, March 2015 (Impact Factor 6.49)

II. Imran Khan, Fatna Belqasmi, Roch Glitho, Noël Crespi, Monique Morrow, Paul Polakos, "Wireless Sensor Network Virtualization: Early Architecture and Research Perspectives", *Accepted for publication in May/June issue of IEEE Network Magazine*, 2015 (Impact Factor 3.720)

III. Imran Khan, Fatima Zahra Errounda, Sami Yangui, Roch Glitho, Noël Crespi, "Getting Virtualized Wireless Sensor Networks' IaaS Ready for PaaS", *IoTIP-15 Workshop in 11th IEEE International Conference on Distributed Computing in Sensor Systems (IEEE DCOSS 2015) conference*, 2015, June 10-12, Fortaleza, Brazil.

IV. Imran Khan, Roch Glitho, Noël Crespi, "Design and Analysis of Virtualization Framework for Wireless Sensor Networks", *in proceedings of 5th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks PhD forum 2013 (IEEE WoWMoM 2013 PhD forum)*, 2013, June 4-7, Madrid, Spain.

V. Imran Khan, Fatna Belqasmi, Roch Glitho, Noël Crespi, "A Multi-Layer Architecture for Wireless Sensor Network Virtualization", *in proceedings of 6th Joint IFIP Wireless and Mobile Networking Conference (WMNC'2013)* 2013, April, 23-25, Dubai, UAE.

VI. Imran Khan, Rifat Jafrin, Fatima Zahra Errounda, Roch Glitho, Noël Crespi, Monique Morrow and Paul Polakos, "A Data Annotation Architecture for Semantic Applications in Virtualized Wireless Sensor Networks", *in proceedings of 14th IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2015) – Technical Session*, 2015, May 11-15, Ottawa, Canada. (CORE rank A, Acceptance rate 26%)

VII. Imran Khan, Rifat Jafrin, Jagruti Sahoo, Roch Glitho, and Noël Crespi, "Towards Provisioning of Semantic Applications over Virtualized Wireless Sensor Network Infrastructure-as-a-Service", *in preparation for submission in IEEE Transactions on Cloud Computing.*

VIII. Imran Khan, Jagruti Sahoo, Son Han, Roch Glitho, and Noël Crespi, "A Genetic Algorithm-based Solution for Efficient In-network Sensor Data Annotation in Virtualized Wireless Sensor Networks", *under review in IEEE CCNC 2016 conference.*

In addition, the author has also authored or co-authored one journal, three conference papers and three versions of an IETF draft in the relevant research areas, including Cloud Computing, Machine-to-Machine communication, 4G EPC Network, service composition, Quality-of-Experience in wireless networks, and Future Internet.

Table of contents

List of Tables

List of Figures

"— Possibilities do not add up. They multiply."

–Paul Romer

Chapter 1

Introduction

Contents

With their ability to sense real-world events, Wireless Sensor Networks (WSNs) act as a bridge between physical and virtual worlds. These WSNs are composed of nodes that are amalgamations of micro-electro-mechanical systems, wireless communications and digital electronics, and have the ability to sense their environment, perform computations and communicate [1], [2]. Over the last decade or so we have seen a proliferation of WSN applications and services in multitude of application domains, such as health care, building automation, agriculture, smart cities and security & surveillance.

As demand for new, exciting and innovative applications and services grows, it becomes imperative for the service providers to efficiently reuse existing infrastructure until it reaches the end of its life-cycle. In this situation, application and service provisioning requires a well-defined process to not only consider the requirements and constraints of the new applications and services, but also to take into account the specifics of the respective communication networks over which the applications and services will be deployed. In short, concrete architectural solutions are required that decouple the applications and services from the deployed infrastructure.

However, there are some inherit limitations regarding how the applications and services have been provisioned in WSNs so far. Traditionally, there has been a tight coupling between

the applications and services, and the deployed WSN infrastructure. This has considerably limited the scope of innovation and has promoted domain-specific solutions, which are difficult to reuse by new users especially from different domains. Because of the resource constraints in WSNs, tailored solutions are often bundled with the deployment of the network infrastructure. With end-users and their requirements predetermined, these tailored solutions serve their purpose but only until new applications and services are contemplated. It becomes inherently difficult to accommodate new applications and services simply because the deployed solutions were never designed for them and any changes require considerable capital and human effort. To tackle this, inefficient approaches like redundant infrastructure deployments are used. However, such approaches are becoming less and less appealing due to the associated costs in general and availability of alternative technologies in particular, that an potentially allow the reuse of the deployed WSN infrastructure.

There are two new technologies (or rather paradigms shifts), in particular, that are beginning to revolutionize not only the usage of applications and services but also how they are provisioned. These technologies are Internet-of-Things (IoT) and Cloud Computing. IoT is termed as next technological paradigm that aims to realize communication between many types of objects, machines and devices on a massive scale [3]. IoT is expected to have a profound impact on our daily lives, even bigger than the Internet, consequently raising many challenges to address [4]. WSNs can be considered as one of the basic constituents of IoT because they can help users (humans or machines) to interact with their environment and react to real-world events. IoT not only provides opportunities for plurality of heterogeneous devices to connect and communicate with each other but also makes it possible for application and service providers to offer new innovative applications and services.

Cloud computing offers elastic provisioning of large-scale infrastructures to multiple concurrent users [5]. Through its three facets – Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) – cloud computing effectively decouples the applications and services from the deployed infrastructure [6] and allows multiple actors to use it as and when required. Service providers use platforms (offered as PaaS) to provision applications and services that are offered as SaaS on a pay-per-use basis to end-users or other applications. The platforms ease the provisioning process by adding levels of abstraction to the infrastructure offered as IaaS. The infrastructure is the actual dynamic pool of resources used by the applications. Cloud computing has several inherent benefits such as, efficient usage of resources, scalability, elasticity, and rapid development and introduction of new applications. Cloud computing uses the established concept of virtualization to provide concurrent access to resources through abstractions. Virtualization allows users (applications and services) to utilize resources as dedicated to them whereas in reality multiple users access them concurrently [7]. This is usually achieved by dedicated software such as a hypervisor or a middleware.

The work in this thesis targets WSNs and applies the concepts of cloud computing to

provision applications and services in an efficient manner. The main focus of this thesis is to offer WSN deployments as IaaS enabling their efficient usage by concurrent applications and services. To accomplish this, the thesis makes four contributions. First contribution is a detailed state-of-the-art review which has been missing from the literature. The second contribution is a novel architecture to provision traditional WSN applications over a deployed WSN and its extension to offer a deployed WSN as IaaS and allow it to interact with PaaS. Third contribution is a novel architecture to provision semantic-based WSN applications in a domain independent way. The fourth contribution is an ontology development and management tools to easily create ontologies and a heuristic-based algorithm to facilitate the selection of capable nods for the sensor data annotation.

1.1 Motivation and Research Problems

Since their mainstream introduction towards the end of 20th century, WSN deployments have been used as means to bridge the gap between the physical world and the virtual world. With their ability to sense, compute and communicate, WSNs provide their users with the ability to react to various physical phenomenon and take required actions, which sometimes involves invoking actuators to generate some mechanical response. Over the years, the use of WSNs has increased and now there are many application domains where WSNs help in solving real-world problems [2]. Fig. 1 shows a generalized view of the typical WSN application domains and example applications.

Despite their potential benefits, by and large WSNs have remained domain-specific and task-oriented and reusing the same deployed WSN for new application was prohibitively expensive due to associated cost and human effort.

However, as capable sensor platforms emerged and the concept of IoT paradigm grabbed attention, it soon became apparent that sharing WSN deployments can have several benefits. For example, applications that were not envisioned a priori may be able to utilize existing WSN deployments. The second benefit is the elimination of tight coupling between WSN services/applications and WSN deployments. This allows experienced as well as novice application developers to develop innovative WSN applications without knowing the technical details of the WSNs involved. Another benefit is that WSN applications and services can utilize as well as be utilized by third-party applications. It can also help to define a business model, with roles such as WSN provider, virtual WSN provider and WSN service provider.

The last benefit is particularly interesting since it can pave way for new and innovative applications like the ones mentioned at the beginning of this chapter. For example a city-wide public sensor deployment can be shared between the citizens as well as various government departments such as fire, traffic & event management, surveillance and security.

Many researchers now consider sharing deployed WSN among multiple applications and provide its motivation. According to the authors in [8], the sharing of WSN deployments

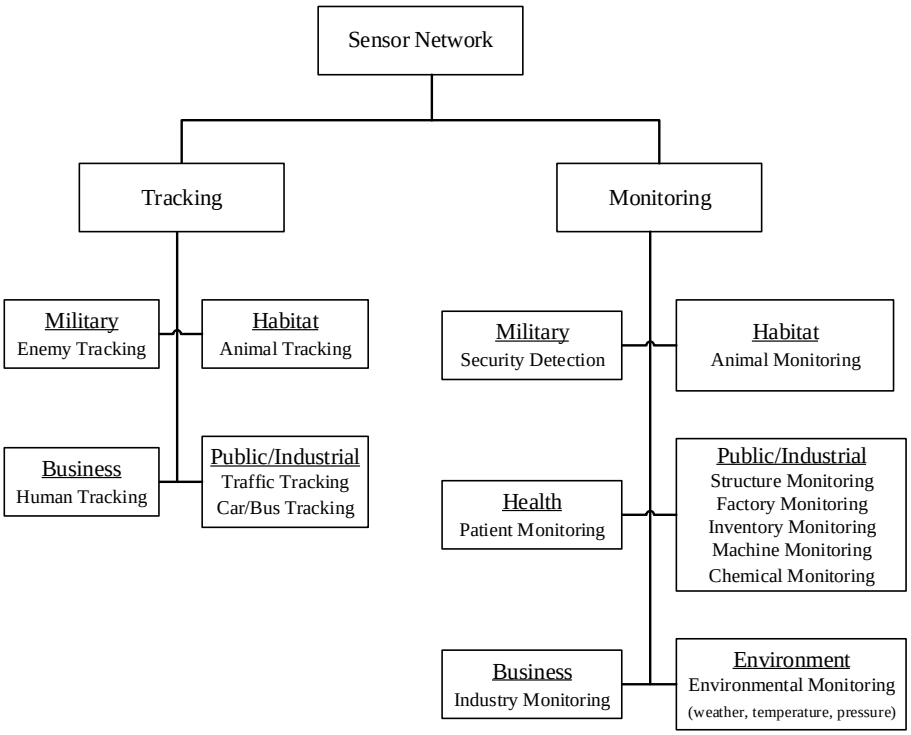


Figure 1.1: WSN application types, domains and example applications as shown in [2]

is a powerful enabler for information sharing in the context of IoT by using it along with data analysis techniques. A smart city environment is considered in [9], where sharing of WSN deployments could be used to efficiently utilize the deployed infrastructure. To achieve this type of utilization, the use of multiple concurrency models is advised, depending on the usage context. In [10] the sharing of WSN deployments is envisioned as an important technology to create large-scale sensor platforms that are used to satisfy efficient usage of network resources.

There are two approaches to allow multiple applications to share the deployed WSN resources. One is to allow multiple applications to share the gathered data from a WSN. In this approach a sink/gateway node collects all data from the WSN and then it is shared among multiple users, for example in [11] WSNs are merged into the cloud by sending observed sensor data through a host manager which lies outside the WSN. The host manager simply collects the sensor data, profiles/aggregates it and then allows multiple applications to use it for their purposes. The second approach is to use the capabilities of the individual sensor nodes to execute multiple applications tasks concurrently and allow applications to group these sensor nodes together according to their requirements [12].

The key difference between the two approaches is that the former approach allows sharing

of WSN data among multiple users, while latter approach allows sharing of WSN nodes by multiple applications. This thesis is focused on the second approach because it allows to provision more innovative applications over the deployed WSN even the ones which are not known during the deployment. The emergence of capable sensors will greatly improve the efficiency of the deployed WSN and will also encourage new business models.

However, provisioning applications and services in such shared environment, where WSN nodes execute multiple application tasks, is not trivial. During the state-of-the-art review, the lack of applicable and demonstrated architectural solutions was observed.

With this motivation, used as basis, this thesis addresses the following questions:

- Q1: *What is the current state-of-the-art dealing with the possibility of utilizing WSNs for multiple applications and services? What is the taxonomy and relevant research works in this area?*
- Q2: *How multiple and concurrent WSN applications can be provisioned over a deployed WSN? What is an efficient approach to build architecture to accomplish this? How WSN infrastructure can interact with a PaaS? What features must be supported by a WSN infrastructure to allow PaaS to develop and deploy WSN applications and services?*
- Q3: *How semantic web technologies can be used to efficiently provision WSN applications? In particular how semantic-based applications and service can receive annotated sensor data in real-time? Also how sensor data annotation can be performed in a distributed manner, in standardized way while making sure that future enhancements to the WSN infrastructure are also taken care of?*
- Q4: *How to enable a WSN IaaS owner to provide mechanism to support semantic-based application without making the deployed infrastructure application domain-specific? How to have an efficient and robust mechanism to annotate sensor data that is applicable to resource-constrained environments such as WSNs?*

1.2 Concepts and Research Methodology

This thesis focuses on proposing novel architectural solutions to enable cloud-based cost-efficient application and service provisioning in wireless sensor networks. To that end, three architectural solutions, one heuristic-based genetic algorithm and a comprehensive state-of-the-art review are presented to address the research questions, identified in Section 1.1. The following important concepts and research methodology is used for this thesis.

1.2.1 Concepts

The well-established concept of virtualization, when coupled with WSNs, gives us virtualized WSNs (vWSNs). This concept is core of all the work done in this thesis. Virtualization allows

for the abstraction and sharing of computer and network resources, as well as the co-existence of several entities on the same physical node [13]. WSNs can be virtualized at node level [14] and also at network level [15]. It is assumed that in vWSNs all sensors support some form of node-level virtualization themselves, i.e. the capability to run concurrent tasks.

The concept of Cloud Computing [16] is used to offer WSN IaaS to multiple applications and services. This is achieved by using appropriate abstractions to hide the complexity of the underlying WSN IaaS and allow applications and services to receive data using a standard and homogeneous interface. By using this concept, it becomes possible for third party WSN infrastructure owners to offer their infrastructure to developers to build new application and service and offer them as SaaS. Typical examples of these third party WSN infrastructure owners can be a city administration or a global scientific research organization.

The architectures, proposed in this thesis, use the concept of overlays as cornerstone. Overlays have several advantages like, they are distributed, lack central control and allow resource sharing [17]. The concept of overlays is used to provision applications and services by logically grouping sensors, executing similar tasks, to exchange data. Overlays are also used for network operation functions, such as sensor data annotation and storage of the ontology. Additionally, the concepts of super peer and client peer [18] are used to store and retrieve the ontology in a distributed manner.

Another concept, used this thesis, is semantic annotation, which helps in adding additional metadata to the raw data to enhance its meanings and to provide situational awareness to the end user [19]. However, semantic annotation need domain concepts and relationships between them. These concepts and relationships are provided by the ontologies. The work in this thesis proposes a base ontology to annotate sensor data. The base ontology is developed as an extension of the Semantic Sensor Network (SSN) Ontology from W3C Semantic Sensor Network Incubator Group [20]. The base ontology is independent of any application domain to make it usable for all application domains.

1.2.2 Research Methodology

In this thesis, the following research methodology is used to solve the identified research questions.

First a step-by-step approach is used to design concrete architectural solutions. The first step begins with the high-level specifications of different network entities and their interactions with various actors. This includes inputs to and expected outputs from various entities specified in the architecture, the requirements (technical and non-technical) specification and the architectural principles that serve as foundation for the design of the architectures. The second step is to make design choices, which include identifying the right set of technologies (e.g. protocols, data format/encoding schemes, and communication types). The final step is implementation and testing of the designed solutions over real networks or using development

kits for the proof-of-concept prototypes.

All proposed architectures are implemented and evaluated using proof-of-concept prototypes using two sensor kits, Java SunSpot kit [21] and AdvanticSys kit [22]. Java SunSpot are high-end sensor nodes that have better processing and storage capabilities than earlier generation of sensor nodes. They have multiple on-board sensing capabilities, are J2ME-based hence easy to program and offer advanced features such as support for Over-the-Air commands. Java SunSpots do not use any operating system, instead they have Squawk VM [23] running over the hardware. AdvanticSys kit consists of TelosB motes that represent early generation of sensor nodes. They also have multiple on-board sensors but have very low processing and storage capabilities. They support popular operating systems like TinyOS [24] and Contiki [25]. In this thesis, Contiki OS is used. These early generation sensors are mainly used to demonstrate the support for legacy sensors and heterogeneity in the proposed architectures.

The following is the detail of the performance metrics (or measures) used to evaluate the proposed architectures. The paper in annex B uses HTTP POST Delay, Overlay Creation Delay and Fire Notification Delay as performance evaluation metrics. The paper in annex C uses Virtual Sensor Creation Delay and Virtual Sensor Start Time as performance evaluation metrics. The paper in annex F uses End-to-End Delay, Ontology Download Time, Discovery Delay, Expected Operation Time of Java SunSpots, and the Impact of tasks on current draw from Java SunSpots battery as performance evaluation metrics. The paper in annex G uses Overlay Creation Delay, Annotation Ontology Dissemination Time and Ontology Download Time as performance evaluation metrics. Finally the paper in annex H uses the value of fitness function for network size of 1000, 2000 and 3000 sensors. It also presents the impact of using different crossover rates on the final solution.

1.3 Contributions of the Thesis

This thesis makes four contributions in total which are described here briefly and linked to the original research papers addressing them. Detailed contribution of each original research paper is presented in subsequent chapters.

The *first* contribution of the thesis is a detailed state-of-the-art review. Since the thesis is addressing a new direction concerning provisioning of concurrent application and service provisioning in WSNs, we were particularly interested to find out how various solutions in WSN domain has evolved and what are the recent trends. Particularly what kind of research projects are being done by academics as well as industries. The paper in annex A is the outcome of this contribution and provides are comprehensive state-of-the-art review of the existing works. Such review was missing from the existing literature. The paper starts with the basics and motivation for WSN virtualization using carefully selected scenarios. A taxonomy of the existing works is presented by identifying three categories; node-level virtu-

alization solutions, network-level virtualization solutions and hybrid solutions. Each work is characterized using its properties and is evaluated using a set of requirements derived from the scenarios. Additionally several pertinent research projects are also reviewed. Towards the end of the paper, several pertinent research issues are discussed with hints on how they could be tackled.

The *second* contribution of the thesis is presented in two parts. First, a novel multilayer WSN virtualization architecture is presented that supports provisioning of sensor applications and services over multiple WSN deployments. Using our architecture, virtualized WSNs can be utilized by concurrent applications and services. The papers in annex B, D and E are related to this contribution. The overall concept of WSN virtualization framework was presented in the paper in annex D along with possible research avenues. The initial version of the architecture was presented in the paper in annex E without any performance results. The paper in annex B presents the final architecture and the proof-of-concept implementation along with results. The proposed architecture uses the concept of overlays to deploy new applications and services. It is applicable to new generation/powerful (Java SunSpots) as well as resource-constrained WSN nodes (TelosB). To allow resource-constrained WSN nodes to participate in overlay related operations, we introduced the concept of Gates-to-Overlays (GTO) nodes, which perform such tasks on their behalf. The proposed architecture provides platform independence and uses separate interfaces for data and control messages. It is implemented and evaluated by developing a scenario-based proof-of-concept prototype using Java SunSpot kit. The results show the viability of our proposed architecture.

The second part of this contribution is presented in the paper in annex C. It addresses the challenges that are faced when WSN IaaS has to interact with PaaS for application and service provisioning. In this contribution we identified fundamental differences between traditional IaaS and WSN IaaS precluding the straightforward re-use of traditional PaaS offerings. Keeping these differences in view, we extend our previous WSN virtualization architecture to make it a true virtualized WSN IaaS to interact with WSN PaaS. The differences between Traditional IaaS and WSN IaaS are identified in terms of resources, capabilities and protocol support. In traditional IaaS we have the concept of Virtual Machine (VM) that allows time and resource sharing of host machines by partitioning them into multiple dedicated execution environments [26]. In WSN IaaS we have the concept of Virtual Sensor (VS), which is a logical representation of the physical sensor to allow sharing of its sensing capabilities (e.g., temperature and light sensing capabilities) [12]. We identified seven concrete differences between VMs and VSs and using them proposed several enhancements to the original architecture. Using Java SunSpot kit we demonstrated the feasibility of the proposed architecture using a simple smart home scenario and implemented it as a standalone Java application.

The *third* contribution is novel data annotation architecture to provision semantic applications in virtualized WSNs. The paper in annex F describes this contribution. The proposed architecture supports in-network sensor data annotation and uses overlays as the

cornerstone. We introduce the concept of base ontology to annotate sensor data independently of any application domain. The idea is to offer a deployed WSN as IaaS to multiple, independent users. Since, it is hard to predetermine the type of applications that will be deployed over the WSN, the base ontology is created by extending the SSN Ontology from W3C Semantic Sensor Network Incubator Group to ensure its interoperability.

Two separate overlays are used, ontology overlay to store the base ontology in a distributed manner and annotation overlay to annotate the sensor data. The peers in the ontology overlay act as super peers while the peers in the annotation overlay act as client peers. The super peers store the base ontology and client peers request for it whenever they need to annotate sensor data. For each sensor, executing a task that requires annotation, there is a corresponding entity in the annotation overlay whose task to perform the annotation (and request ontology from super peer, if it does not have it already). Since it is not possible for resource-constrained sensors to support such functionality, we reuse the GTO nodes concept that perform this function on their behalf. Using GTO nodes that proposed architecture is applicable to both resourceful and resource-constrained sensors. A proof-of-concept prototype, based on a scenario, is developed and implemented using Java SunSpot, AdvanticSys Kits and Google App Engine. Three different prototype configurations are used for evaluation purposes.

The *fourth and the final* contribution is presented in the paper in annex G. It extends the work done in the paper in annex F to incorporate two important features to offer WSN IaaS. The first feature includes an easy to use mechanism to create, distribute and management of the base ontology for the WSN infrastructure owner. A web-based GUI application is developed that allows a WSN infrastructure owner to create and manage base ontology. The developed ontology should reflect the deployed WSN instead of any application domain. The deployed WSN infrastructure may include heterogeneous sensors hence the base ontology may become large making it difficult to store it in the WSN as mentioned in the third contribution. We propose to split the base ontology (according to the physical phenomena) into distinct portions and store each portion in the WSN.

This strategy demands an efficient algorithm to select capable nodes in the WSN to store thee portions of ontology. The *second* feature that we propose addresses this issue. We propose a heuristic-based multi-objective genetic algorithm used for the selection of capable nodes for storing the base ontology. It is important to mention that by capable nodes we mean new generation of sensors (mostly IP-capable sensors), GTO nodes (base station nodes, sink nodes). The algorithm takes into account the energy, storage space of the sensors and selects the ones that have maximum energy and storage space available. Once suitable nodes are identified, the portions of the base ontology are equally distributed among them. For example if the base ontology is split into temperature sensor portion and humidity sensor portion, there will be n nodes containing each of these portions. By replicating the base ontology portions, the architecture will be able to cope with node failures and network dynamics. The

simulation results of the algorithm are presented while the ontology management extension is implemented and evaluated using a proof-of-concept prototype using Java SunSpot kit.

1.4 Thesis Organization

The following chapters summarize the main contribution of the research papers, provide discussions and the ideas for future work. Chapter 2 discusses the background, motivation and summary of the state-of-the-art. It is a summary of Paper I. Chapter 3 describes WSN virtualization architecture to provision multiple application over deployed WSNs. It also discusses the architecture to enable interactions between WSN IaaS and PaaS, along with early performance measurements. This chapter is a summary of Papers II, III, IV and V. Chapter 4 discusses the sensor data annotation architecture and presents implementation details and performance evaluation results. It summarizes the contributions of Paper VI. Chapter 5 is devoted to ontology management tool and heuristic-based genetic algorithm for node selection, along with performance measurements. This chapter summarizes the contributions of Paper VII and Paper VIII. Chapter 7 presents the items for the future work and summary of the thesis. Finally the research papers are attached with this thesis in the following order. Paper I is in annex A, paper II in annex B, paper III in annex C, paper IV in annex D, paper V in annex E, paper VI in annex F, paper VII in annex G and paper VIII in annex H.

1.5 Overview of My Publications

Paper I in annex A is *"Wireless Sensor Network Virtualization: A Survey"*. It provides a comprehensive state-of-the-art review along with clear taxonomy of available solutions. Several research issues are also identified along with hints to solve them.

Paper II in annex B is *"Wireless Sensor Network Virtualization: Early Architecture and Research Perspectives"*. It presents a novel WSN virtualization architecture to provision multiple application over deployed WSNs along with prototype and performance measurements.

Paper III in annex C is *"Getting Virtualized Wireless Sensor Networks' IaaS Ready for PaaS"*. It presents the architectural enhancements made to the WSN virtualization architecture to enable interactions between WSN IaaS and PaaS. Details of prototype and performance measurements are also presented.

Paper IV in annex D is *"Design and Analysis of Virtualization Framework for Wireless Sensor Networks"*. It introduces the WSN virtualization problem with a high-level view

and possible avenues of research that could be explored.

Paper V in annex E is *"A Multi-Layer Architecture for Wireless Sensor Network Virtualization.* It presents an early version of the WSN virtualization architecture along with motivation in the form of a use case.

Paper VI in annex F is *"A Data Annotation Architecture for Semantic Applications in Virtualized Wireless Sensor Networks"*. It discusses a data annotation architecture that supports, in-network, real-time annotation of sensor data independent of any application domain. Details of prototype and performance measurements are also presented.

Paper VII in annex G is *"Towards Provisioning of Semantic Applications over Virtualized Wireless Sensor Network Infrastructure-as-a-Service"*. It presents an ontology development and management tool and builds on the architecture presented in Paper VI in annex F. Details of prototype and performance measurements are also presented.

Paper VIII in annex H is *"A Genetic Algorithm-based Solution for Efficient In-network Sensor Data Annotation in Virtualized Wireless Sensor Networks"*. It discusses a heuristic-based algorithm to select capable nodes for storing of ontology in a distributed manner. The performance measurements of the proposed algorithm are also presented.

The author is the lead contributor in all the papers and has lead implementation, prototyping, compilation of results and writing of the papers. The author worked with two masters students at Concordia University for Paper III, VI and VIII and assigned them specific tasks for implementation. For paper VII the author has worked with a PhD student at Telecom SudParis and assigned him implementation tasks. During the preparation of the papers, the author discussed the progress in the meetings and incorporated the suggestions/inputs given by other co-authors and supervisors.

"— Study the past if you would define the future..."

–Confucius

Chapter 2

Background and State-of-the-Art

Contents

2.1 Introduction

This chapter discusses the background, motivation and summary of the state-of-the-art in the area of WSN virtualization and is based on annex A. It addresses the following research question:

What is the current state-of-the-art dealing with the possibility of utilizing WSNs for multiple applications and services? What is the taxonomy and relevant research works in this area?

At the beginning of this thesis work, it was observed that there is lack of comprehensive state-of-the-art. Existing surveys [27] and [28] lacked the technical depth and critical review of existing works. In order to address this, we started with the basics of WSN virtualization by categorizing it into two categories. In order to show the pertinence of WSN virtualization, two motivating scenarios are discussed. Then by identifying the needs of various actors in these scenarios, eight requirements are identified. Each existing work is discussed, in detail, to show how it supports various categories of WSN virtualization. Twenty six works related to WSN virtualization are discussed in total and each one of them is evaluated using the identified requirements. Also characteristics of these works are identified, in each WSN virtualization category, to show their strengths and contributions. Recently this topic has gained attention from academic and industrial quarters as evident from several research projects. In Paper I, seven research projects are discussed. Finally several important research issues and their potential solutions are also presented.

2.2 Basics of WSN Virtualization

WSN virtualization can be broadly classified into two categories: Node-level virtualization and Network-level virtualization. The former allows execution of concurrent execution of multiple application tasks on a sensor node [29], while the later allows dynamic formation of logical groups of sensor nodes, where each group is dedicated to an application or a service. Node-level virtualization can be achieved by sequential (one-by-one) or simultaneous execution (by context switching/multi-threading) of application tasks on a sensor node. Network-level virtualization forms Virtual Sensor Network (VSN) which consists of a subset of a WSN's nodes dedicated to an application or a service at a given time [15]. VSNs ensure resource efficiency, because the remaining sensor nodes remain available for multiple applications (even the ones that had not been envisaged when the WSN was deployed), although not necessarily simultaneously. The basic concept of node-level virtualization is illustrated in Fig. 2.1 whereas Fig. 2.2 shows the two possible realizations of network-level virtualization.

2.3 WSN Virtualization – Motivating Scenarios

The best way to show the pertinence and need for WSN virtualization is through motivating scenarios. This way interactions between various actors and entities can be shown and based on these interactions, it becomes easy to derive requirements for each of these actors. The two scenarios described here are taken from the existing literature and are altered to illustrate the motivation and the benefits of using WSN virtualization in certain situations.

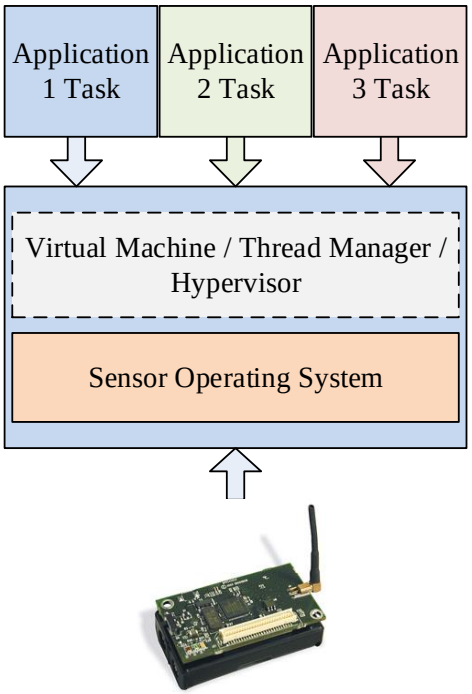
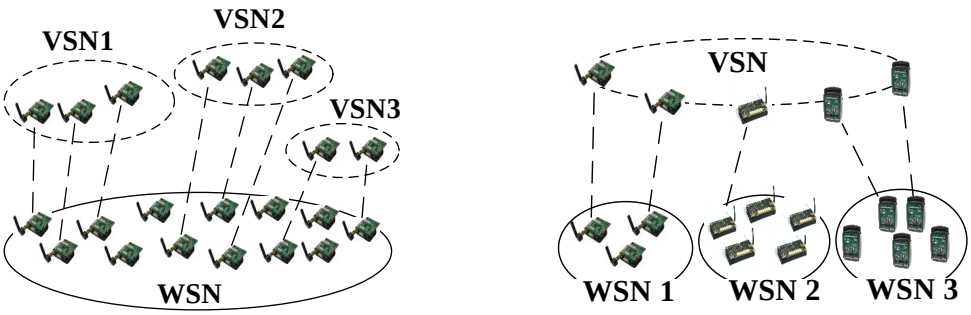


Figure 2.1: WSN Node-level Virtualization



(a) Multiple VSNs over single WSN

(b) Single VSN over multiple WSNs

Figure 2.2: WSN Network-level Virtualization

2.3.1 Fire Monitoring Scenario

Consider the example of a city near an area where brush fires are common [30]. We assume that the city administration is interested in the early detection of fire eruption and in its course, using a WSN and a fire contour algorithm to determine the curve, shape and direction of fire. One approach is that the city administration could deploy WSN nodes all over the

city (i.e., on each street and at individual houses), but this is not very efficient because some individuals may have already deployed WSN nodes in their homes to detect fires. A more efficient approach would be for the city administration to deploy WSN nodes to areas under its jurisdiction, i.e., streets and parks, and to re-use the WSN nodes already deployed in private homes.

In this scenario, two different applications share the same WSN infrastructure: one, belonging to home owners, is confined to private WSNs deployed in individual houses, and the other belongs to the city administration and shares the private WSN nodes with the WSN nodes deployed by the city administration. Periodic notification or query-based models are not suitable because the city administration application requires complete access to all the WSN nodes for adaptive sampling.

Another issue is that to execute a fire contour algorithm in a distributed fashion, WSN nodes need to exchange fire notification messages with each other. The query-based data exchange approach is not efficient as it will force the execution of the fire contour algorithm at a remote centralized location, since two WSN nodes located in their respective private domains cannot exchange data directly. An overlay network is one possible solution. This scenario illustrates the need for WSN virtualization, as two different users need to share a common resource, i.e., WSN nodes.

2.3.2 Heritage Building Monitoring

A real-world deployment of a WSN is presented in [31], in which a WSN is used to monitor the impact of constructing a road tunnel under an ancient tower in Italy, as it was feared that the tower could lose its ability to stand on its own and collapse during the construction. Now consider that there are three users interested in the fate of the tower. The first is the construction company, as it needs to make sure that the tower does not lose its ability to stand on its own, otherwise it will have to pay a heavy fine. The second user is the conservation board that routinely monitors all the ancient sites around the city, and the third user is the local municipality which will have to plan emergency remedial/rescue actions in case the tower falls during the construction.

It is quite possible that the conservation board has already deployed its own WSN to monitor the health of ancient sites including this tower. In this case the construction company and the local municipality can use the existing sensor nodes during the construction period.

In the absence of WSN virtualization, there are only two possible solutions. One is to rely on the information provided by the conservation boards application. However this information may not be at the required granularity level. Worse, some of the information that is needed might simply not be available because the requirements of the construction company and of the local municipality were not considered when the conservation board application was designed and implemented. The second solution is that each user deploys

redundant WSN nodes. Here WSN virtualization can play a pivotal role by fulfilling the requirements of each user.

2.4 WSN Virtualization – Requirements

From the motivating scenarios we derived a set of eight requirements that a comprehensive WSN virtualization solution should tackle. In Paper I, all existing works were evaluated using these requirements. Table 2.1 illustrates the evaluation of all existing works based on these requirements.

- *Support for node-level virtualization*: This fundamental requirement ensures that the sensor nodes can support the concurrent execution of multiple applications.
- *Support for network-level virtualization*: This concerns the ability of sensor nodes to dynamically form groups and execute application tasks together for individual application.
- *Support for application/service priority*: This is useful for mission-critical applications/services.
- *Platform-independence*: The solution should be independent of any particular hardware or software platform.
- *Support for resource discovery mechanism*: The solution should address both sensor and service discovery.
- *Support for resource-constrained sensor nodes*: It is important to allow the use of existing deployments of sensors (most of them are early generations) for WSN virtualization.
- *Support for heterogeneity*: The proposed solution should be applicable to a variety of WSN platforms with different capabilities (e.g. processing power, memory).
- *Ability to select sensor nodes for application tasks*: When multiple applications concurrently utilize a deployed WSN, selection of proper sensor nodes is very important because applications may have spatial and temporal requirements [21].

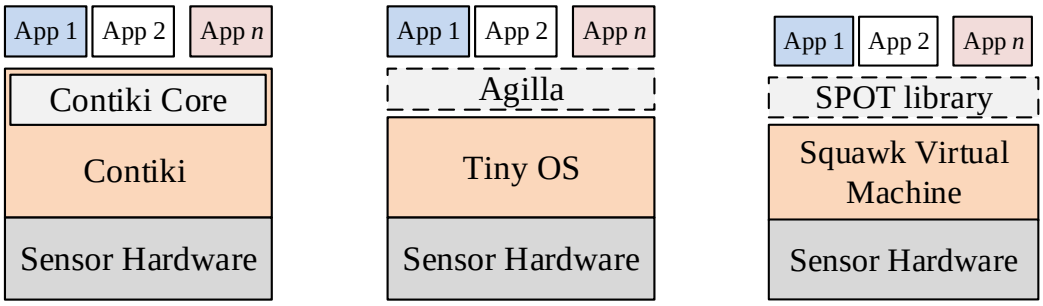
2.5 WSN Virtualization – Summary of State-of-the-Art

We categorize the existing work as Node-level virtualization, Network-level virtualization and Hybrid solutions. Hybrid solutions combine both node- and network-level virtualization. Each category is further classified based on the approaches used.

2.5.1 Node-Level Virtualization

Node-level virtualization can be realized by either *i)* a capable operating system like Contiki [25], *ii)* using a middleware like Agilla [39] or *iii)* by using a virtual machine like Squawk that directly runs over the sensors hardware. In the early-generation sensor nodes, the programming model of choice was event-driven, as it was simple to implement, but once its limitations were found, the thread-based approach was used to implement more complex and concurrent tasks in sensor nodes. Of all these works, TinyOS and Contiki have become extremely popular and have good community support. Contiki is now considered as a platform for the IoT [58] and has incorporated many innovative features over the last decade. RIOT [34] is a new work to design a capable OS to run C/C++ applications on heterogeneous sensor platforms. Fig. 2.3 shows the examples of node-level virtualization solutions.

The following characteristics are identified for node-level virtualization solutions: Programming mode, Programming Language, Separation between operating system and application tasks, Protocols supported, and Support for real-time applications.



(a) OS-based solution (e.g., Contiki)

(b) Middleware-based solution (e.g.,Agilla)

(c) Virtualmachine-based solution (e.g., Squawk VM)

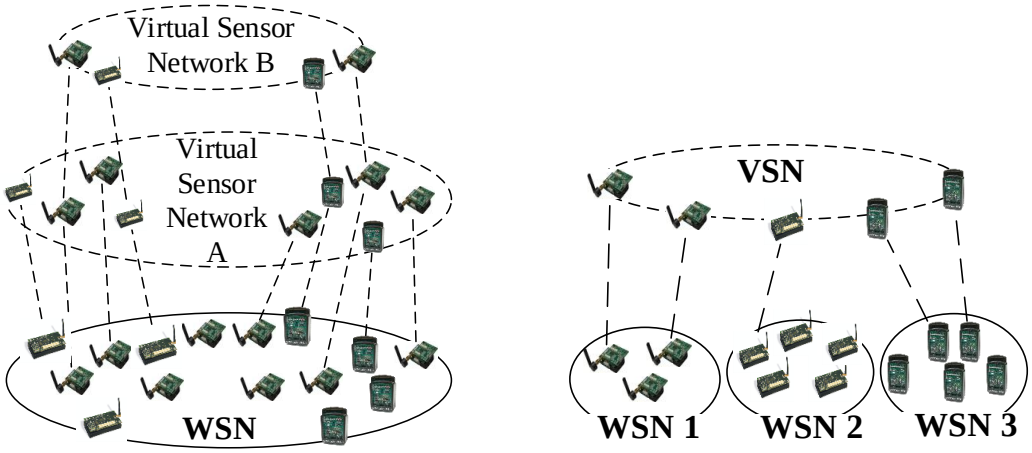
Figure 2.3: Examples of node-level virtualization solutions

2.5.2 Network-Level Virtualization

Network-level virtualization can be realized by using *i)* cluster-based approach or *ii)* by creating VSNs using overlays. The early work used the concept of clusters but managing clusters itself is quite challenging. The majority of work on cluster-based solutions in WSNs is focused on improving routing, energy efficiency and security. We need solutions that facilitate the creation of application-specific clusters that adapt to the dynamics of the network and of the monitored events. Recently overlay solutions are being used for network-level virtualization but it is still largely unexplored territory and applicable solutions are missing. It is expected that with the advent of IoT paradigm there will be more emphasis on proposing

solutions that enable localized coordination between sensors for applications and services. Fig. 2.4 shows the two types of node-level virtualization solutions.

The following characteristics are identified for node-level virtualization solutions: Network formation mechanism, Algorithm/Protocol used, and Evaluation method.



(a) Virtual network-based solutions

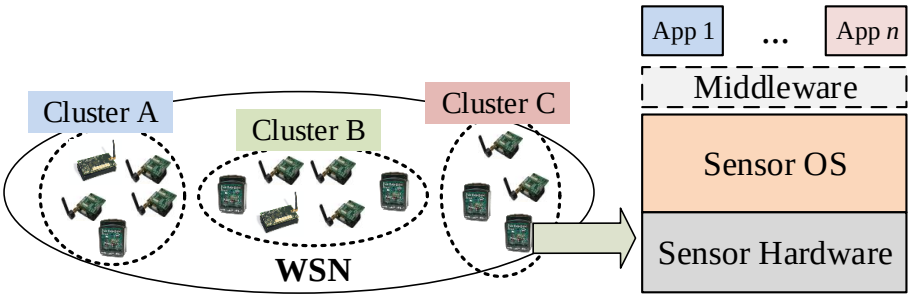
(b) Cluster-based solutions

Figure 2.4: Examples of Network-level virtualization solutions

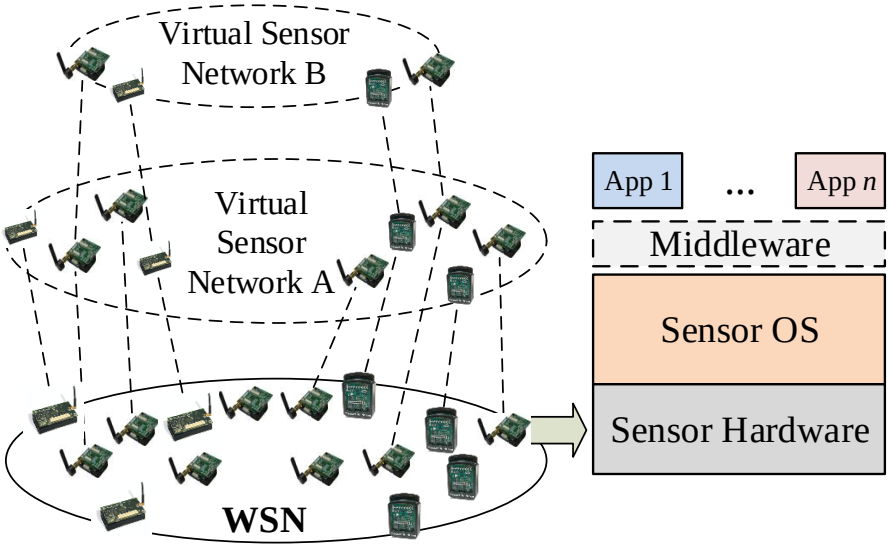
2.5.3 Hybrid Solutions

Hybrid solutions combine both node- and network-level virtualization mechanisms. Most recent research work has focused on providing hybrid solutions for WSN virtualization. A few recently-concluded research projects have addressed WSN virtualization, but their solutions are embryonic and multiple issues remain. For example, some solutions are platform dependent (SenShare [54]), others are theoretical and at conceptual level (VITRO [55]). However, as more capable sensor platforms and software solutions (e.g. mbed [59]) emerge, hybrid solutions are expected to get more attention. Fig. 2.5 shows the three hybrid virtualization solution types.

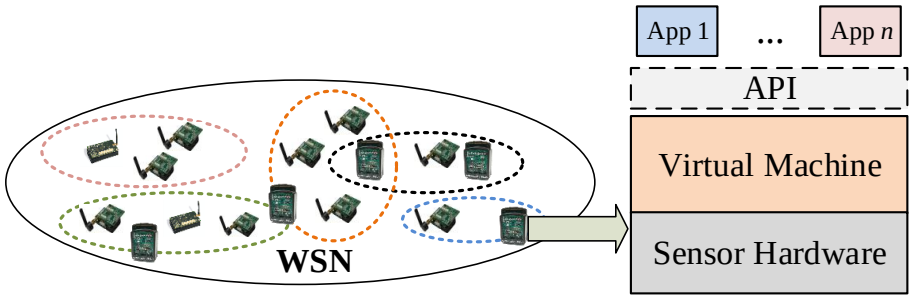
The following characteristics are identified for hybrid solutions: Programming mode, Programming Language, Separation between operating system and application tasks, Protocols supported, Support for real-time applications, Network formation mechanism, Algorithm/Protocol used, and Evaluation method.



(a) Middleware and cluster-based solutions



(b) Middleware and virtual network-based solutions



(c) Virtual machine and dynamic grouping-based solutions

Figure 2.5: Examples of Hybrid virtualization solutions

Table 2.1: Summary of the State-of-the-Art

		Requirements						
Solution	Type	Virtualization Type	Application Priority	Platform Independence	Resource Discovery	Applicable to Resource-constrained Nodes	Heterogeneity	Sensor Selection for Application Tasks
SenSmart [33]	OS-based	Node-level	No	Yes	No	Yes	Yes	No
RIOT [34]	OS-based	Node-level	Yes	Yes	Yes	Yes	Yes	No
PAVENET [35]	OS-based	Node-level	Yes	No	No	Yes	No	No
SenSpire [36]	OS-based	Node-level	Yes	Yes	No	Yes	Yes	No
Nano-CF [37]	VM-based	Node-level	No	Yes	No	Yes	Yes	No
UMADE [38]	VM-based	Node-level	No	No	No	Yes	No	Yes
Agilla [39]	VM-based	Node-level	No	No	Yes	Yes	Yes	No
Lite-OS [40]	OS-based	Node-level	Yes	No	No	Yes	Yes	No
Squawk VM [41]	VM-based	Node-level	Yes	Yes	No	No	No	No
VMSTAR [42]	VM-based	Node-level	Yes	No	No	Yes	No	No
MANTIS [43]	OS-based	Node-level	Yes	No	No	Yes	No	No
TinyOS [44]	OS-based	Node-level	No	Yes	No	Yes	Yes	No
Contiki [25]	OS-based	Node-level	Yes	Yes	No	Yes	Yes	No
Mate [45]	VM-based	Node-level	No	No	No	Yes	Yes	No
Khan et al. [46]	Overlay-based	Network-level	Yes	Yes	Offline Publication	Yes	No	No
MENO [47]	VN-based	Network-level	—	—	—	—	—	No
IoT-VN [48]	VN-based	Network-level	No	Yes	Yes	Yes	No	No
Tayanan et al. [49]	VN-based	Network-level	No	No	No	No	No	No
Jayasumana et al. [50]	VN-based	Network-level	No	Yes	No	Yes	No	No
Dilum et al. [51]	Cluster-based	Network-level	No	Yes	No	Yes	No	Yes
Han et al. [52]	Cluster-based	Network-level	No	Yes	No	Yes	No	Yes
Sensomax [53]	Middleware and Cluster	Hybrid	No	No	No	No	No	No
SenShare [54]	Middleware and Overlay	Hybrid	Yes	No	Yes	No	No	Yes
VITRO [55]	Hypervisor and VM	Hybrid	No	Yes	Yes	Yes	No	No
Majeed et al. [56]	Middleware and Cluster	Hybrid	Yes	No	No	Yes	No	Predetermined
Melete [57]	VM and Dynamic Grouping-based	Hybrid	No	No	Yes	Yes	No	Yes

2.6 WSN Virtualization Projects and Research Issues

In Paper I, several pertinent research projects are also reviewed. These included the early projects like CitySense [60] to the more recent European funded projects like iCore [61] and Butler [62]. The following characteristics are identified for each research project: Project aim, Project scope, Virtualization type, Network devices and Evaluation setup. Table 2.2 lists these projects and provides their summary based on these characteristics.

The following research issues are also identified in the paper along with discussions on their potential solutions:

- Advanced Node-level Virtualization
- Network-level Virtualization
- Discovery and Publication
- Service Composition
- Sensor Node Selection and Task Assignment
- Application Task Dissemination
- Reference Designs and Architectures
- New Protocols, Algorithms and Simulation Tools
- WSN Virtualization Business Model & Standardization
- Energy Efficient Solutions
- Access Control, Authentication, and Accounting
- WSN Virtualization Application Scenarios and Test-beds

For each research issue multiple possible solutions are discussed by considering the most recent research efforts in this domain.

2.7 Lessons Learned

The important lesson learned in this work is regarding the availability of the capable software and hardware solutions for WSN virtualization. The upcoming solutions like mbed operating system from ARM and RIOT operating system provide an insight into the future trends. They are expected to offer a rich set of features than the existing operating systems. Currently capable solutions like Contiki OS, TinyOS, Squawk VM, and Agilla middleware do exist that support the execution of multiple application tasks at the same time, however

runtime isolation between application tasks and the sensor operating systems is not available. Such isolation will allow virtual sensors could be deployed and removed just like virtual machines in traditional IaaS. On the hardware side there are many capable sensor kits such as Java SunSpots, Preon32 sensor kits from Virtenio GmbH [63] (Java-based and similar to SunSpots) and Phidgets kit [64]. These sensor kits can be used to research and prototyping purposes. As more advances are made in the hardware arena, more capable sensors kits will emerge in the future.

Another lesson is that there is a lack of architectural solutions to provision applications and services in virtualized WSNs. For example, VITRO project [55] does discuss a reference architecture without going into the details of functional entities, interactions between them, interfaces, and protocols. To this end, this thesis proposes three concrete architectural solutions, along with specification of functional entities, their interactions, interfaces and protocols backed up by the proof-of-concept prototypes.

Table 2.2: WSN Virtualization Related Projects

Project (Year)	Project Aim	Project Scope	Virtualization Level	Virtualization Type	Network Devices	Evaluation Setup
CitySense [60] (2008)	Provide city-wide test bed for distributed & networking research	Academic research	Network-level	Gateway-based virtualization	Embedded PCs with Linux acting as gateways	100+ PCs distributed over an urban area
FRESnel [65] (2010 - 2012)	Provide a federated WSN framework for multiple applications	Academic research	Node- and Network-level	Sensor node-based virtualization	iMote2 nodes using embedded Linux	35 iMote2 nodes distributed in an academic building
VITRO [55] (2010 - 2013)	Develop architectures, algorithms for VSNs.	Academic research + Industry	Node- and Network-level	Gateway-based virtualization	TelosB,IRIS, iSENSE, xbee, TmoteSkye, AdvanticSys kit	Simulations + 5 test bed setups by project partners
Smart Santander [66] (2010 - 2013)	Provide city-wide IoT experimentation platform for smart city applications	Academic research + Industry	Network-level	Gateway-based virtualization	Sensor nodes, IoT devices, RFID tags, GPRS devices	20,000 sensors deployed in four European cities
iCore [61] (2011 - 2014)	Provide cognitive framework consisting of virtual objects, composite virtual objects & business perspectives	Academic research + Industry	Abstract representation of sensors	Gateway-based virtualization	Sensors, ICT devices, everyday objects	Will utilize Smart Santander test bed
Butler [62] (2011 - 2014)	Provide secure, pervasive, energy-efficient & context-aware architecture	Academic research + Industry	Abstract representation of sensors	Gateway-based virtualization	Smart objects, mobile devices and smart servers	Several field-trials and proof-of-concepts
ViSE [67] (2008 - 2011)	Provide public access to a WSN test bed using the GENI framework	Academic research	Abstract representation of sensors	Gateway-based virtualization	High-end nodes running Linux and acting as gateway nodes	Three nodes deployed in a town near a forested area

2.8 Summary

This chapter provided a comprehensive state-of-the-art review as well as discussion regarding basics of WSN virtualization, which were not addressed before. A clear taxonomy of the existing works was presented and critically reviewed. Relevant research projects as well as future research issues were also discussed. WSN virtualization is very much relevant in the context of the IoT, in which small-scale devices, at an unprecedented scale, are expected to provide services to multiple applications concurrently, but we have yet to find a comprehensive solution that meets this challenge.

"— There's no good idea that cannot be improved on."

–Michael Eisner

Chapter 3

Wireless Sensor Networks Virtualization Architecture

Contents

3.1 Introduction

This chapter describes WSN virtualization architecture with details on proof-of-concept prototype and performance evaluation results. It also discusses the architectural enhancements to enable interactions between WSN IaaS and PaaS, along with early performance measurements. It is based on annex, B, C, D, and E and addresses the following research question:

How multiple and concurrent WSN applications can be provisioned over a deployed WSN? What is an efficient approach to build architecture to accomplish this? How WSN infrastructure can interact with a PaaS? What features must be supported by a WSN infrastructure to

allow PaaS to develop and deploy WSN applications and services?

The paper in annex D provides a general high-level view of WSN virtualization domain and possible avenues that could be explored. The paper in annex E presents an early version of the WSN virtualization architecture. These papers mainly contribute to the development of the WSN virtualization idea and its use for concurrent application and service provisioning unlike the existing approaches. The paper in annex B presents a complete architecture designs, implementation, and performance measurements whereas the paper in annex C builds on the previous work and extends the proposed architecture to allow interactions between WSN infrastructure and a PaaS.

The cornerstone of the proposed architecture is the concept of overlays, which is used to logically group sensor nodes together to execute applications tasks for concurrent applications and services. In this architecture, each logical group of sensor nodes belongs to a single application. Overlays have several advantages: they are distributed, lack central control and allow resource sharing [68]. Overlays are used to improve the transmission of data between end-hosts without requiring any change to the underlying infrastructure. P2P overlays can achieve significant performance improvements and better resource usage despite limited network capabilities and high failure recovery times. The same level of performance from the overlays can be achieved in WSNs if capable nodes are used to perform overlay related operations. These capable nodes are used in the proposed architecture to allow resource-constrained and early generations of sensors to be part of the overlays. In order to fulfil the fundamental requirements, a designated functional entity is used that provides the level of abstraction required to hide the details of the underlying WSN deployment from the applications. The proposed architecture is based on several architectural principles that make it easy to fulfil the identified requirements.

The type of applications considered in this architecture are traditional WSN applications that require notification messages from various sensor nodes detecting various physical phenomena such as fire, temperature, humidity, movement and so on.

3.2 Proposed Architecture

Fig. 3.1 shows our proposed multi-layer architecture. There are four layers (physical, virtual sensor, virtual sensor access and overlay), two paths (data and control), five interfaces (data (Di), proprietary Di (PDi), control (Ci), proprietary Ci (PCi) and gateway (Gi)) and a registration server.

At the physical layer we have independent WSNs that consist of two types of sensor nodes, i.e., resource constrained (type A) and capable (type B) sensors. Typical examples of type A and type B sensors are TelosB motes and Java SunSpots respectively. There are some specialized nodes in each WSN deployment, called GTO nodes. Their role is to help type A

sensors join the application overlays and provide heterogeneity. The examples of the nodes that can act as GTO nodes are gateways, base station nodes, sink nodes or a capable type B sensors. For example, in the motivating example in section 2.3.2, if the existing sensors are of type A, then either the existing gateway node or Type B sensors, deployed by the construction company, can help those sensors to become part of the construction company overlay. This might increase the complexity of the type B sensor nodes but it does allow flexibility and applies to new generation smart sensor nodes.

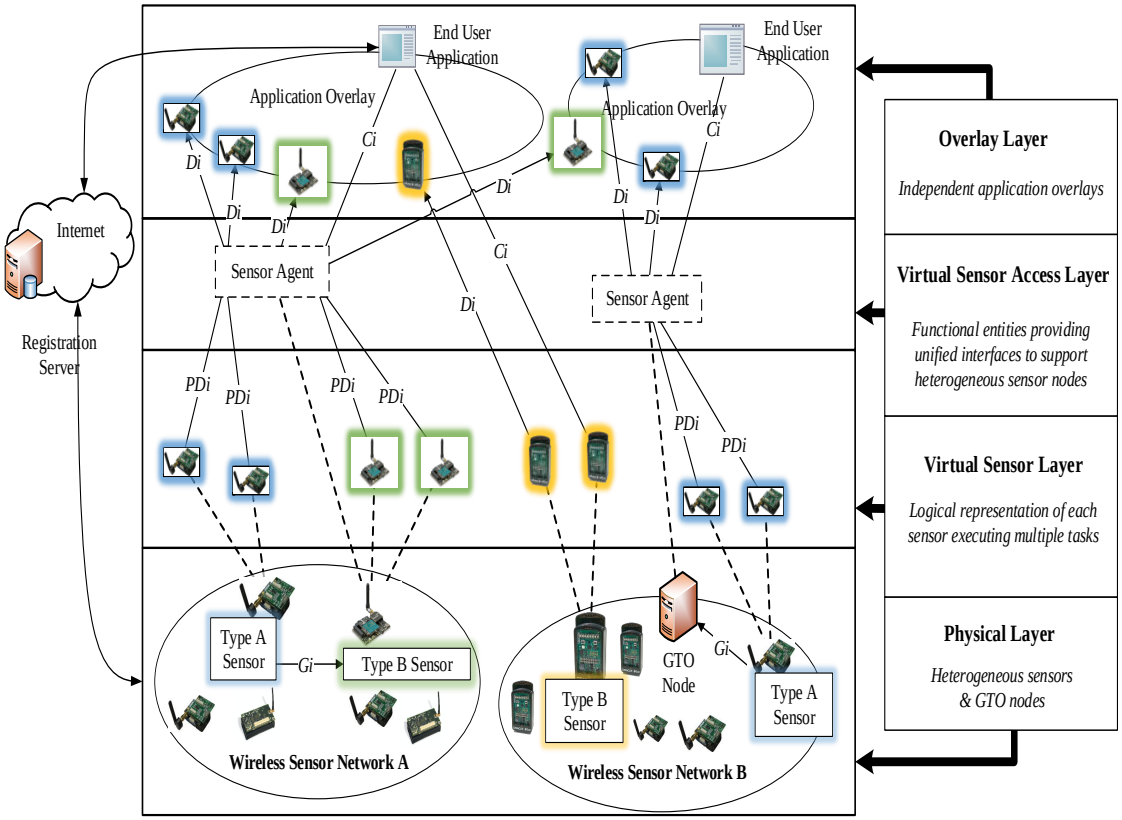


Figure 3.1: Multi-layer WSN Virtualization Architecture

The virtual sensor layer consists of the logical representation of each sensor executing multiple application tasks concurrently. Each logical representation is called a virtual sensor in our architecture, which is an abstraction of an application task run by a sensor. There is a one-to-one mapping between an application task and the end-user application, meaning that an application cannot have two virtual sensors (two different application tasks) on a sensor node. The realization of virtual sensors is platform dependent hence it is assumed that they can only communicate with other entities over a proprietary (platform dependent) interface. This assumption is particularly true for type A sensors. The number of virtual sensors that can be supported by a physical sensor node varies and depends on the capabilities of the

physical sensor node.

The virtual sensor access layer consists of a functional entity called Sensor Agent (SA), which ensure platform independence. This is achieved by providing standardized interfaces (Di and Ci) to interact with the end-user applications, and using platform-specific (proprietary) interfaces (PDi and PCi) to interact with the underlying physical sensor nodes. Example of a typical standard interface is a RESTful interface. SAs receive data from the virtual sensors and forward it to the end-user applications. SAs can be implemented either in capable (type B) sensors or in GTO nodes.

The overlay layer consists of independent application-specific overlays (two are shown in the Fig. 3.1, but there could be many more). Each application overlay is created by the end user application and consists of virtual sensors that run the overlay application tasks. An overlay protocol is used for message exchange inside an overlay. A Registration Server, which contains the details of the deployed sensor nodes, is used by end-user applications to find sensor nodes.

Overall, the proposed architecture can be used in many scenarios where sensors are shared by multiple applications. For example, consider a simple brush fire scenario where the city administration is interested in the early detection of brush fire eruption and in its evolution, using a WSN and a Fire Contour Algorithm. (FCA). Some houses in the area already have their own sensors to detect fire. To accelerate the deployment of its application and avoid redundancy, the city administration opts to deploy sensors in areas under its jurisdiction (i.e., streets and parks) and use the sensor nodes already deployed in private homes. The home owners get incentives like tax rebates for allowing the use of their sensors. In this scenario all of the privately owned sensors execute two application tasks one for the home owner and one for the city administration.

3.3 Proof-of-Concept Prototype

In order to measure the performance of the proposed architecture, we implemented a simple brush fire eruption scenario using Java SunSpot development kit.

In the scenario, the city administration is interested in the early detection of brush fire eruption and in its evolution, using a WSN and the FCA. We used a simple probabilistic FCA, considering that a distant house will send fire notifications less frequently than a nearby house because the fire is far from it. Some houses in the area already have their own sensors to detect fire. To accelerate the deployment of its application and to avoid redundancy, the city administration has opted to deploy sensors in areas under its jurisdiction (i.e. streets and parks) and to incorporate the sensor nodes already deployed in private homes. The home owners get incentives like tax rebates for allowing the use of their sensors by city administration. The home gateways acts as GTO nodes. All of the privately-owned sensors execute two application tasks – one for the home owner and one for the city administration.

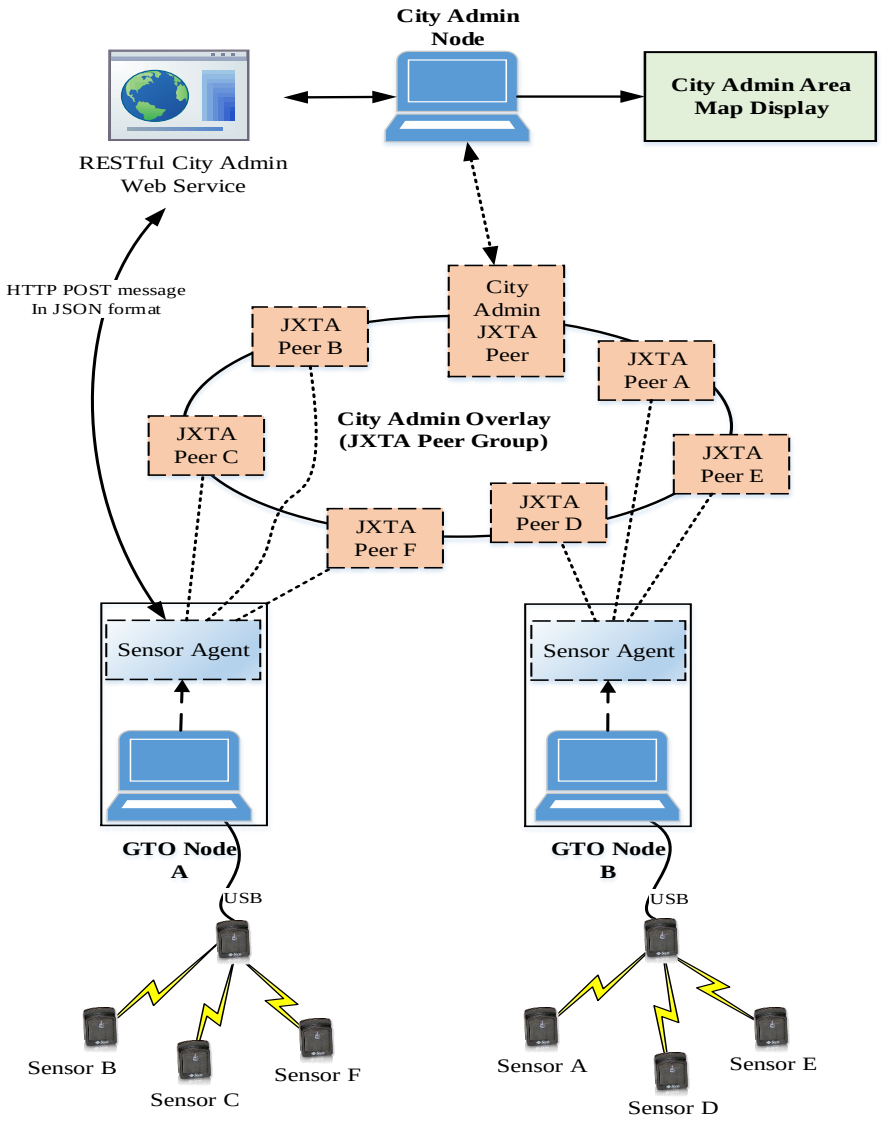


Figure 3.2: Instantiation of the architecture

We used six SunSpots (each executing three application tasks) and two base stations for performance measurements. The prototype setup is shown in Fig. 3.2.

3.4 Performance Measurements and Results

The performance of the prototype was assessed in terms of the following delays: *HTTP POST Delay* (HPD), *Overlay Creation Delay* (OCD), and *Fire Notification Delay* (FND).

HPD is the time difference between when the GTO node sends an HTTP POST request and when it receives the corresponding success code (201 created). HPD is calculated for each sensor. OCD is the time it takes to set up the city administration overlay from a non-

existent state to a ready state, when it advertises its fire contour service and is ready to accept join requests. We measured this delay inside the Java code to ensure that the OCD does not include the JVM start-up delay. FND is measured as the time it takes for the city admin node to multicast fire notification messages to JXTA peers and to receive their replies after they execute fire contour algorithm. For each experiment we restarted the JVM and cleared the previous JXTA configuration cache. All delays are measured in milliseconds and calculated at the sender side.

The HPD measurements are shown in Fig. 3.3 (for clarity, only 15 measurements are shown). It is observed that the delay for first POST message is much larger than that for the subsequent messages. This long delay is due to the three-way handshake of TCP connection that takes place during the first POST message, whereas for subsequent requests a persistent HTTP connection (a.k.a. HTTP keep-alive) reduces delay considerably.

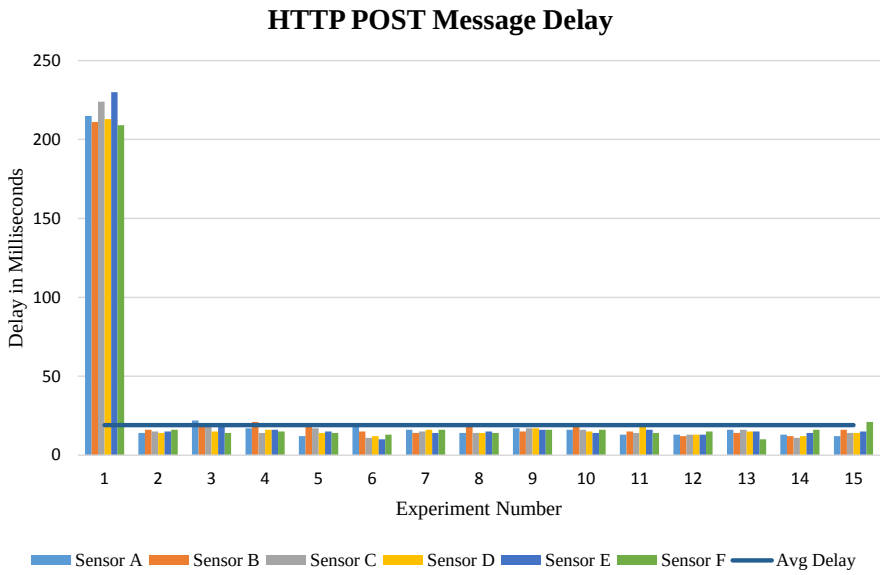


Figure 3.3: HTTP POST Message Delay

The average OCD of city admin overlay is 1983ms from 50 iterations, as shown in Fig. 3.4. This delay includes the JXTA core start-up, the creation of a fire contour service, related pipe advertisement, a JXTA multicast socket and the thread for accepting join requests from other JXTA peers.

The average FND of five sensors that executed a fire contour algorithm in response to the notification message sent by a city admin JXTA peer is 19.58ms. The FND of all sensors is shown in Fig. 3.5.

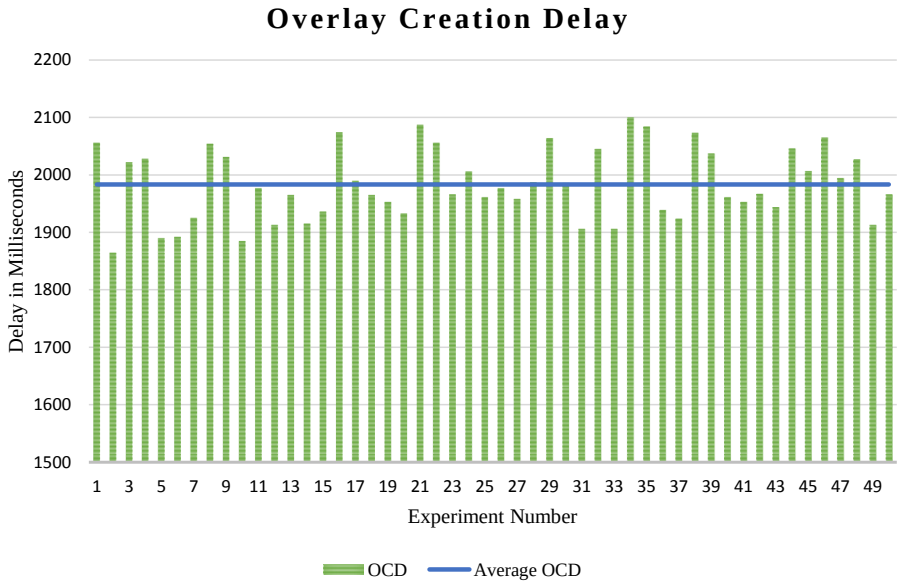


Figure 3.4: Overlay Creation Delay

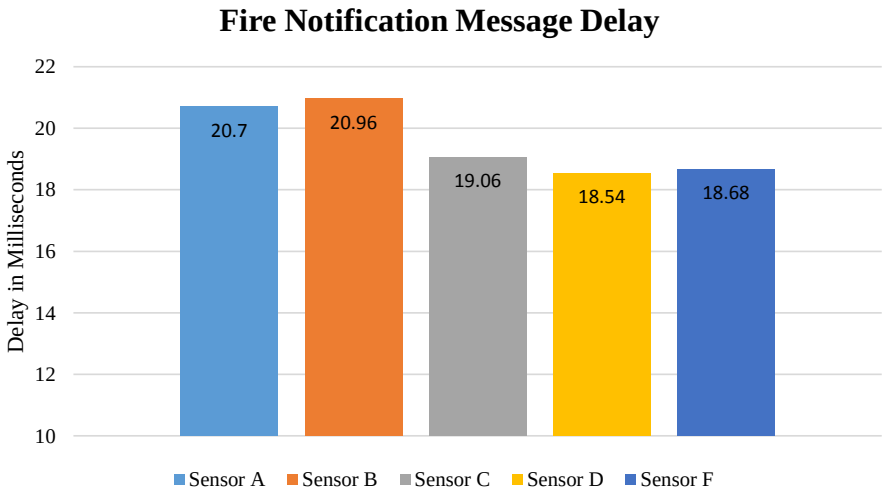


Figure 3.5: Fire Notification Message Delay

In order to determine the overhead of WSN virtualization, if we consider the scenario where there is no overlay network and the fire notification is sent as HTTP message (similar to HPD) which takes 18.96ms. Then the overhead introduced by the WSN virtualization for FND is approximately 3.27%. The implementation demonstrates that WSN virtualization is indeed feasible and does not incur much overhead. Node-level virtualization is achieved with Java SunSpots easily while network-level virtualization is achieved using JXTA, and once JXTA is operational, the delays are minimal. OCD is inevitable, but in the long-run, using JXTA is beneficial as it provides a robust and scalable solution.

3.5 Enabling Interactions between Virtualized Wireless Sensor Networks' IaaS and PaaS

So far the works in this chapter has made it possible to offer a deployed WSN as IaaS by using the concept of virtualization. The proposed WSN IaaS, can be used to provision traditional applications in an efficient way. However, a true cloud-based WSN IaaS requires interactions with a PaaS so that the latter can efficiently host and execute WSN applications and offer them as SaaS to the end-users.

Interactions between vWSNs and PaaS are largely an unexplored area and the architectures presented so far cannot provide the required level of interactions. For example, instantiation of VSs, starting them on demand and stopping them. In order to address this, the paper in annex E proposes a multi-layer architecture to offer competent virtualized WSN IaaS (vWSN IaaS), which are able to interact with PaaS to allow service providers to rapidly provision WSN-based applications and services. An important point discussed in the paper is that vWSN IaaS are fundamentally different from traditional IaaS. This is due to the inherit limitations of the WSNs and their nodes. In total seven differences (in terms of VM and VS) are discussed in the paper which are as follows.

- A VM allows for the sharing of resources (e.g., computing and storage) of the host machine, whereas a VS allows sharing of sensing capabilities (e.g., temperature, light, and humidity) by executing multiple application tasks. The key difference is that a VM aims at sharing the host machine resources, whereas a VS may use the computing and storage of the host sensor, but it aims at sharing the sensing capabilities of the host sensor. In Java SunSpots, for instance, application tasks access the on-board sensors to sense the physical phenomenon, and send the data accordingly.

- Multiple heterogeneous VMs (in terms of operating systems) can be simultaneously deployed on the same host. For instance, a host can support a Linux-based VM and/or a Windows-based VM at the same time. However, VSs are tightly coupled with their sensor OS/middleware. For example, a sensor cannot support Contiki-based VS and TinyOS-based VS at the same time.

- Multiple VMs can be deployed in an isolated manner. The creation, deployment, and migration of VMs does not affect the execution of existing VMs. On the other hand, the deployment of new VS may disturb the execution of existing VS(s). This is due to the limited resources and the tight coupling between the VS and the sensor OS/middleware. Similarly, migrating VS from one physical sensor to another is not a standard feature yet. To the best of our knowledge, Java SunSpots is the only platform that provides support for VS migration (as serialized Java Isolates). There is a work in which an agent-based system for Java SunSpots is developed for VS migration [69].

- VMs can be addressed by other entities that are similar to their host machines. Each VM can be assigned a public or private IP address and can be accessed accordingly. However, there is currently no standard mechanism for addressing a VS. Typically, a local ID is used and may vary depending on the platform. This necessitates some address mapping/translation mechanism to communicate with a VS. For instance, in Java SunSpots, each VS can be addressed by a MIDlet ID.

- For VM, there are no power/energy-related issues, whereas a VS inherits these issues from the host sensor nodes. This means that the creation, deployment, and operation of a VS are not only dependent on the capabilities/resources of the host sensor, but also on its available energy. The always-on or always-available concept is not applicable to WSN world.

- For VMs, there are already some open source and proprietary solutions (e.g., KVM and VMware). However, no such solutions exist for VSs.

- At the IaaS level, the role of a VM is to maximize the use of a host machines resources (e.g., computing and storage), while the role of a VS is to use the sensing capabilities of the host sensor in an efficient manner. Therefore, to achieve cost-efficiency, traditional IaaS may create several VMs on a limited number of host machines. However, achieving cost-efficiency in vWSN IaaS may not lead to the creation of several VSs on a few host sensor nodes since the creation of VSs is strongly correlated to the applications' desired coverage of a geographic area.

3.6 Extended Architecture

The proposed architecture is shown in Fig. 3.6. The bottom two layers (WSN Infrastructure and Virtual Sensors) are similar to the ones in the previous architecture and consist of heterogeneous sensors, GTO nodes and virtual sensors (both traditional as well as semantic). The functionality of these two layers and the roles of their entities are same as described in Section 3.2. Next layer is Virtual Sensor Manager, which contains two functional entities: The VS Manager and VS Communicator. VS Manager receives requests to instantiate, start,

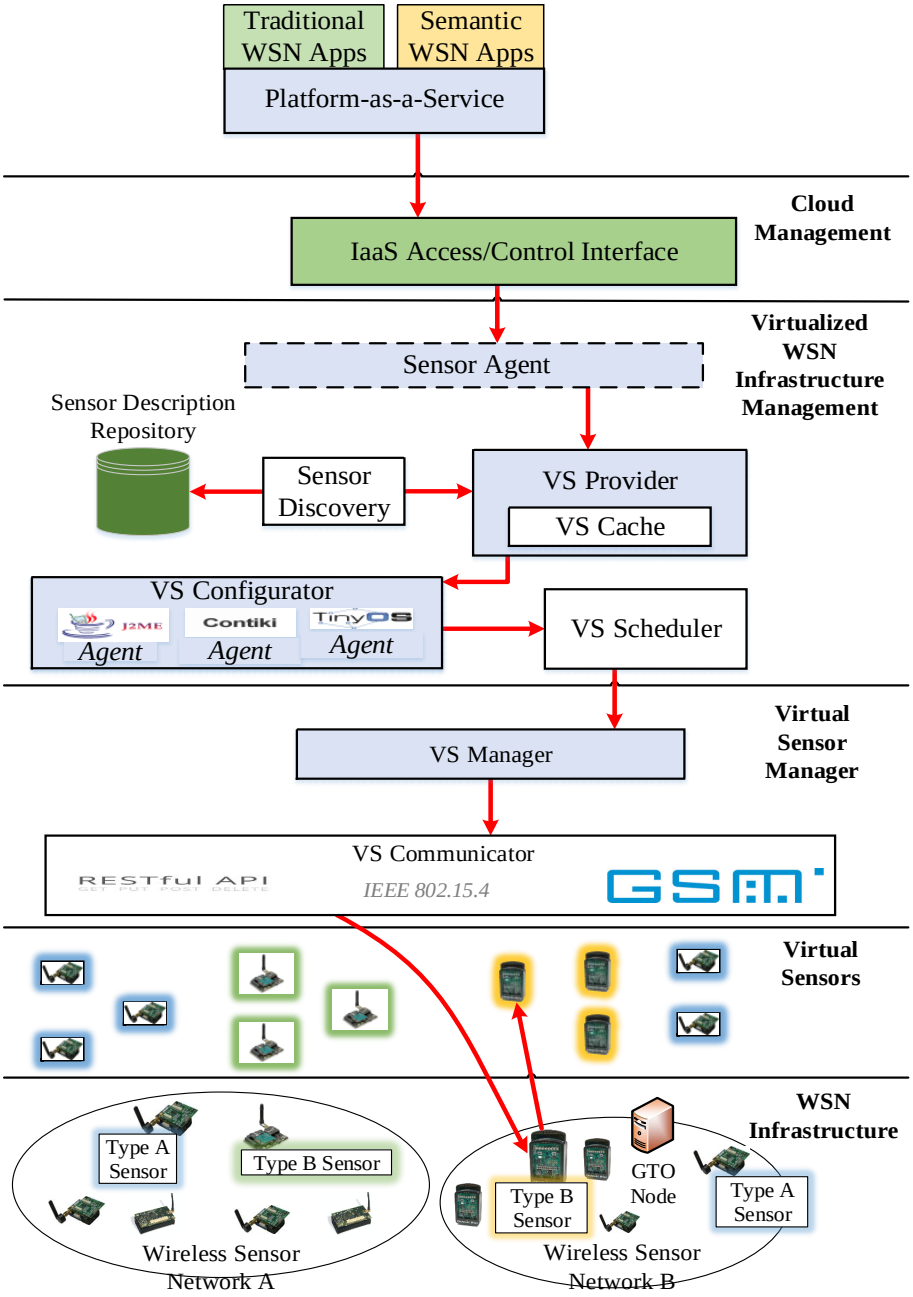


Figure 3.6: Proposed vWSN IaaS Architecture

stop, delete, and migrate VS. The VS Communicator supports platform-specific protocols to interact with different sensor platforms to promote platform heterogeneity, such as IEEE 802.15.4, Bluetooth, Cellular and RESTful.

The Virtualized WSN Infrastructure Management layer contains several new entities as well as SAs from the previous architecture. SAs interact with the PaaS on behalf of the VS

in order to provide platform independence. The additional entities in the architecture are: Sensor Description Repository (contains all relevant information about the deployed sensors), Sensor Discovery entity (interacts with the repository to search for the required sensors), VS Provider (receives VS creation requests from the WSN PaaS and makes decision about when to create, start, or stop a VS), VS Configurator (prepares task codes based on the requests received from the VS Provider), and VS Scheduler (creates, starts, stops, and disseminates task codes according to give schedule).

The task codes are generated from a skeleton code file that does nothing useful on its own but can read from a parameter list and run a desired task. An example is the skeleton code that reads a manifest file (i.e., used in Java SunSpot platform) to initialize parameters such as sensor type, sampling interval, desired unit, and an end-point address to send data output.

The final layer is the Cloud Management layer, which includes an entity called the IaaS Access/Control Interface. This interface exposes a RESTful API that allows multiple users (i.e. PaaS) to interact with the deployed vWSN IaaS through a set of REST-based operations.

3.7 Proof-of-Concept Prototype

In order to measure the performance of the proposed architecture, we used a simple scenario where an application developer developed a simple smart home application.

In the scenario, a smart home application is required to help home owners to configure the use of their appliances when environmental conditions change. For example, the A/C should start automatically when temperature exceeds a given threshold. Similarly, the deck lights should be turned-on automatically when natural light drops below a given threshold. The developer first discovers the light and temperature services to design, create and deploy the smart home application. The prototype setup is shown in Fig. 3.7.

For the prototype we used two Java SunSpot kits: two base station nodes and four SunSpots with on-board sensors. The vWSN IaaS layers were implemented as a Java standalone application. A simple PaaS was programmed as a standalone Java application. Eclipse IDE and JDK 1.7 were used for the application development. Two laptops, connected to a LAN, were used for the prototype. The first one had the vWSN IaaS, and the second one had the PaaS. The vWSN IaaS laptop was connected to the Java SunSpot base stations to communicate with the remote SunSpots Over-the-Air (OTA). The smart home application was developed as a simple Java application.

3.8 Performance Measurements and Results

The performance of the prototype was assessed in terms of the following metrics: VS Creation Delay (VSCD) and VS Start Time (VSST).

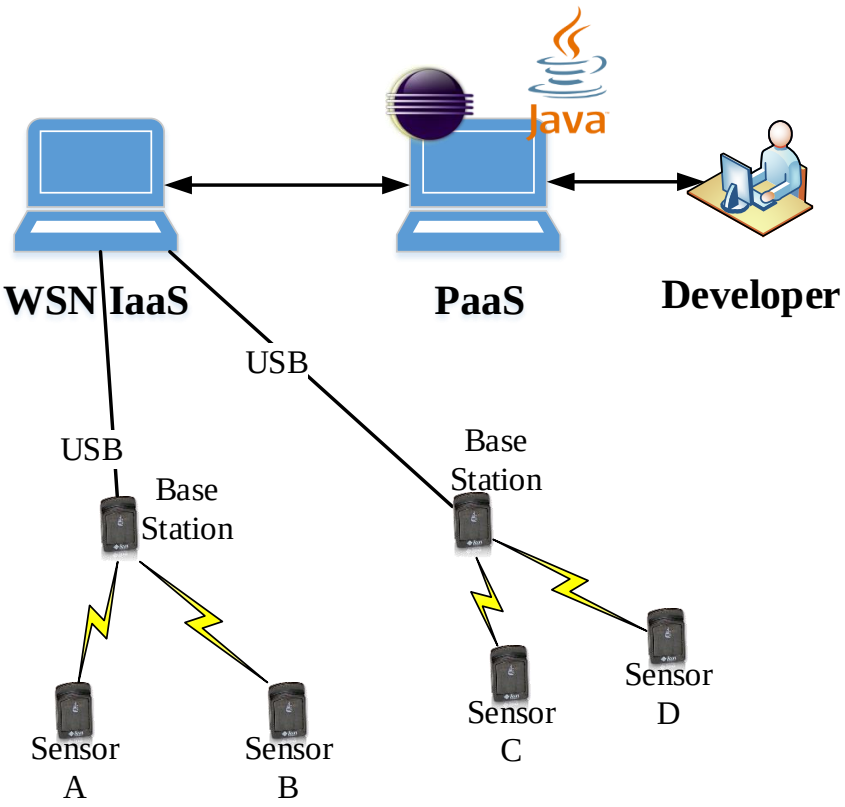


Figure 3.7: Prototype Setup

VSCD is the time spent between the moment the WSN infrastructure receives the VS creation request from the PaaS and the moment the VS is successfully created. We measured two types of VSCD. In the first type, the shared base station instance is created once and used repeatedly for VS creation, hence it only shows VS creation delay. In the second type, a shared base station is created every time a VS creation request is received from the PaaS, hence it shows VS creation delay plus the delay to create the shared base station instance.

VSST is the time spent when the WSN infrastructure receives the VS start request from the PaaS and when the corresponding VS is successfully started. All experiments were repeated 50 times with a confidence interval of 95%.

Fig. 3.8 shows the VSCD. The average value of the first type of VSCD after 50 iterations is 14.973 seconds while for the second type of VSCD, the average value increased by around 62%, to 24.282 seconds. The reason for this increase is that the shared base station instance takes time to probe for the available SunSpots. This delay is unavoidable and is not related our architecture. The higher values of both types of VSCD are also due to time taken by the Ant build tool to build, compile, and create the executable file for remote SunSpots.

Fig. 3.9 shows the VSST of the 50 experiments. On the average it takes 4.2 seconds to start the newly created VS after receiving the request from the PaaS. Again, this delay

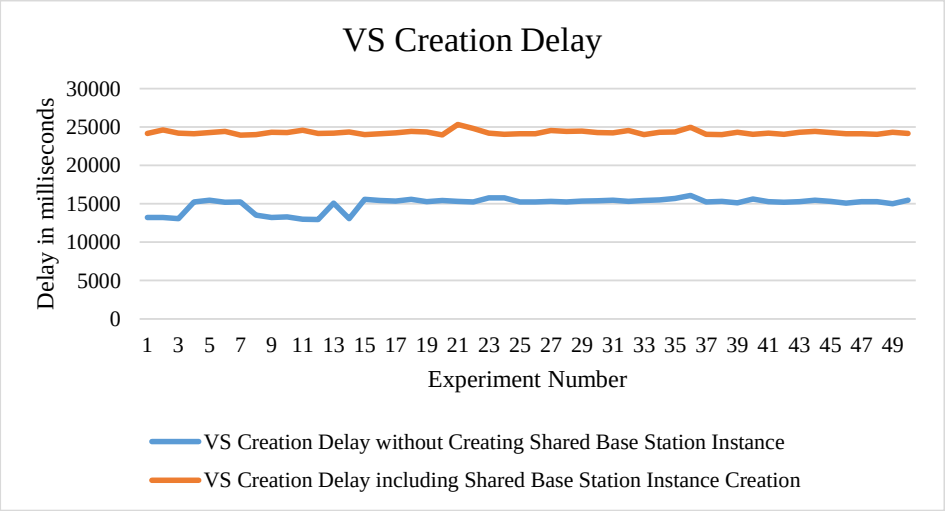


Figure 3.8: VS Creation Delay

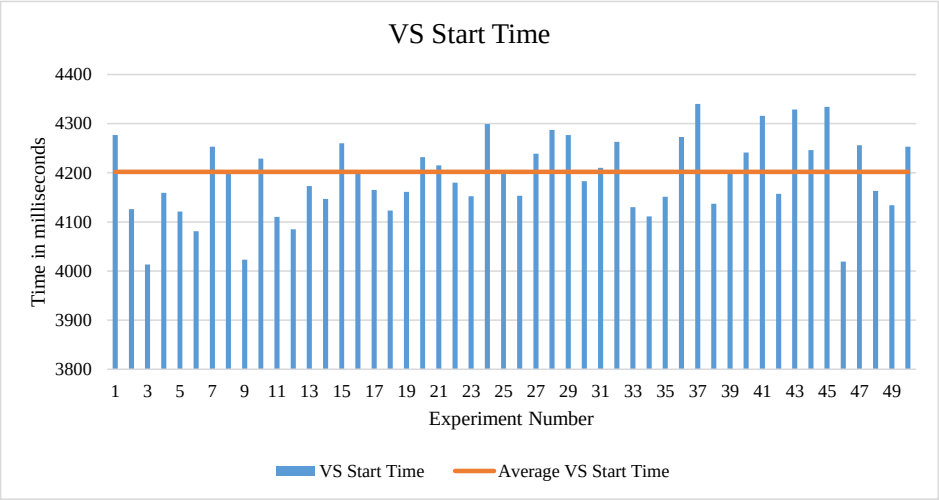


Figure 3.9: VS Start Time

included the remote SunSpot synchronization delay before the newly created VS was started.

3.9 Lessons Learned

Cloud computing offers elastic provisioning of resource and has revolutionized the way applications and services are deployed and consumed. It is clear that cloud computing can ease the WSN application and service provisioning over vWSN IaaS. For a business model,

WSN owners can simply focus on managing the resources while allowing PaaS developers to use their WSN deployments to provision new applications and services. However, it is learnt that current PaaS are not fully capable of using such deployments in an efficient manner. For example, currently there is no mechanism to discover and manage virtual sensors and their offered services at the PaaS level. Instead most solutions simply receive sensor data and use it without taking full advantage of a vWSN IaaS. Hence our lesson is that a capable vWSN IaaS needs an equally capable PaaS for developer to get the same level of application development and deployment environment as in traditional IaaS.

Another lesson is that the real potential of vWSN IaaS can be realized when sensors are able to make intelligent decisions based on the P2P communication with other sensors in the vicinity. Not only can it help in creating some interesting applications in mobile WSNs (vehicular ad hoc networks, mobile crowed sensing) but it can help in achieving efficient decentralized solutions. Currently IEEE 802.15 WPAN Task Group 8 is working on the specifications of Peer Aware Communications (PAC) in WPAN environments [70]. PAC offers number of features like peer information discovery without association, efficient discovery signalling rate of 100Kbps, data rate of 10Mbps, simultaneous multi-group communication, multi-hop relay and security. PAC is very much relevant to the WSNs since most sensors create a WPAN for communication. This could help in interesting application scenarios, for example, sensors can offload their tasks to a willing sensor in their vicinity when needed.

3.10 Summary

This chapter presented the solutions regarding concurrent application and service provisioning through WSN virtualization. First a novel multilayer architecture is presented that allows multiple applications and service to be provisioned over a deployed WSN. Additionally, the architecture is enhanced with new layers, entities and functionalities to allow interactions between WSN IaaS and PaaS to develop and deploy WSN applications and services. Both these contributions are backed up by the real-world prototypes along with performance measurements. This architecture can be used by traditional WSN applications and services.

"— It is not what you meant to say, but it is what your saying meant."

–Walter M. Miller Jr.

Chapter 4

Data Annotation Architecture for Semantic Applications in Virtualized Wireless Sensor Networks

Contents

4.1 Introduction

This chapter discusses the sensor data annotation architecture and presents implementation details and performance measurements. It is based on annex F and addresses the following research question:

How semantic web technologies can be used to efficiently provision WSN applications? In particular how semantic-based applications and service can receive annotated sensor data in real-time? Also how sensor data annotation can be performed in a distributed manner, in standardized way while making sure that future enhancements to the WSN infrastructure are also taken care of?

Semantic applications and services are gaining popularity and are particularly relevant in the WSN domain. These applications and services allow their users to get the high-level details of the events monitored by the sensors and infer additional knowledge to gain situational awareness. Traditional WSN applications do provide such kind of information to their users. Another benefit is that the semantic applications allow their users to use queries like *what is the current status of the fire?* and *what is the current location of the fire?* to get results like *initial fire* and *in a public library* respectively. Therefore, it is imperative to provision traditional as well as semantic applications in virtualized WSNs. However, semantic application require annotated data which is tricky to perform in such situations.

In order to tackle this issue we extend our previous virtualized WSN architecture to provision traditional as well as semantic WSN applications in virtualized WSNs through in-network sensor data annotation in a distributed manner. For this, the architecture uses overlays and has functional entities as super peers and peers to store the ontology and annotate raw sensor data respectively. Two main challenges addressed in this paper are *i)* how to annotate sensor data in real-time instead of annotating it at a central location and *ii)* how to keep virtualized WSN infrastructure independent of any particular application domain, because annotating data requires domain ontologies.

The first challenge is addressed by using in-network functional entities to annotate the sensor data in real-time before it leaves the network. In this new architecture, virtual sensors can be of two types: semantic and non-semantic. The raw data from semantic sensor is sent to the functional entity, responsible for the annotating it, and later it is sent to end-user application via SA (similar to the architecture in Chapter 3 – Section 3.2). The second challenge is addresses by creating a base ontology that reflects the deployed WSN infrastructure and not any application domain. For this standard SSN ontology is extended to create the base ontology. For the annotation process this base ontology is stored in the network in a distributed manner using the super peer concept. The functional entity (responsible for annotating sensor data) acts as peer and requests for the required base ontology from the super-peer to annotate raw sensor data. In this work, it is assumed that end-user applications and services will apply required application domain ontology (e.g. fire domain) since the proposed architecture is independent of any application domain.

4.2 Proposed Architecture

The proposed architecture, shown in Fig. 4.1, is based on our previous WSN virtualization architecture, presented in Chapter 3.

The architecture consists of four layers. The physical layer consists of sensor nodes that support node-level virtualization. Both resource-constrained (e.g. TelosB, called Type A) as well as capable (e.g. Java SunSpots, called Type B) sensor nodes are considered. Capable sensors as well as high-end machines (e.g. base stations and sink nodes) act as GTO nodes to

facilitate resource-constrained sensors to support node-level virtualization. The second layer is Virtual Sensor layer that abstracts as virtual sensors, the simultaneous tasks run by the physical sensors. In this work we assume that here can be two types of virtual sensors: those who run semantic application tasks (and require data annotation), called semantic virtual sensors and those who run non-semantic (traditional) application tasks, called virtual sensors.

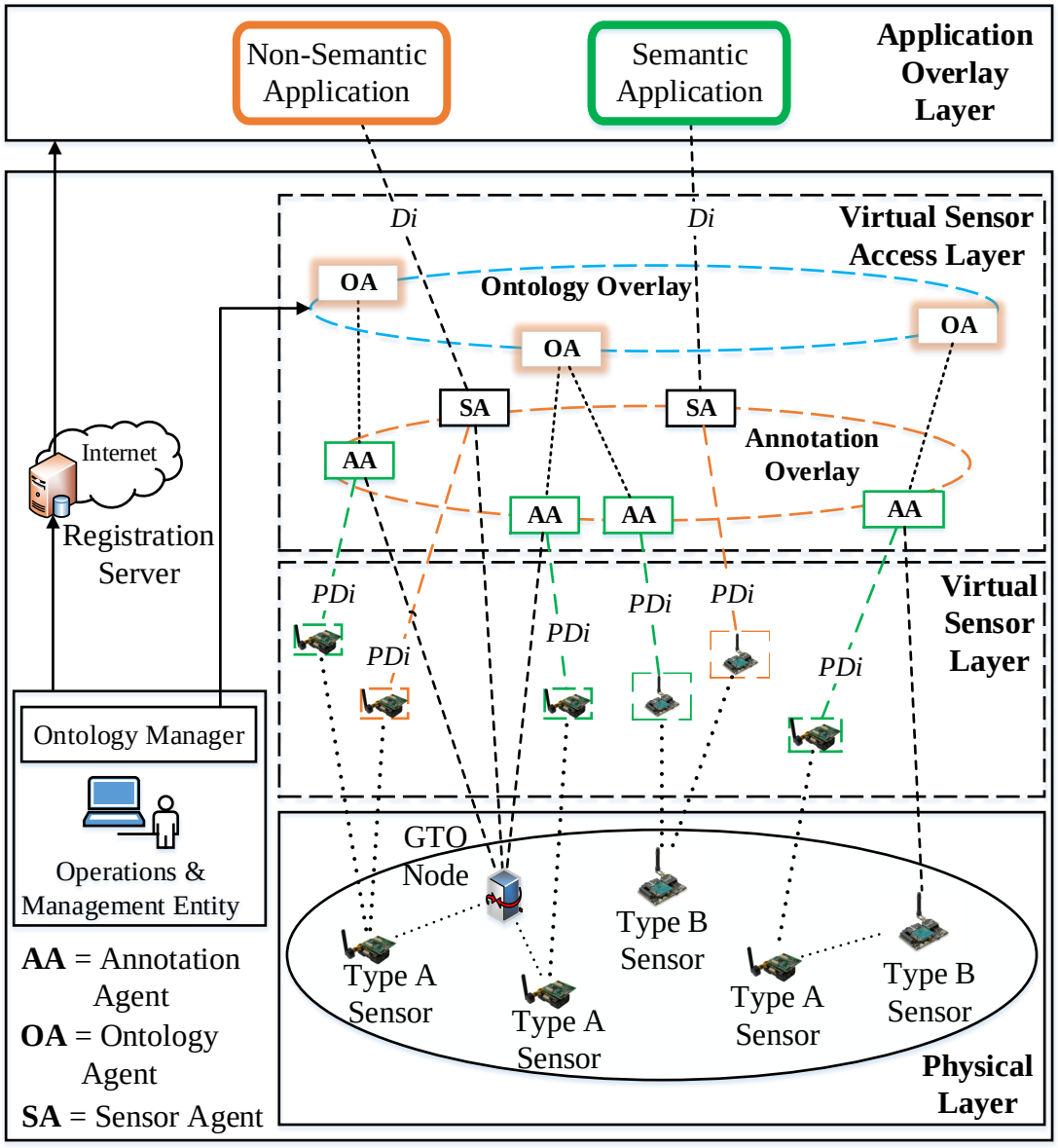


Figure 4.1: Multi-layer WSN Virtualization Architecture

The virtual sensor access layer has two new functional entities and two overlays. The functional entities are Annotation Agents (AAs) and Ontology Agents (OAs). We term an

agent as an entity that provides a given functionality, therefore several agents are used in our architecture. The Annotation overlay consists of AAs and SAs. The AAs receive raw sensor data from semantic virtual sensors for annotation. They use the base ontology for this purpose. Once the annotation is performed (RDF file generated), they communicate with SAs in the same overlay to send the annotated data to the semantic applications. SAs have the same role as mentioned in section 3.1, i.e., abstracting the underlying WSNs nodes and providing a standard access mechanism to end-user applications. The functionality of AAs can be implemented in capable sensors as well as GTO nodes.

The Ontology overlay consists of OAs, which are responsible for storing the base ontology in a distributed manner. The OAs act as super-peers and provide the requested ontology to the AAs. They do not deal with the sensor data. The Operations & Management (O&M) entity (typically infrastructure owner), is responsible for providing the base ontology. Since O&M entity is aware of the type of sensors deployed in the WSN, it can easily develop and disseminate the base ontology to the ontology overlay.

The base ontology that we developed reflects the deployed WSN infrastructure; sensors, their type, capabilities (e.g. available sensors, sensing range), properties (e.g. units of measurements, dimensions) and information like data format supported. Since a single sensor node can have multiple on-board sensors (e.g. TelosB has light, temperature and humidity sensors whereas Java SunSpot has light, temperature and accelerometer) the base ontology consists of multiple concepts each related to the available sensors. Fig. 4.2 shows the temperature part of the base ontology.

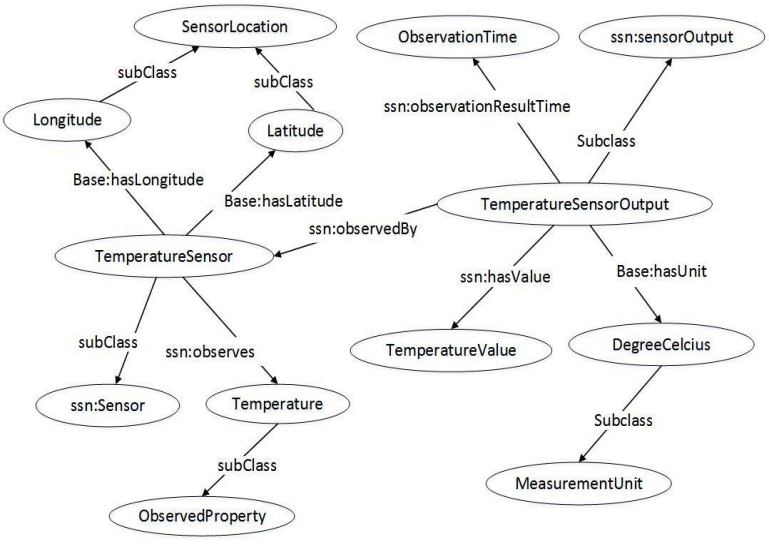


Figure 4.2: Temperature sensor part of the base ontology

Instead of storing one single base ontology file in OAs, we use a simple method to split the base ontology into multiple parts (each related to a single concept) and store these parts

in OAs. This way an AA does not need to keep light or humidity related concepts when it only requires temperature concept to annotate sensor data.

4.3 Proof-of-Concept Prototype

In order to measure the performance of the proposed architecture, we used a simple fire monitoring scenario where a semantic application tracks the fire in real-time.

In the scenario, the city administration and home owners deploy fire detecting sensors in public streets and in private homes, respectively. These sensors run multiple application tasks concurrently, using virtual sensors and semantic virtual sensors. The raw data from semantic virtual sensors is first annotated and then sent to fire monitoring semantic application via SA. The application applies the domain ontology and a set of rules using a reasoned to infer additional knowledge. If a fire event is detected then a notification is sent to the end-user. The end-user may query for additional information such as fire status and location.

We used two different sensor kits for the prototype, Java SunSpot and TelosB motes from AdvanticSys Kit. In total we used six SunSpots (two as base stations), four TelosB motes (one as border router) running Contiki OS. All sensors nodes executed multiple application tasks. The performance measurements are made using three different prototype setups (configuration A, B and C) which are shown in Fig. 4.3. Fig. 4.4 and Fig. 4.5 respectively.

In configuration A, TelosB motes are used which used a GTO node for annotation purposes. In configuration B, Java SunSpots are used who annotated their data by themselves. We achieved this by using μJena library [71]. In configuration C, raw sensor data was sent to the fire monitoring semantic application. The application was developed using Apache Jena Framework and deployed in cloud-based Google App Engine as SaaS.

In configuration A and B, fire monitoring semantic application received the annotated data while in configuration C, it received raw sensor data and performed annotation itself.

4.4 Performance Measurements and Results

The prototype's performance was assessed in terms of the following metrics: End-to-End Delay (E2ED), Ontology Download Time (ODT), Impact of the scalability of AAs, Expected Operation Time (EOT) of Java SunSpots, and the Impact of tasks on current draw from Java SunSpots battery.

E2ED is the time difference between when the semantic virtual sensors sent their raw data and when the corresponding success code (200 OK) is received from the fire monitoring semantic application. This delay includes the time taken by all intermediate steps. ODT is the time it takes an AA to request and to receive the required ontology from an OA. Impact of scalability of AAs was studied in terms of discovery of an OA and ODT. To find EOT of Java SunSpots, we executed both semantic and non-semantic tasks continuously until the

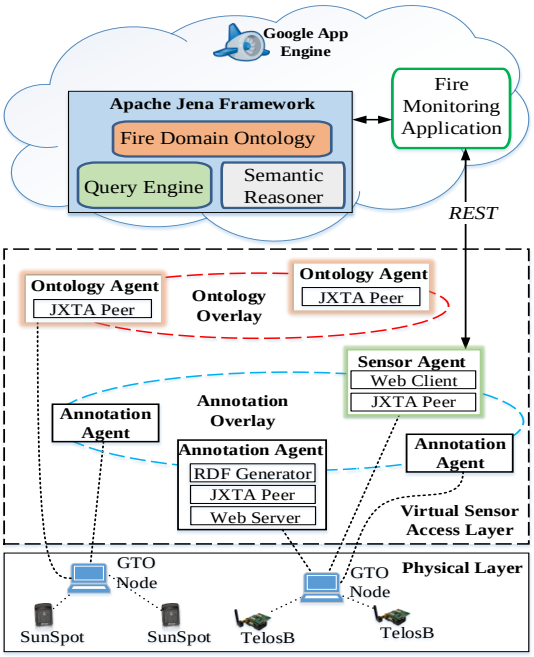


Figure 4.3: Prototype Configuration A

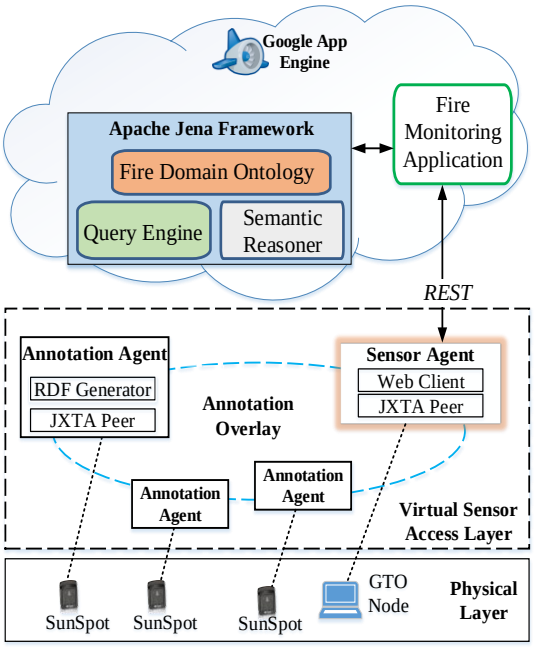


Figure 4.4: Prototype Configuration B

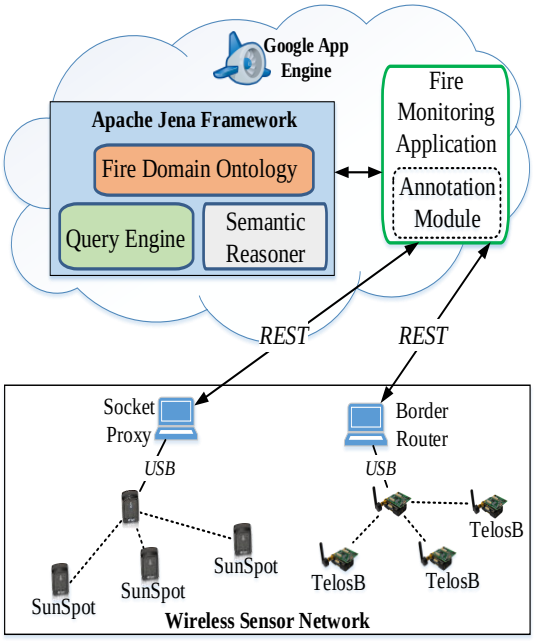


Figure 4.5: Prototype Configuration C

Spots died. For this purpose no sleep or power saving mechanism was used. Finally we determined the current draw from Java SunSpot battery while in shallow-sleep mode (no task, radio ON), executing semantic, and non-semantic tasks.

The average E2ED of configurations A, B and C is 3566ms, 4575ms, and 3187ms respectively as shown in Fig. 4.6.

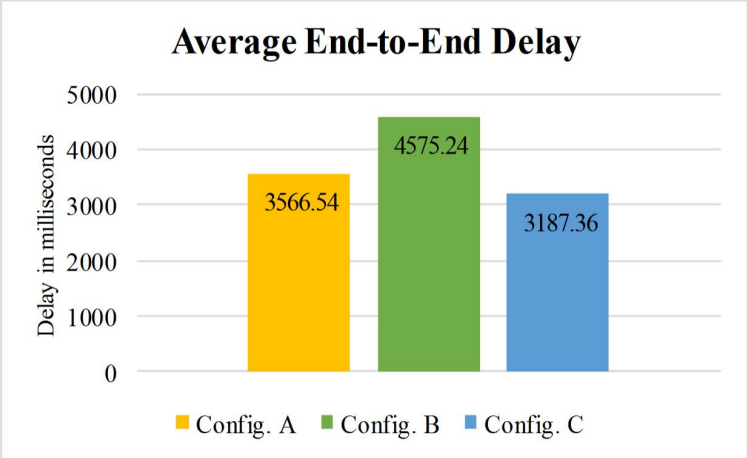


Figure 4.6: Average End-to-End Delay

The E2ED of three configurations from 50 experiments is shown in Fig. 4.7. The E2ED of configuration B is highest but considering the resource-constrained sensors it is acceptable.

Performing annotation outside the network has less delay as shown by configuration C but this comes at the expense of developing (or discovering) the base ontology beforehand.

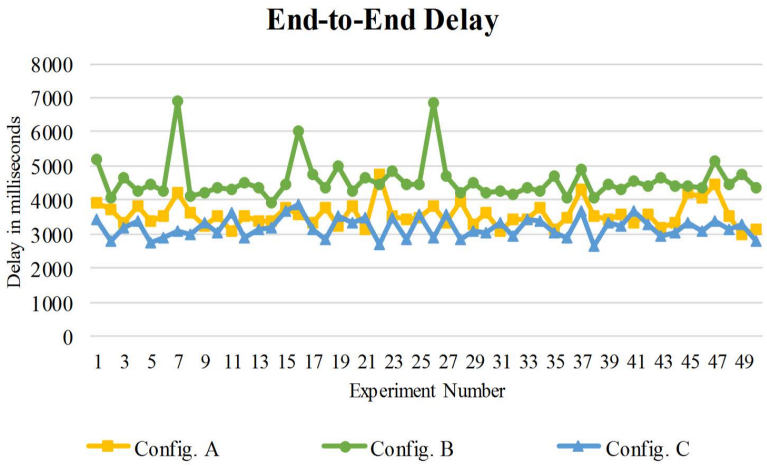


Figure 4.7: End-to-End Delay of All Configurations from 50 Experiments

The average ODT for configuration A is 94ms, as shown in Fig. 4.8. This is typical for JXTA under LAN environment.

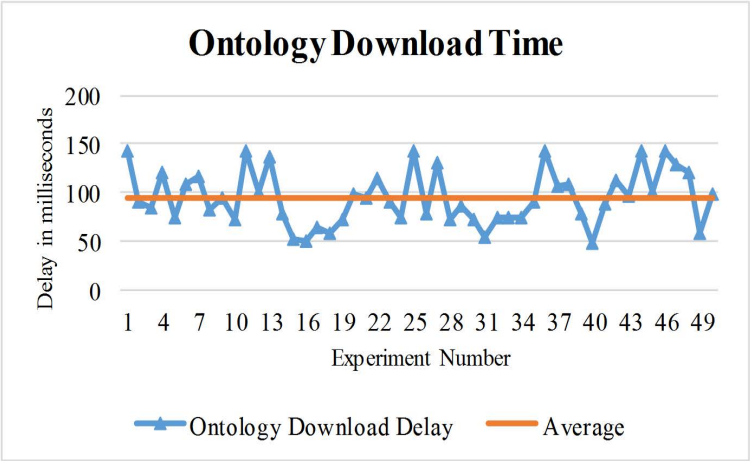


Figure 4.8: Ontology Download Time

Since JXTA was used for implementation, it had direct impact on the scalability part. We find that there is an increase in OA discovery time when number of AAs increase. This is shown in Fig. 4.9. However, the increase in AAs did not impact the ODT which was again around 100ms mainly because OA was already discovered.

Fig. 4.10 shows the EOT of the Java SunSpots while running a semantic and a non-semantic task, without using any sleep mechanism. SunSpots lasted 571 and 603 minutes

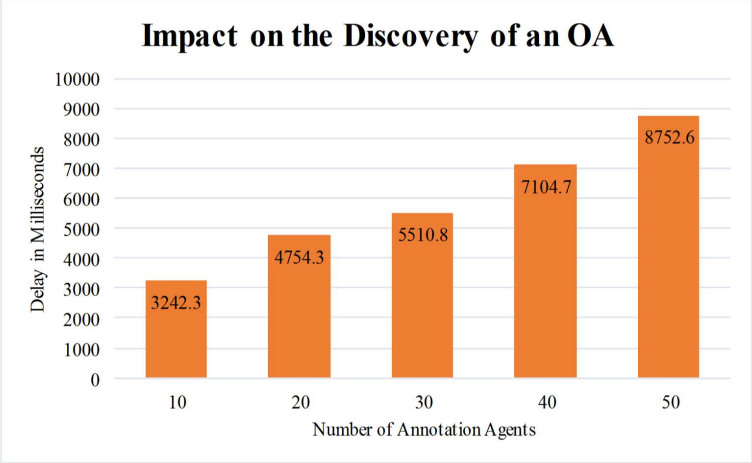


Figure 4.9: Impact on the Discovery of an OA

for semantic and non-semantic tasks respectively in a lab environment.

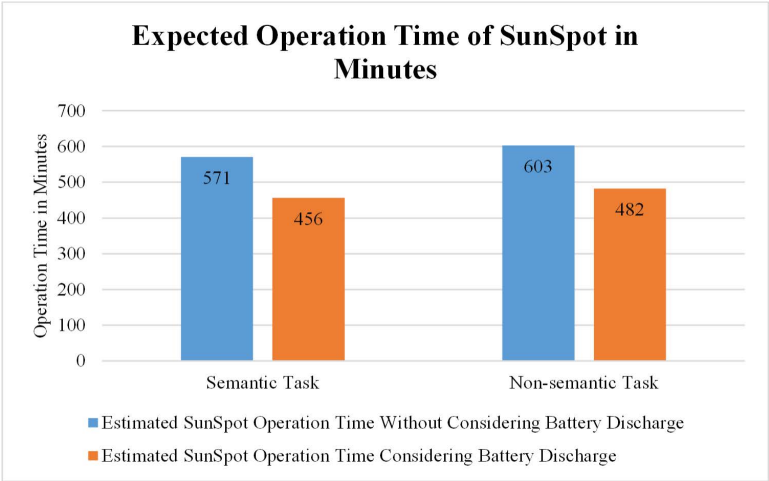


Figure 4.10: Expected Operation Time of Java SunSpots (always on)

We also calculated current draw by Java SunSpots, which are 38mA current (base value) during the shallow mode (no task, radio ON), 75.6mA for non-semantic task (98% increase from base value) and 79.8mA for semantic task (109% increase from base value).

4.5 Lessons Learned

An important lesson learned during this work is regarding semantic web technologies and the key roles they can play in vWSN IaaS. Semantic web can effectively eliminate the inconsistencies that may exist in the sensor data sent by various WSN deployments. For example

temperature data from three different WSN deployments can be received as key-value pair where key can be mentioned as *t, temp, temperature* respectively, and values being the observed value. In this situation application or service faces a difficult situation to interpret *t, temp and temperature* as same phenomena unless the developer has pre-empted such inconsistencies and dealt with them in his program. This issue becomes more prominent in vWSN IaaS when applications use data from different deployments. In our experience, incorporating semantic web technologies and standard ontologies with vWSN IaaS provides a mechanism for using sensor data in a consistent manner.

Another lesson is that by using an ontology that corresponds to the WSN, the deployed infrastructure could be made independent of any application domain. This way the annotated sensor data can be used by application from different application domains. However, in order to make use of the annotated data, applications will require their own required domain ontology to further annotate and use the sensor data for their purposes.

4.6 Conclusion

This chapter presented the work dealing with the issue of provisioning semantic applications and services over a virtualized WSN as demonstrated by the real-world prototype. In addition to this, the work proposed the concept of base ontology which is independent of any application domain and truly reflects the deployed WSN infrastructure. This opens up the possibility for WSN infrastructure owners to offer their network to a variety of users from different domains.

"— Logic will get you from A to B. Imagination will take you everywhere."

–Albert Einstein

Chapter 5

Provisioning of Semantic Applications over Virtualized Wireless Sensor Network IaaS

Contents

This chapter presents the ontology management tool and heuristic-based genetic algorithm for capable node selection, along with performance measurements. The chapter is based on annex G and H. It addresses the following research question:

How to enable a WSN IaaS owner to provide mechanism to support semantic-based application without making the deployed infrastructure application domain-specific? How to have an efficient and robust mechanism to annotate sensor data that is applicable to resource-constrained environments such as WSNs?

The main contribution of the architecture, presented in Chapter 4, is in-network sensor data annotation in a distributed manner in real-time. However, there are two issues that are not addressed by it, *i)* Allow WSN infrastructure owners to create and manage the ontology, and *ii)* Facilitate the efficient sensor data annotation in a distributed manner. We proposed using base ontology for annotation purposes since it is independent of any application domain. The base ontology reflects the deployed sensors in the WSN and their capabilities. It also provides a standard way to share the sensor data to other users promoting interoperability. By extending SSN ontology, our proposed base ontology conforms to the existing standards, therefore the data annotated using the base ontology can be readily used by other applications and services according to their requirements.

With this background it is clear that there is a need to allow WSN infrastructure owners to create and manage base ontology in an efficient manner. An ontology development and management application is developed for this purpose. It is a web-based GUI application that uses MySQL Database, Apache Tomcat and Protégé 3.8 API. Using an easy and step-by-step approach, the base ontology can be created even by a novice user without knowing technical details. It is possible to add/modify concepts such as temperature, humidity and so on. Similarly when new sensors are deployed in the WSN, the base ontology can be modified with their details.

Once the ontology is developed it needs to be stored in the deployed WSN in a distributed way to be used when required. In this work we propose to store base ontology in multiple nodes by dividing it into multiple parts such that each part contains a single concept. In order to select a set of capable nodes to store these parts a heuristic-based genetic algorithm is proposed. The algorithm tries to achieve multiple objectives and provides an optimal set of nodes for the storage of base ontology concepts. It is executed by a central node that has the status information of all the sensors.

5.1 Proposed Architecture

The proposed architecture, shown in Fig. 5.1, uses our previous WSN virtualization architecture, presented in Chapter 4.

There is a new node, at physical layer, called WSN IaaS Manager that has a global view of the deployed WSN infrastructure. WSN IaaS Manager is responsible to select capable nodes for storing the base ontology and then disseminate the ontology files to the selected nodes. The virtual sensor layer remains same as in the previous architecture. There is a new functional entity in the ontology overlay in the virtual sensor access layer called Ontology Manager (OM). The role of OM is to hold the base ontology and provide it to the OAs when requested. OM is distributed over many capable nodes in the network (i.e. GTO nodes and Type B sensors). The rest of the functional entities and the overlays in virtual sensor access layer remain same as in previous architecture.

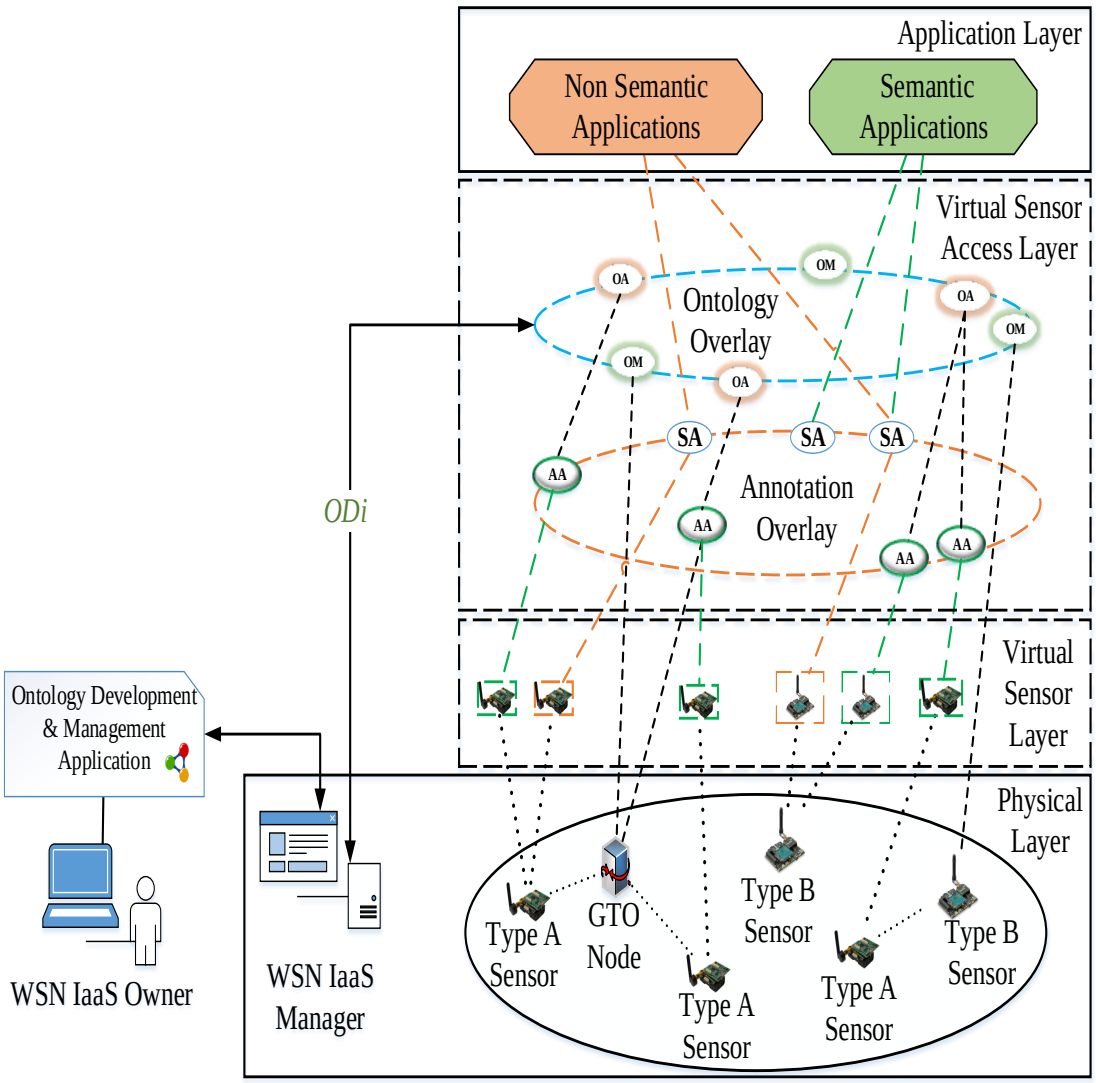


Figure 5.1: Proposed Architecture

In order to develop and manage the base ontology, the WSN infrastructure owner uses the Ontology Development and Management Application. Once the base ontology is developed, it is provided to the WSN IaaS Manager who then sends it to the network. The only entity, that can receive the base ontology from the WSN IaaS Manager is OM, hence it is important that OM have the most update version of the base ontology at all times.

5.2 Proof-of-Concept Prototype

In order to measure the performance of the proposed architecture, we use the same fire monitoring scenario where a semantic application tracks the fire in real-time. A simple prototype is used where the WSN IaaS Owner develops the base ontology using the Ontology Development and Management Application and disseminates it to the network. Fig. 5.2 shows the prototype setup.

We used three laptops connected to a private LAN. One laptop is used to host the Ontology Development and Management Application and to act as WSN IaaS Manager. The second laptop acts as OM while third laptop acted as OA and implements the partial functionality of AA, as mentioned before. The respective functionalities of these entities were implemented as Java applications in all three laptops. One Java SunSpot kits was used consisting of 1 base station node and 2 SunSpots with on-board sensors. Each SunSpot executed two application tasks at the same time. The annotation functionality of AA was implemented in the SunSpots as mentioned before.

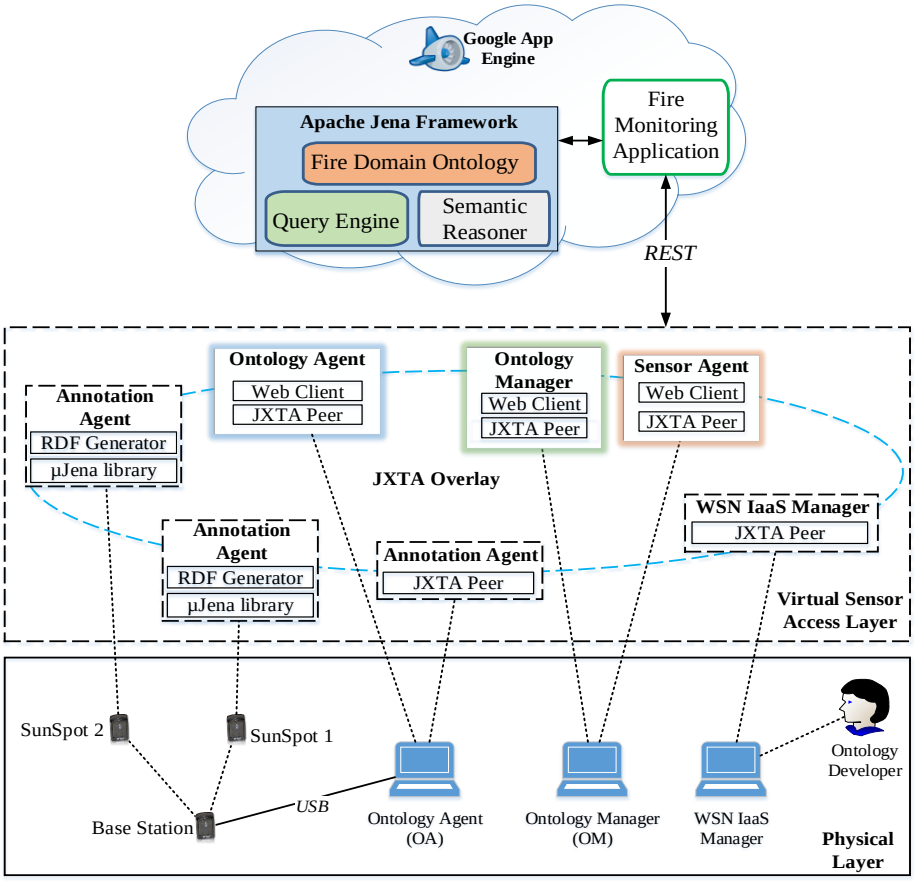


Figure 5.2: Prototype Setup

5.3 Performance Measurements and Results

The performance of the prototype was assessed in terms of the following metrics: Overlay Creation Delay (OCD), Ontology Dissemination Time (ODisT) and Ontology Download Time (ODT). OCD is the time to create JXTA overlay from a non-existent state to a ready state, when it is ready to accept join requests. We measured this delay inside the Java code to ensure that the OCD does not include the JVM start-up delay. ODisT is the combination of the following delays, i) Delay from ontology application to WSN IaaS Manager, ii) Delay from WSN IaaS Manager to OM, and iii) Delay from OM to OAs. For ODT we measured the delay when an AA sends request to OA for a missing part of base ontology. All these experiments were repeated 50 times with 95% confidence interval.

The average OCD is found to be 1906ms from 50 experiments as shown in Fig. 5.3. It is important to remember that the OCD pretty much depends on the configurations of the machines that act as JXTA peers and is unavoidable. However, it is experienced only during the overlay initiation phase so does not necessarily make much impact during the sensor data annotation process.

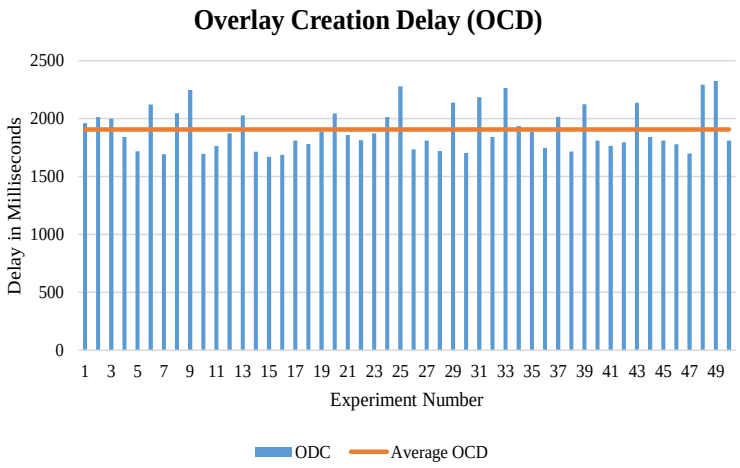


Figure 5.3: Overlay Creation Delay

The average ODisT from 50 experiments is around 109ms as shown in Fig. 5.4. ODisT includes *i)* Delay from ontology application to WSN IaaS Manager, *ii)* Delay from WSN IaaS Manager to OM, and *iii)* Delay from OM to OAs. During our experiments we found that the delay from ontology application to WSN IaaS Manager is negligible since both entities were on the same laptop. Therefore this delay is not included in the given results. The delay from WSN IaaS Manager is shown in vertical lines in Fig. 5.4. The average delay is 56ms. The delay from OM to OA is shown as dots in Fig. 5.4 and on the average it is about 54ms.

The average ODT from 50 experiments is 137ms, as shown in Fig. 5.5. The reason for higher ODT as compared to the delay from WSN IaaS Manager to OM and delay from

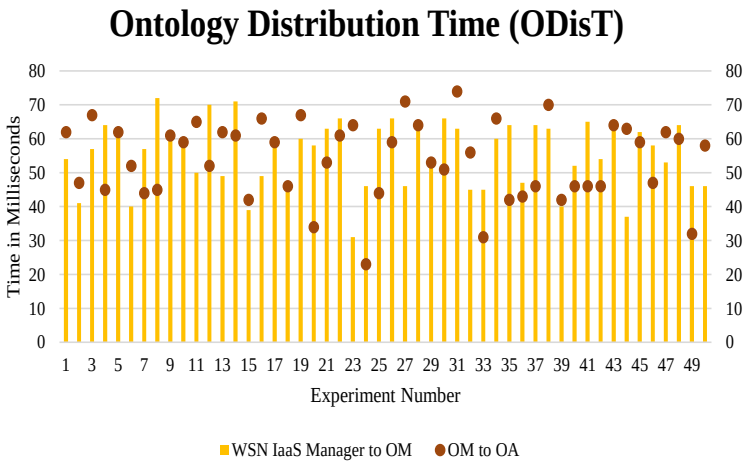


Figure 5.4: Ontology Distribution Time

OM to OA is because ODT includes the request and reply delay as AA first sent a request for the ontology file and later received it where as for the other delays there was no request message, WSN IaaS Manager and OM simply sent the ontology file to the destination without receiving any request message.

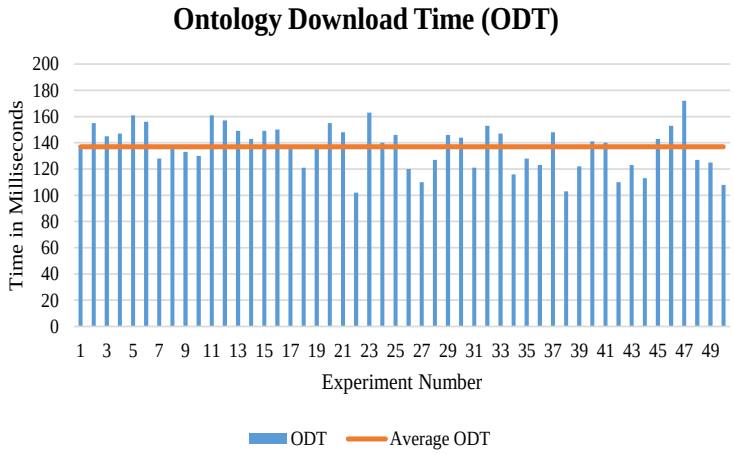


Figure 5.5: Ontology Download Time

5.4 Multi-objective Genetic Algorithm for Capable Node Selection

In our proposed architecture, ontology concepts are stored in multiple capable sensor nodes in a distributed way. In order to have a dynamic and lightweight solution, we propose to select a set of capable nodes to store the ontology concepts (i.e. act as OAs). In this work,

we assume that all AAs are able to act as OAs (in terms of capabilities). However, due to limited resources of AAs (energy and memory), it is necessary to select an optimal number of them that will act as OAs. This selection needs to ensure that only nodes that fulfil the energy and storage requirements, at that particular time, are selected. Once a set of nodes is identified, ontology concepts are provided to them. Thus in this work we have energy-related requirements and memory related requirements. The optimal selection of OAs can be modelled as a multi-objective optimization problem where the objectives include maximizing residual energy and maximizing residual storage. The solution to this problem provides the OAs and their respective AAs. We use the GA to solve the optimization problem. The GA is executed by a central node, i.e. WSN IaaS Manager shown in Fig. 5.1.

5.4.1 Problem Representation

We propose a two-level encoding scheme to encode a chromosome for GA as shown in Fig. 5.6. The level-1 encoding is used to represent the OAs; whereas the level-2 encoding is used to represent the members (i.e. AAs) of each OA. The two-level encoding is based on binary encoding. The level-1 encoding consists of n binary bits, where n is number of active sensors in the network. In the chromosome each gene represents a sensor. A value of 0 in level-1 encoding means that the sensor is not selected to act as OA whereas a value of 1 means it is selected to act as OA.

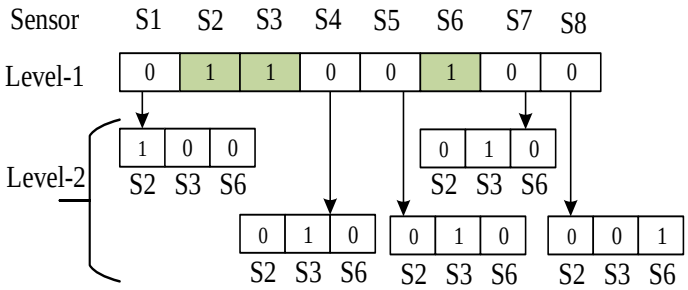


Figure 5.6: Two-level encoding example

The level-2 encoding is described as follows. Each gene in level-1 encoding that has 0 bit has a level-2 binary string. Each such string consists of m bits where m is equal to the number of 1 bits in level-1 encoding. To encode this, a position is randomly chosen between 1 and m and is filled with 1. Rest m-1 positions are filled with 0s.

Note that, numeric encoding could have been used for level-2 encoding. However, binary encoding has an advantage of being flexible in performing mutation operation as described later.

5.4.2 GA Operators

Crossover and mutation are the two genetic operations performed on the chromosomes in a population. In a crossover operation, genes from different chromosomes (parents) are recombined to produce new chromosomes (children). The crossover operation ensures that after many generations, best features of the parents are carried to the next generation.

In the genetic algorithm, 2-point crossover is used. In particular, the 2-point crossover operation is applied to level-1 encoding. When a gene is extracted from a parent chromosome, the corresponding level-2 encoding is also extracted. Since the proposed encoding results in variable length level-2 string for each chromosome, the crossover operation must preserve the number of 1 bits in each chromosome. Thus, the basic 2-point crossover operation cannot be applied directly on the chromosomes. One way of resolving this issue is to consider fixed length level-2 strings which also requires pre-specifying the number 1 bits for the level-1 encoding for each chromosome in the initial population. The other way is to adopt a variant of 2 point crossover operation [13] which preserves the number of 1 bits in each chromosome that will be produced after crossover. The basic 2-point crossover operation on fixed length level-2 strings is illustrated in Fig 5.7.

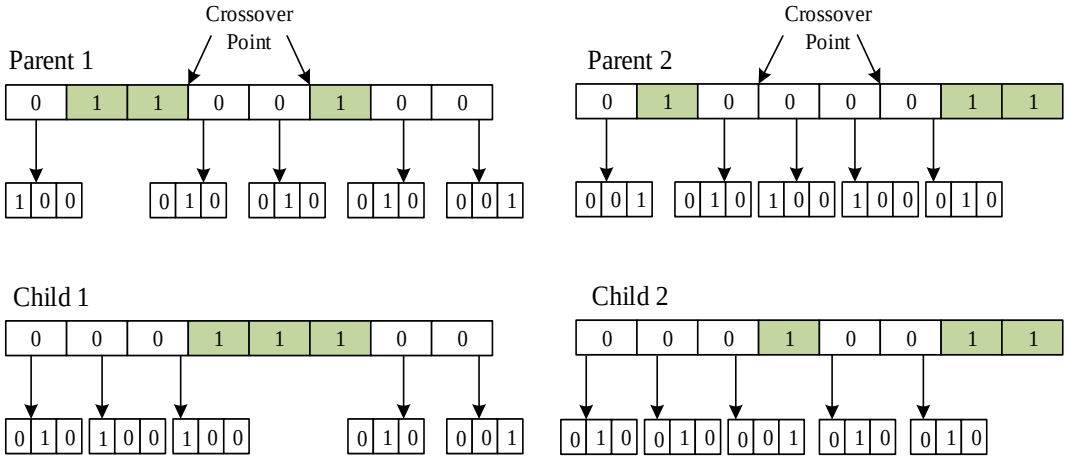


Figure 5.7: Example 2-point crossover operation (fixed length level-2 string)

After the crossover operation, mutation operation is applied to each child. During this operation two genes, selected at random, are interchanged in a chromosome. In our case, the mutation operator is applied in 2 steps to the level-2 strings. In the first step, two genes that have 0 bits in level-1 encoding are selected at random. Then, their level-2 strings are interchanged.

In the second step, a gene that has 0 bit in level-1 encoding is selected at random. Then, in its level-2 binary string, a random position corresponding to a 0 bit is selected. Then,

this 0 bit is interchanged with the 1 bit. These steps are illustrated in Fig. 5. Note that, with numeric encoding for level-2 strings, only step-1 can be performed. Thus, using binary encoding for level-2 strings results in flexibility in mutation operation.

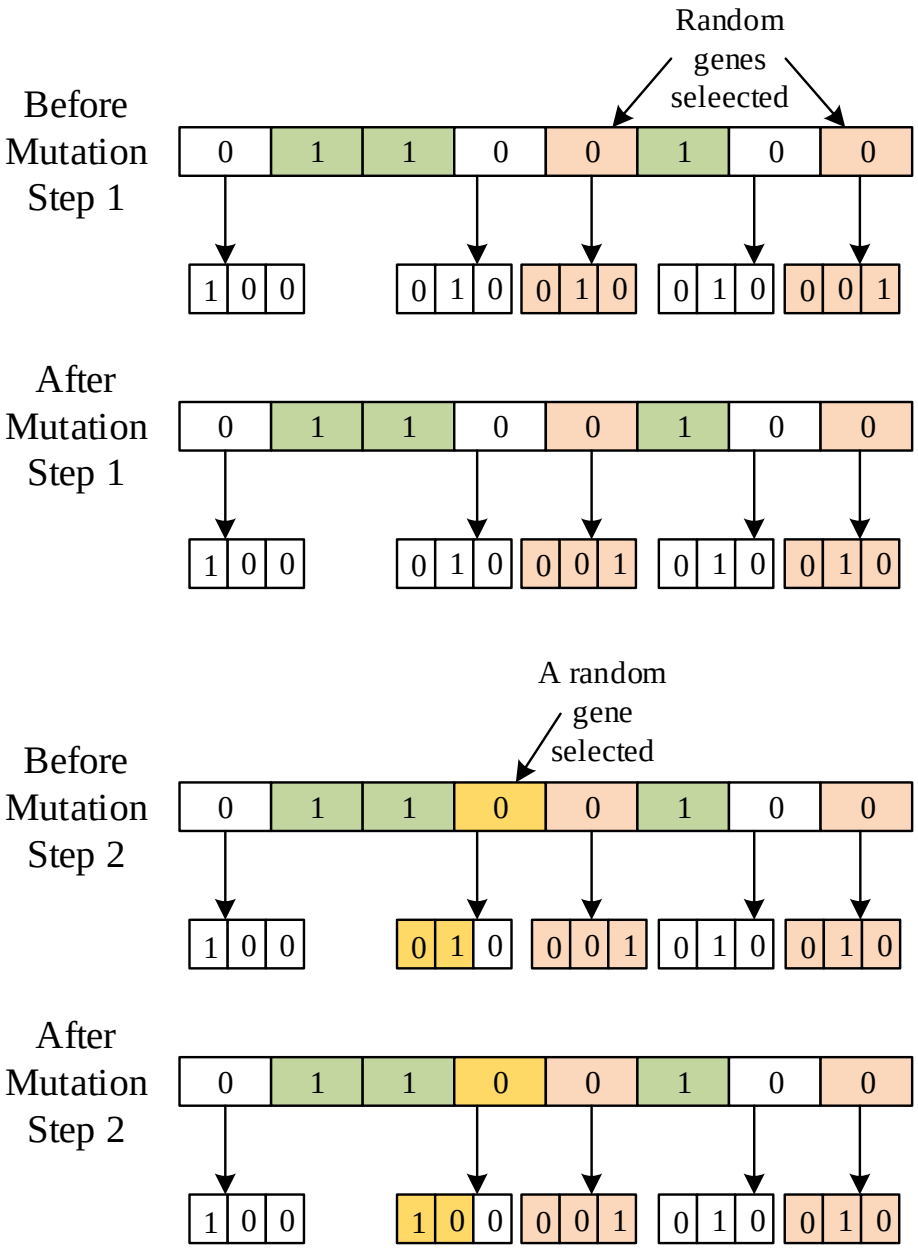


Figure 5.8: Example mutation operation

5.4.3 Objective Functions

In order to select most promising and fittest individual to produce new generation in each iteration, we use two objectives function that allows us to compare the individuals. Our main objectives are:

f1: Maximizing the residual energy of sensors

f2: Maximizing the residual storage of sensors

These objectives ensure that the total residual energy and total residual memory of sensor nodes are maximized. For computing f1, we adopted the following energy model.

$$E_{Tx} = \begin{cases} l * E_{elect} + l * \varepsilon_{fs} * d^2 & (d < d_0) \\ l * E_{elect} + l * \varepsilon_{mp} * d^4 & (d \geq d_0) \end{cases}$$

Figure 5.9: Example mutation operation

Where, E_{Tx} denotes energy consumed in sending l bit of data to a node at distance *d*. E_{elect} is the amount of energy consumption per bit to run the transmitter and receiver circuitry. The details of other parameters can be found in [72].

The objective f1 is expressed as:

$$f1 = \sum_{j=1}^{m} \left[E_{max\,,j} - \left[\left(\sum_{i=1}^{n_o(j)} E_r \right) + E_j(o) \right] \right]$$

Where,

m = *number of sensors selected as OAs,*
$E_{max,j}$ = *Current residual of energy of sensor j,*
$no(j)$ = *Number of AAs for which sensor j act as OAs,*
E_r = *Energy spent in communication between two sensors,*
$E_j(O)$ = *Energy spent by sensor j in receiving ontology file*

$E_{max,j}$ is known from the status received from sensors before executing GA. E_r and $E_j(O)$ are computed using the above energy model.

The objective f2 is expressed as:

$$f2 = \sum_{j=1}^{m} \left[M_{max\,,j} - \left[\left(\sum_{i=1}^{NA} M_j \right) + M_j(o) \right] \right]$$

Where,

m = *number of sensors selected as OAs,*
$M_{max,j}$ = *Current value of storage of sensor j,*
N_A = *Total number of application tasks running in sensor j,*
M_j = *Storage needed for applications in sensor j,*
$M_j(O)$ = *Storage needed for ontology file in sensor j*

Since the OA selection problem is a multi-objective optimization problem, there is not one optimal solutions rather a set of solutions called pareto-optimal solutions. However, finding the best or a good trade-off solution is often difficult as it requires a proper analysis of the pareto-front. Therefore, multi-objective optimization problems are often solved using scalarization or weight sum approach which transforms multi-objective optimization to single-objective optimization.

Using this approach, the new objective function is expressed as:

$$Z = W1 * f1 + W2 * f2$$
$$W_1 + W_2 = 1,\ 0 \leq W_1, W_2 \leq 1$$

Where, W1 and W2 are weights and indicate the relative importance of the objective functions. These weights can be adjusted based on the need.

The final proposed capable node selection algorithm is shown next.

Algorithm 1: Capable Node Selection Algorithm

Result: Set of capable nodes that can act as OAs

Input: Population size α
total number of sensors n
crossover probability β
mutation probability γ
number of iterations σ

Output: solution X

`/* Initialization */`
`/* Level 1 Encoding */`
Generate α random solutions of size n
for $i \leftarrow 0$ **to** n **do**
    individual$[i] = randomInt[0, 1]$;
    `/* if individual[`i`] == 1 then it is an OA */`
    `/* else it is not an OA */`
end
`/* Level 2 Encoding */`
for *each individual k not Selected As OA in Level 1* **do**
    Create a binary string of length m bits ;
    `/* where` m `is the number of 1s at level-1 */`
    Fill one random bit of the string with 1 ;
    Fill rest of the bits with 0 ;
end
repeat
    $crossOver(with\ \beta\ probability)$;
    $mutation(with\ \gamma\ probability)$;
    $fitnessEvalution()$;
    $replaceWithNewGeneration()$;
    $iterations++$;
until $iterations <= \sigma$;
return *Solution*

5.5 Simulation Results

The performance of the proposed algorithm is assessed using simulations, the details are as follows.

5.5.1 Simulation Setup

The proposed algorithm is implemented using Apache Commons Math library [73]. We execute the algorithm for large number of sensors to get a near-optimal solution containing the list of sensors that can act as OM in the deployed WSN IaaS.

Since our work targets capable and advanced sensor platforms, we considered Java SunSpots for our simulation. Java SunSpots have built-in rechargeable Li-ion battery with a total energy of about 9590 joules. The current residual energy of the sensors is fixed at random from 50% to 100% of this value. A uniform value of 50j is assumed for communication between OAs and AAs. Similarly a uniform value of 80j is assumed for OAs to receive ontology file from the central OM node. In our previous work, described in chapter 4, we developed the base ontology with multiple concepts (e.g. temperature, light, carbon, humidity). The maximum ontology file size we had was around 8Kb for a single concept. In this work, we assume a storage space of 10Kb for storing a single ontology file in an OA. In addition to this we consider the scenario where OAs may be executing applications tasks themselves. Here we assume that each OA executes three application tasks. It is important to mention that rev 8 of Java SunSpot provides about 7200Kb of storage space of application tasks. Hence considering 10Kb for ontology storage makes sense. Each experiment was repeated 10 times and the results presented here show the average values of these experiments.

5.5.2 Results

The total fitness value of the optimal set of OAs is shown in Fig. 5.9 It is important to mention that the values are higher because fitness value of all OAs is combined. We observe that the higher population size (i.e. more sensors) do not necessarily lead to maximum fitness value. However, as iterations passed, the fitness became larger. For this result we kept crossover rate at 0.8. For each population size the best solution was found in the last few iterations (e.g. 47^{th}, 48^{th} and 50^{th}).

The next results shows the number of OAs and AAs obtained by using different crossover rates. Fig. 8, 9 and 10 show the average number of OAs and AA obtained using crossover probability of 0.2, 0.5 and 0.8 respectively. It is interesting to note that the low crossover rate leads to less number of OAs. This means that there will be more AAs associated with one OA. As we increased the crossover probability, number of OAs increased as well irrespective of the population size. Another interesting observation is that there is not major increase in number of OAs when population size increases. In fact the number of OAs remain pretty

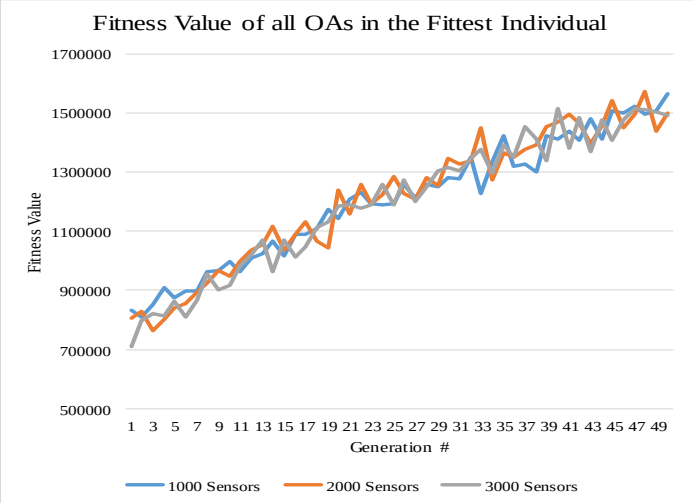


Figure 5.10: Fitness value of all OAs in the fittest individual

much consistent w.r.t population size.

We also performed simulation by varying the mutation rate but found that it did not have any major impact.

Overall these results provide us interesting insights and motivate us to further improve the algorithm and solve other research problems in this area.

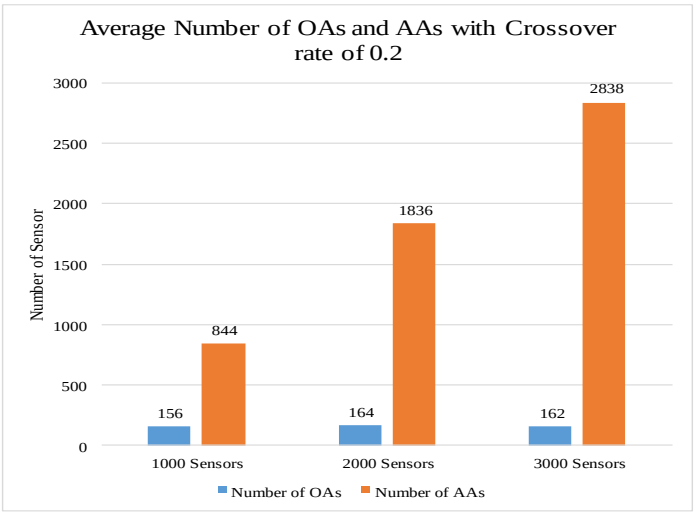


Figure 5.11: Average number of OAs and AAs obtained with crossover rate of 0.2

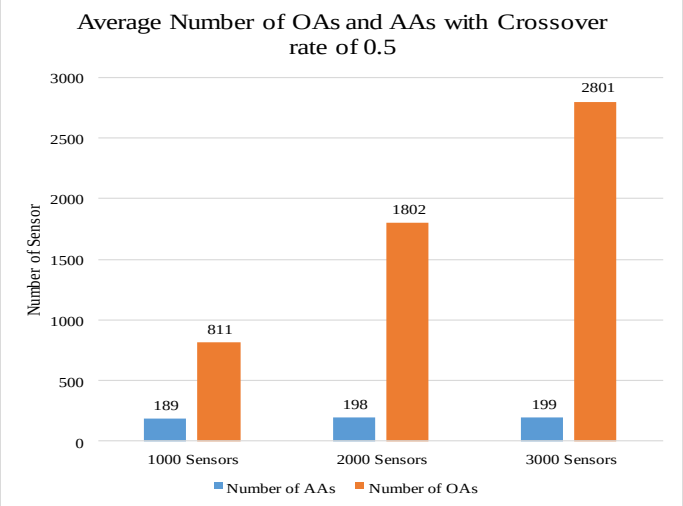


Figure 5.12: Average number of OAs and AAs obtained with crossover rate of 0.5

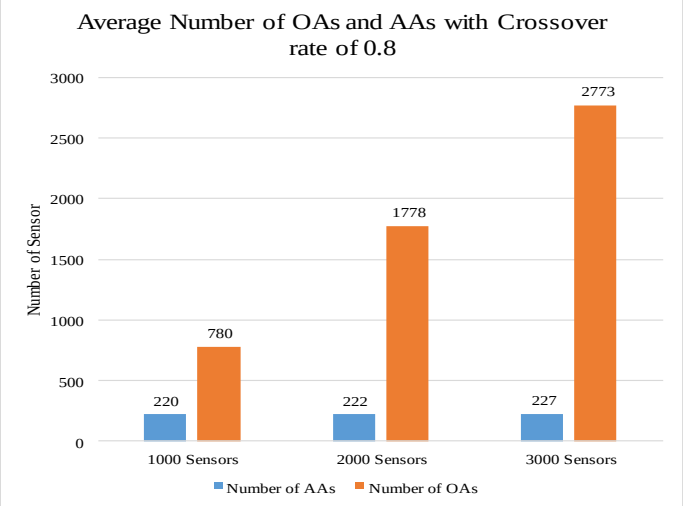


Figure 5.13: Average number of OAs and AAs obtained with crossover rate of 0.8

5.6 Lessons Learned

An important lesson learned during this work is that the creation of ontologies can be accomplished easily using available technologies and tools. The open source technologies and tools like MySQL and Protégé OWL API can be readily used for this purpose. However, the importance of an ontology developer having the domain knowledge cannot be ruled out.

The process of creating ontology in a standard format can be simplified so that a domain expert with limited IT/technical knowledge is able to rapidly create and manage ontologies. Another lesson is that even though annotated sensor data is used by third party applications and may not be relevant to the WSN IaaS owner, it still makes sense to have sensor data storage mechanisms at the WSN infrastructure to allow for historic or performance analysis of sensors and their data. Such historic sensor data can be used for data visualisation or analysis or can be shared with public using the Linking Open Data Cloud project [73].

5.7 Conclusion

This chapter presented the work regarding ontology development and management tool. The tool uses open source software and technologies to help WSN IaaS owner to easily create, manage and extend the base ontology concepts. A heuristic-based algorithm is also presented to provide a near-optimal set of capable nodes to store the base ontology in a distributed manner. Once a near-optimal set of capable nodes it available, the own files containing the base ontology concepts can be disseminated to the network.

"— What is not started will never get finished."

–Johann Wolfgang von Goethe

Chapter 6

Conclusion

Contents

This chapter provides a summary of the thesis along with some work items for the future work that will aid in extending the work done in this thesis.

6.1 Summary

Since their mainstream introduction towards the end of 20th century, WSN deployments have been used as means to bridge the gap between the physical world and the virtual world. With their ability to sense, compute and communicate, WSNs provide their users with the ability to react to various physical phenomenon and take required actions. WSNs are used in a plurality of application domains. The most obvious drawback of the current WSN deployments is that they are domain-specific, task-oriented, and are tailored for particular applications with little or no possibility of reusing them for newer applications. This strategy is inefficient and leads to redundant deployments when new applications are contemplated. WSNs are considered as basic building blocks of IoT paradigm where sensors, along with multitude of everyday objects communicate, interact and share data on a massive scale. Therefore, it is not unrealistic to envision that future WSN deployments will have to support multiple applications simultaneously.

This thesis proposed several architectural solutions to efficiently provision applications and services concurrently over vWSNs. Each architectural solution is based on identified principles that are used to satisfy identified user requirements. The proposed architectures are designed independent of any platform or protocol. Proof-of-concept prototypes, consisting of difference sensor kits, were used to validate all proposed solutions. In total five contributions were made in this thesis.

Firstly, a comprehensive state-of-the-art review is presented which is a new contribution in this domain. A clear taxonomy along with technical details of existing works is presented. Each work is critically evaluated using a set of requirements. Several pertinent research projects are also discussed along with future research issues with hints on their potential solutions.

Second contribution is presented in two parts, one is a novel WSN virtualization architecture that allows provisioning of concurrent applications. Using the architecture, WSN deployments can be realized as vWSN IaaS. The architecture uses the concept of overlays and is used as basis for tackling research problems in later contributions. The architecture is evaluated using a proof-of-concept prototype using Java SunSpot kit. The second part of this contribution is a capable architecture that allows interactions between vWSN IaaS and PaaS for dynamic provisioning of application and services. The architecture is based on the fundamental differences and consist of several functional entities to facilitate the dynamic creation and execution of virtual sensors when requested by PaaS. This architecture is also validated by using Java SunSpot kit.

The third contribution is a novel architecture that allows in-network, distributed, real-time annotation of raw sensor data to provision semantic applications in vWSNs. A new base ontology concept is used to facilitate annotation of sensor data independent of any application domain. The proposed architecture uses the concept of overlay and super-peers to annotate and store the base ontology in the network. A proof-of-concept prototype is used to evaluate the proposed architecture using Java SunSpot and AdvanticSys kit.

The fourth contribution of the thesis is architectural solution for vWSN IaaS owner to easily create and manage base ontology along with a heuristic-based genetic algorithm is used to select a set of capable nodes to store the base ontology in an optimal manner. A proof-of-concept prototype is used to evaluate the proposed architecture using Java SunSpot while simulation results of the proposed algorithm are presented.

6.2 Future Work

WSN virtualization is a very vast topic and demands thorough investigation and solutions for various technical challenges. Paper I provides comprehensive list of research challenges, along with potential avenues to solve them. In this section we present some new work items that will compliment and extend the work done in this thesis.

The first future work item is to address how vWSN IaaS can efficiently publish or advertise its available sensors and their services. Paper III, VI, V, VI and VII all assume a static publication process where the WSN owners publish their nodes to a central repository, however a dynamic publication is required. A P2P based architecture can be a solution like the one in [75] that does not rely on any central mechanism to discover the services. However, no such solution exists for virtualized WSNs. Recent IETF service discovery protocols like

CoAP resource discovery [76], [77] and DNS-SD [78] can be used to design efficient discovery and publication solutions in resource-constrained environments.

This is important because eventually this information will be used by PaaS to compose and deploy new applications and services. As a potential solution semantically enriched vWSN IaaS can play a pivotal role and aid a PaaS to discover and use available sensors and their services. vWSN IaaS and their resources could be formalized as ontologies hiding the heterogeneity and complexity and published for different PaaS. A simple publication mechanism could work for a particular PaaS but in order to allow discovery from multiple PaaS requires a standard solution. Yet another direction is that a service recommendation system for virtualized WSNs, for context-aware discovery of resources and services.

The second future work item is extending the vWSN IaaS architecture presented in Paper VII to provide virtual sensor reservation in-advance and subscription based notification mechanisms. Using this mechanism, users such as PaaS could pre-book their required sensors and setup periodic notification for their applications. The issue of reservation seems tricky but in WSNs it makes sense because often applications are interested to get information about real-world events at a particular location and at a particular time. For example, if application A is interested to use sensors in room X at time T but all are busy then some kind of solution is needed to address this issue. At a higher level of abstraction, this issues is essentially the mapping of end-user application requirements to the available sensors. Once the reservation is done, then the end-user can subscribe to receive sensor data according to its requirements. This idea could be extended to a scenario where two vWSN IaaS could lease their resources to each other against certain incentives like the idea discussed in [79].

The last future work item concerns an important issue to monitor virtual sensors in real time and tackle their failures at vWSN IaaS. This issue concerns the management aspects of vWSN and it is not tackled in this thesis. This issue is very important because the end-user applications and services will be hosted in containers at PaaS and will be linked to virtual sensors to receive sensor data. In ideal situation everything will work smoothly but as WSN are prone to failures, it is important to work on mitigation strategies when a virtual sensor fails and minimize the impact on the applications at PaaS. Issues like finding another virtual sensor in real-time are not trivial since the new virtual sensor may not fulfil the spatial and temporal requirements of the application. As a potential solution, an active monitoring mechanism at vWSN IaaS will be useful to take care of such failures. By using a pre-emptive approach, the PaaS could be notified before such failures occur.

Bibliography

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," Computer Networks, vol. 38, no. 4, pp. 393–422, Mar. 2002.

[2] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292-2330, Aug. 2008.

[3] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy, "Smart objects as building blocks for the Internet of things," IEEE Internet Computing, vol. 14, no. 1, pp. 44–51, Jan. 2010.

[4] M. A. Feki, F. Kawsar, M. Boussard, and L. Trappeniers, "The Internet of Things: The Next Technological Revolution," Computer, vol. 46, no. 2, pp. 24–25, 2013.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," Commun. ACM, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[6] L. M. Vaquero, L. Rodero-Merino, J. Caceres and M. Lindner, "A break in the clouds," ACM SIGCOMM Computer Communication Review, vol. 39, pp. 50, 2008.

[7] A. Khan, A. Zugenmaier, D. Jurca and W. Kellerer, "Network virtualization: a hypervisor for the Internet?," IEEE Communications Magazine, vol. 50, pp. 136-143, 2012.

[8] A. Merentitis, et al., "WSN Trends: Sensor Infrastructure Virtualization as a Driver Towards the Evolution of the Internet of Things," presented at the UBICOMM 2013, The Seventh International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, 2013, pp. 113–118.

[9] Ramdhany, Rajiv and Coulson, Geoff. "Towards the Coexistence of Divergent Applications on Smart City Sensing Infrastructure" Proceedings of 4th International Workshop on Networks of Cooperating Objects for Smart Cities 2013 (CONET/UBICITEC 2013), Philadelphia, USA, April 8, 2013: pp.26-30

[10] S. Abdelwahab, B. Hamdaoui, M. Guizani, and A. Rayes, "Enabling Smart Cloud Services Through Remote Sensing: An Internet of Everything Enabler," IEEE Internet of Things Journal, vol. 1, no. 3, pp. 276–288, Jun. 2014.

[11] M. Fazio, M. Paone, A. Puliafito, and M. Villari, "Huge amount of heterogeneous sensed data needs the cloud," in 2012 9th International Multi-Conference on Systems, Signals and Devices (SSD), 2012, pp. 1–6.

[12] I. Khan, et al., "Wireless Sensor Network Virtualization: Early Architecture and Research Perspectives", Accepted, IEEE Network. 2015

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

[14] P. Levis and D. Culler: "Mat: A tiny virtual machine for sensor networks, In ASPLOSX: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, 2002, pp. 85-95.

[15] A. P. Jayasumana, Q. Han, and T. H. Illangasekare, "Virtual Sensor Networks - A Resource Efficient Approach for Concurrent Applications," in Fourth International Conference on Information Technology, 2007. ITNG '07, 2007, pp. 111–115.

[16] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," J Internet Serv Appl, vol. 1, no. 1, pp. 7–18, Apr. 2010.

[17] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," IEEE Communications Surveys Tutorials, vol. 7, no. 2, pp. 72–93, Second 2005.

[18] B. Beverly Yang and H. Garcia-Molina, "Designing a super-peer network," in 19th International Conference on Data Engineering, 2003. Proceedings, 2003, pp. 49–60.

[19] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff, "Semantic annotation, indexing, and retrieval," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 2, no. 1, pp. 49–79, Dec. 2004.

[20] M. Compton, et al., "The SSN ontology of the W3C semantic sensor network incubator group," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 17, pp. 25–32, Dec. 2012.

[21] R. B. Smith, "SPOTWorld and the Sun SPOT," in Proceedings of the 6th International Conference on Information Processing in Sensor Networks, New York, NY, USA, 2007, pp. 565–566.

[22] Advanticsys. May 2015. http://www.advanticsys.com/shop/prokit-p-27.html - [accessed 25/05/2015]

[23] D. Simon and C. Cifuentes, "The Squawk Virtual Machine: Java^{TM} on the Bare Metal," in Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, New York, NY, USA, 2005, pp. 150–151.

[24] P. Levis, et al., "TinyOS: An Operating System for Sensor Networks," in Ambient Intelligence, W. Weber, J. M. Rabaey, and E. Aarts, Eds. Springer Berlin Heidelberg, 2005, pp. 115–148.

[25] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in 29th Annual IEEE International Conference on Local Computer Networks, 2004, 2004, pp. 455–462.

[26] V. Medina and J. M. Garca, "A survey of migration mechanisms of virtual machines," ACM Computing Surveys, vol. 46, pp. 1-33, 2014.

[27] M. M. Islam, M. M. Hassan, G.-W. Lee, and E.-N. Huh, "A Survey on Virtualization of Wireless Sensor Networks," Sensors, vol. 12, no. 2, pp. 2175–2207, Feb. 2012.

[28] M. M. Islam and E.-N. Huh, "Virtualization in Wireless Sensor Network: Challenges and Opportunities," Journal of Networks, vol. 7, no. 3, Mar. 2012.

[29] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun, "Supporting Concurrent Applications in Wireless Sensor Networks," in Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, New York, NY, USA, 2006, pp. 139–152.

[30] I. Khan, F. Belqasmi, R. Glitho, and N. Crespi, "A multi-layer architecture for wireless sensor network virtualization," in Wireless and Mobile Networking Conference (WMNC), 2013 6th Joint IFIP, 2013, Dubai, UAE, pp. 1–4.

[31] M. Ceriotti, et al., "Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment," in Proceedings of the 2009 International Conference on Information Processing in Sensor Networks, Washington, DC, USA, 2009, pp. 277–288.

[32] X. Wang, J. Wang, Z. Zheng, Y. Xu, and M. Yang, "Service Composition in Service-Oriented Wireless Sensor Networks with Persistent Queries," in 6th IEEE Consumer Communications and Networking Conference, 2009. CCNC 2009, 2009, pp. 1–5.

[33] R. Chu, L. Gu, Y. Liu, M. Li, and X. Lu, "SenSmart: Adaptive Stack Management for Multitasking Sensor Networks," IEEE Transactions on Computers, vol. 62, no. 1, pp. 137–150, Jan. 2013.

[34] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," presented at the The 32nd IEEE International Conference on Computer Communications (INFOCOM 2013), 2013.

[35] S. Saruwatari, M. Suzuki, and H. Morikawa, "PAVENET OS: A Compact Hard Real-Time Operating System for Precise Sampling in Wireless Sensor Networks," SICE Journal of Control, Measurement, and System Integration, vol. 5, pp. 24–33, 2012.

[36] W. Dong, C. Chen, X. Liu, Y. Liu, J. Bu, and K. Zheng, "SenSpire OS: A Predictable, Flexible, and Efficient Operating System for Wireless Sensor Networks," IEEE Transactions on Computers, vol. 60, no. 12, pp. 1788–1801, Dec. 2011.

[37] V. Gupta, et al., "Nano-CF: A coordination framework for macro-programming in Wireless Sensor Networks," in 2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2011, pp. 467–475.

[38] S. Bhattacharya, A. Saifullah, C. Lu, and G. Roman, "Multi-Application Deployment in Shared Sensor Networks Based on Quality of Monitoring," in 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010, pp. 259–268.

[39] C.-L. Fok, G.-C. Roman, and C. Lu, "Agilla: A Mobile Agent Middleware for Self-adaptive Wireless Sensor Networks," ACM Trans. Auton. Adapt. Syst., vol. 4, no. 3, pp. 16:1–16:26, Jul. 2009.

[40] Q. Cao, T. Abdelzaher, J. Stankovic, and T. He, "The LiteOS Operating System: Towards Unix-Like Abstractions for Wireless Sensor Networks," in International Conference on Information Processing in Sensor Networks, 2008. IPSN '08, 2008, pp. 233–244.

[41] D. Simon, et al. "Java^{TM} on the Bare Metal of Wireless Sensor Devices: The Squawk Java Virtual Machine," in Proceedings of the 2nd International Conference on Virtual Execution Environments, New York, NY, USA, 2006, pp. 78–88.

[42] J. Koshy and R. Pandey, "VMSTAR: Synthesizing Scalable Runtime Environments for Sensor Networks," in Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, New York, NY, USA, 2005, pp. 243–254.

[43] S. Bhatti, J. et al, "MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms," Mob. Netw. Appl., vol. 10, no. 4, pp. 563–579, Aug. 2005.

[44] P. Levis and D. Gay, TinyOS Programming. Cambridge University Press, 2009.

[45] P. Levis and D. Culler: "Maté: A tiny virtual machine for sensor networks." In ASPLOSX: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, 2002, pp. 85-95.

[46] I. Khan, F. Belqasmi, R. Glitho, and N. Crespi, "A multi-layer architecture for wireless sensor network virtualization," in Wireless and Mobile Networking Conference (WMNC), 2013 6th Joint IFIP, 2013, Dubai, UAE, pp. 1–4.

[47] J. Hoebeke, et al., "Managed Ecosystems of Networked Objects," Wireless Pers Commun, vol. 58, no. 1, pp. 125–143, May 2011.

[48] I. Ishaq, J. Hoebeke, I. Moerman, and P. Demeester, "Internet of Things Virtual Networks: Bringing Network Virtualization to Resource-Constrained Devices," in 2012 IEEE International Conference on Green Computing and Communications (GreenCom), 2012, pp. 293–300.

[49] R. Tynan, G. M. P. O'Hare, M. J. O'Grady, and C. Muldoon, "Virtual Sensor Networks: An Embedded Agent Approach," in International Symposium on Parallel and Distributed Processing with Applications, 2008. ISPA '08, 2008, pp. 926–932.

[50] A. P. Jayasumana, Q. Han, and T. H. Illangasekare, "Virtual Sensor Networks – A Resource Efficient Approach for Concurrent Applications," in Fourth International Conference on Information Technology, 2007. ITNG '07, 2007, pp. 111–115.

[51] H. M. N. D. Bandara, A. P. Jayasumana, and T. H. Illangasekare, "Cluster Tree Based Self Organization of Virtual Sensor Networks," in 2008 IEEE GLOBECOM Workshops, pp. 1-6.

[52] Q. Han, A. P. Jayasumana, T. Illangaskare, and T. Sakaki, "A wireless sensor network based closed-loop system for subsurface contaminant plume monitoring," in IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008, 2008, pp. 1–5.

[53] M. Haghighi and D. Cliff, "Multi-agent Support for Multiple Concurrent Applications and Dynamic Data-Gathering in Wireless Sensor Networks," in 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013, pp. 320–325.

[54] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, "SenShare: Transforming Sensor Networks into Multi-application Sensing Infrastructures," in Wireless Sensor Networks, G. P. Picco and W. Heinzelman, Eds. Springer Berlin Heidelberg, 2012, pp. 65–81.

[55] L. Sarakis, T. Zahariadis, H.-C. Leligou, and M. Dohler, "A framework for service provisioning in virtual sensor networks," J Wireless Com Network, vol. 2012, no. 1, pp. 1–19, Dec. 2012.

[56] A. Majeed and T. A. Zia, "Multi-set Architecture for Multi-applications Running on Wireless Sensor Networks," in 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2010, pp. 299–304.

[57] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun, "Supporting Concurrent Applications in Wireless Sensor Networks," in Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, New York, NY, USA, 2006, pp. 139–152.

[58] P. Levis, "Experiences from a Decade of TinyOS Development," in Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, Berkeley, CA, USA, 2012, pp. 207–220.

[59] M. Schulz, "mbed Cookbook - IoT," May 2015. [Online]. Available: https://developer.mbed.org/cookbook/IOT - [accessed 25/05/2015]

[60] R. N. Murty, G. Mainland, I. Rose, A. R. Chowdhury, A. Gosain, J. Bers, and M. Welsh, "CitySense: An Urban-Scale Wireless Sensor Network and Testbed," in 2008 IEEE Conference on Technologies for Homeland Security, 2008, pp. 583–588.

[61] F. Berkers, et al., "Constructing a multi-sided business model for a smart horizontal IoT service platform," in 2013 17th International Conference on Intelligence in Next Generation Networks (ICIN), 2013, pp. 126–132.

[62] A. Andrushevich, B. Copigneaux, R. Kistler, A. Kurbatski, F. L. Gall, and A. Klapproth, "Leveraging Multi-domain Links via the Internet of Things," in Internet of Things, Smart Spaces, and Next Generation Networking, S. Balandin, S. Andreev, and Y. Koucheryavy, Eds. Springer Berlin Heidelberg, 2013, pp. 13–24.

[63] Preon32 sensor kit – http://www.virtenio.com/en/products/evaluationskits.html – [accessed 25-05-2015]

[64] Phidgets kit – http://www.phidgets.com/products.php?category=18 – [accessed 25-05-2015]

[65] C. Efstratiou, I. Leontiadis, C. Mascolo, and J. Crowcroft, "A Shared Sensor Network Infrastructure," in Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, New York, NY, USA, 2010, pp. 367–368.

[66] L. Sanchez, et al., "SmartSantander: IoT experimentation over a smart city testbed," Computer Networks, vol. 61, pp. 217–238, Mar. 2014.

[67] D. Irwin, et al. "Towards a virtualized sensing environment," in Testbeds and Research Infrastructures. Development of Networks and Communities. Springer Berlin Heidelberg, 2011, pp. 133-142.

[68] A. Anitha, J. JayaKumari, and G. V. Mini, "A survey of P2P overlays in various networks," in 2011 International Conference on Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), 2011, pp. 277–281.

[69] R. Lopes, F. Assis, and C. Montez. "MASPOT: A mobile agent system for Sun SPOT., In Autonomous decentralized systems (ISADS), 2011 10th international symposium on, pp. 25-31. IEEE, 2011. Kobe, Japan.

[70] S. Cho, S. Chang, and Y. Han, "A fully distributed coordination scheme based on orthogonal requests and responses," in 2014 16th International Conference on Advanced Communication Technology (ICACT), 2014, pp. 804–807.

[71] F. Crivellaro, "μJena: Gestione di ontologie sui dispositivi mobile," Thesis, M.Sc., Politecnico di Milano, Milan, Italy, 2007.

[72] J.-M. Kim, S.-H. Park, Y.-J. Han, and T.-M. Chung, "CHEF: Cluster Head Election mechanism using Fuzzy logic in Wireless Sensor Networks," in 10th International Conference on Advanced Communication Technology, 2008. ICACT 2008, vol. 1, pp. 654–659.

[73] Apache Software Foundation, "Apache commons mathematics library," 2015, http://commons.apache.org/math/

[74] S. Amit, Semantic Services, Interoperability and Web Applications: Emerging Concepts: Emerging Concepts. IGI Global, 2011.

[75] J. Maenpaa, J. J. Bolonio, and S. Loreto, "Using RELOAD and CoAP for wide area sensor and actuator networking," J Wireless Com Network, vol. 2012, no. 1, pp. 1–22, Dec. 2012.

[76] Z. Shelby, "Embedded web services," IEEE Wireless Communications, vol. 17, no. 6, pp. 52–57, Dec. 2010.

[77] Z. Shelby, et al. "Constrained Application Protocol (CoAP), draft-ietf-core-coap-18", work in progress. The Internet Engineering Task Force–IETF, June (2013).

[78] S. Cheshire, and M. Krochmal, "Multicast DNS", IETF RFC 6762, February 2013.

[79] Y. Zhang, J. Wen, "An IoT electric business model based on the protocol of BitCoin", in 2015 18th Int. Conf. Intelligence in Next Generation Networks: Innovations in Services, Networks and Clouds (ICIN 2015), Paris, France, Feb. 2015.

Annex A

Paper I

Wireless Sensor Network Virtualization: A Survey

Imran Khan, *Student Member, IEEE*, Fatna Belqasmi, *Member, IEEE*, Roch Glitho, *Senior Member, IEEE*, Noel Crespi, *Senior Member, IEEE*, Monique Morrow, *Senior Member, IEEE*, and Paul Polakos

***Abstract*—Wireless Sensor Networks (WSNs) are the key components of the emerging Internet-of-Things (IoT) paradigm. They are now ubiquitous and used in a plurality of application domains. WSNs are still domain specific and usually deployed to support a specific application. However, as WSNs' nodes are becoming more and more powerful, it is getting more and more pertinent to research how multiple applications could share a very same WSN infrastructure. Virtualization is a technology that can potentially enable this sharing. This paper is a survey on WSN virtualization. It provides a comprehensive review of the state-of-the-art and an in-depth discussion of the research issues. We introduce the basics of WSN virtualization and motivate its pertinence with carefully selected scenarios. Existing works are presented in detail and critically evaluated using a set of requirements derived from the scenarios. The pertinent research projects are also reviewed. Several research issues are also discussed with hints on how they could be tackled.**

***Index Terms*—Wireless Sensor Network (WSN), Internet-of-Things (IoT), virtualization, node-level virtualization, network-level virtualization.**

I. Introduction

THE emerging Internet-of-Things (IoT) concept is considered to be the next technological revolution, one that realizes communication between many types of objects, machines and devices, and at an unprecedented scale [1]. WSNs can be seen as the basic constituents of IoT because they can help users (humans or machines) to interact with their environment and react to real-world events. These WSNs are composed of nodes that are amalgamations of micro-electro-mechanical systems, wireless communications and digital electronics, and have the ability to sense their environment, perform computations and communicate [2]. The most obvious drawback of the current WSNs is that they are domain-specific and task-oriented, tailored for particular applications with little or no possibility of reusing them for newer applications. This strategy is inefficient and leads to redundant deployments when new applications are contemplated. With the introduction of the IoT, it is not unrealistic to envision that future WSN deployments will have to support multiple applications simultaneously.

Virtualization is a well-established concept that allows the abstraction of actual physical computing resources into logical units, enabling their efficient usage by multiple independent users [3]. It is a promising technique that can allow the efficient utilization of WSN deployments, as multiple applications will be able to co-exist on the same virtualized WSN. Virtualization is a key technique for the realization of the Future Internet [4] and it is indeed quite pertinent to explore it in the context of WSNs.

Virtualizing WSNs brings with it many benefits; for example, even applications that were not envisioned a priori may be able to utilize existing WSN deployments. A second, related benefit is the elimination of tight coupling between WSN services/ applications and WSN deployments. This allows experienced as well as novice application developers to develop innovative WSN applications without needing to know the technical details of the WSNs involved. Another benefit is that WSN applications and services can utilize as well as be utilized by third-party applications. It can also help to define a business model, with roles such as physical WSN provider, virtual WSN provider and WSN service provider.

The WSN virtualization concept can be applied to several interesting application areas. Recent advances in smart phones and autonomous vehicles [5] have made it possible to have multiple on-board sensors on them. Mobile crowd sensing is one area that can take advantage of virtualizing these sensors through participatory and opportunistic sensing [6] and [7]. An opportunistic urban sensing scenario is presented in [7] in which thousands of sensors are required to monitor the CO_2 concentration in an urban city. Instead of deploying these sensors and managing them, WSN virtualization can be used as a key enabling technology to utilize sensors from citizens to provide the required data. Similarly, Sensing-as-a-Service (SaaS) model is presented in [8] along with several use case scenarios. WSN virtualization can help realize the SaaS model through cost-efficient utilization of deployed sensors. Several other motivational examples can be found in [9] and [10].

Of course there are many technical challenges to resolve before such utilization takes place but they also provide a strong motivation for a deeper and complete search space exploration to propose innovative solutions in this area. Many researcher now consider WSN virtualization as a key enabling technology and provide its motivation. According to the authors in

Manuscript received October 27, 2014; revised March 2, 2015; accepted March 3, 2015. This work is partially supported by CISCO systems through grant (CG-576719), European ITEA-2 funded project Web-of-Objects (WoO), and by the Canadian Natural Sciences and Engineering Research Council (NSERC) through the Canada Research Chair in End-User Service Engineering for Communications Networks.

I. Khan and N. Crespi are with the Institut Mines-Télécom, Télécom SudParis, 91011 Evry, France (e-mail: imran@ieee.org; noel.crespi@it-sudparis.eu).

F. Belqasmi is with the College of Innovative Technology, Zayed University, Abu Dhabi, UAE (e-mail: fatna.belqasmi@zu.ac.ae).

R. Glitho is with the Concordia Institute for Information Systems Engineering (CIISE), Concordia University, Montreal, QC H3G 2W1, Canada (e-mail: glitho@ece.concordia.ca).

M. Morrow and P. Polakos are with CISCO Systems, Inc., San Jose, CA 95134 USA (e-mail: mmorrow@cisco.com; ppolakos@cisco.com).

Digital Object Identifier 10.1109/COMST.2015.2412971

[11], WSN virtualization is a powerful enabler for information sharing in the context of IoT by using it along with data analysis techniques. A smart city environment is considered in [12], where WSN virtualization could be used to efficiently utilize the deployed infrastructure. To achieve this type of utilization, the use of multiple concurrency models is advised, depending on the usage context. In [13], WSN virtualization is discussed as a key enabler to promote resource efficiency, with a cooperative model that captures several aspects of WSN virtualization. In [14] WSN virtualization is envisioned as an important technology to create large-scale sensor platforms that are used to satisfy efficient usage of network resources.

There are surveys (e.g., [15]) that cover wireless network virtualization at large, but they do not focus on the specifics of WSN virtualization. Although it is a key enabling technology, the few surveys published to date on WSN virtualization (e.g., reference [16], reference [17]), have several limitations. They do not include real world motivating scenarios and are also dated because they do not review the most recent developments in the area. Furthermore they lack comprehensiveness in terms of what is reviewed and how it is reviewed. There is for instance no well-defined yardstick for the critical analysis of the state of the art. In addition, they do not elaborate on potential solutions when it comes to research directions.

This paper is a survey on wireless sensor network virtualization. It aims at addressing the shortcomings of the very few surveys published so far on the topic. From that perspective it makes the following contributions:

- Real world motivating scenarios for WSN virtualization.
- Comprehensive and in-depth review of the state of the art including the most recent developments in the area.
- Critical analysis of the state of the art using well defined yard-sticks derived from the motivating scenarios.
- An overview of the open issues along with insights on how they might be solved.

In Section II we discuss the basics of WSN virtualization concepts and its types. In Section III, we first present the motivating scenarios and then provide a set of requirements. Based on these requirements we critically review the state-of-the-art in Section IV. Relevant WSN virtualization projects are discussed in Section V. Section VI outlines several research directions and Section VII concludes the paper.

II. WSN Virtualization Basics

WSN virtualization can be broadly classified into two categories: Node-level virtualization and Network-level virtualization. In this section we discuss both these categories.

A. Node-Level Virtualization

WSN node-level virtualization allows multiple applications to run their tasks concurrently on a single sensor node [18], so that a sensor node can essentially become a multi-purpose device. The basic concept of node level virtualization is illustrated in Fig. 1. There are two ways to achieve node-level virtualization: Sequential and Simultaneous execution.

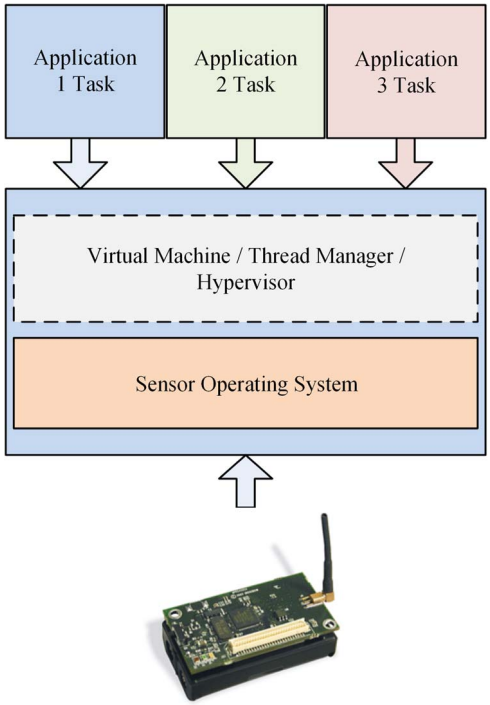


Fig. 1. Execution of multiple applications in a general purpose WSN node.

Sequential execution can be termed a weak form of virtualization, in which the actual execution of application tasks occurs one-by-one (in series). The advantage of this approach is its simple implementation, while the obvious disadvantage is that applications have to wait in a queue. In simultaneous execution, application tasks are executed in a time-sliced fashion by rapidly switching the context from one task to another. The advantage of this approach is that application tasks that take less time to execute will not be blocked by longer running application tasks, while the disadvantage is its complex implementation.

B. Network-Level Virtualization

It is WSN network-level virtualization that enables a Virtual Sensor Network (VSN). A VSN is formed by a subset of a WSN's nodes that is dedicated to one application at a given time [19]. Enabling the dynamic formation of such subsets ensures resource efficiency, because the remaining nodes are available for different multiple applications (even for applications that had not been envisaged when the WSN was deployed), although not necessarily simultaneously.

WSN network-level virtualization can be achieved in two different ways. One way is by creating multiple VSNs over the same underlying WSN infrastructure, as illustrated in Fig. 2(a). WSN nodes that are not part of any VSN remain available for other applications or network functions, such as routing. The second way is where a VSN is composed of WSN nodes from three different administrative domains, as shown in Fig. 2(b), facilitating data exchange between them that would not be possible otherwise.

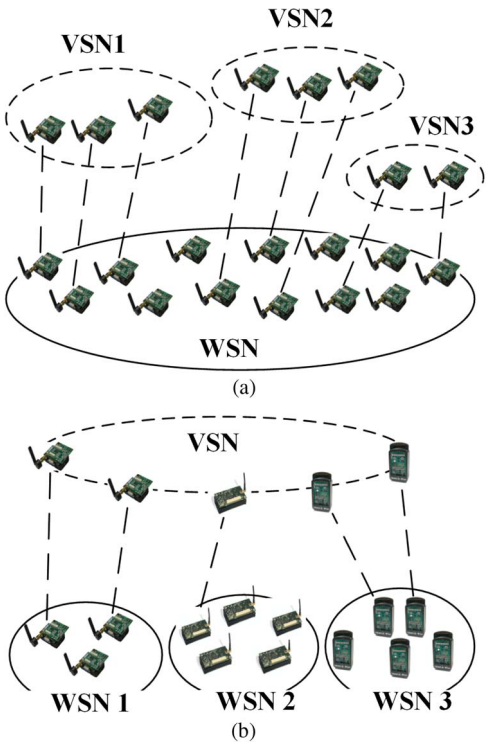


Fig. 2. VSN concept. (a) Multiple VSNs over single WSN. (b) Single VSN over multiple WSNs.

III. WSN Virtualization—Motivating Scenarios and Requirements

In this section we first present two scenarios that are derived from the literature, and then come up with a set of requirements. Using these requirements we critically review the existing work, grouping our summation of that work under three types: node-level virtualization, network-level virtualization, and hybrid solutions.

A. Motivating Scenarios

The scenarios described here illustrate the motivation and benefits of using WSN virtualization in common WSN deployments.

1) Fire Monitoring Scenario: Consider the example of a city near an area where brush fires are common [9]. We assume that the city administration is interested in the early detection of fire eruption and in its course, using a WSN and a fire contour algorithm to determine the curve, shape and direction of fire. One approach is that the city administration could deploy WSN nodes all over the city (i.e., on each street and at individual houses), but this is not very efficient because some individuals may have already deployed WSN nodes in their homes to detect fires. A more efficient approach would be for the city administration to deploy WSN nodes to areas under its jurisdiction, i.e., streets and parks, and to re-use the WSN nodes already deployed in private homes. In this scenario, two different applications share the same WSN infrastructure: one, belonging to home owners, is confined to private WSNs deployed in individual houses, and the other belongs to the city administration and shares the private WSN nodes with the WSN nodes deployed by the city administration. Periodic notification or query-based models are not suitable because the city administration application requires complete access to all the WSN nodes for adaptive sampling.

Another issue is that to execute a fire contour algorithm in a distributed fashion, WSN nodes need to exchange fire notification messages with each other. The query-based data exchange approach is not efficient as it will force the execution of the fire contour algorithm at a remote centralized location, since two WSN nodes located in their respective private domains cannot exchange data directly. An overlay network is one possible solution. This scenario illustrates the need for WSN virtualization, as two different users need to share a common resource, i.e., WSN nodes.

2) Heritage Building Monitoring: A real-world deployment of a WSN is presented in [20], in which a WSN is used to monitor the impact of constructing a road tunnel under an ancient tower in Italy, as it was feared that the tower could lose its ability to stand on its own and collapse during the construction. Now consider that there are three users interested in the fate of the tower. The first is the construction company, as it needs to make sure that the tower does not lose its ability to stand on its own, otherwise it will have to pay a heavy fine. The second user is the conservation board that routinely monitors all the ancient sites around the city, and the third user is the local municipality which will have to plan emergency remedial/rescue actions in case the tower falls during the construction.

It is quite possible that the conservation board has already deployed its own WSN to monitor the health of ancient sites including this tower. In this case the construction company and the local municipality can use the existing sensor nodes during the construction period. In the absence of WSN virtualization, there are only two possible solutions. One is to rely on the information provided by the conservation board's application. However this information may not be at the required granularity level. Worse, some of the information that is needed might simply not be available because the requirements of the construction company and of the local municipality were not considered when the conservation board application was designed and implemented. The second solution is that each user deploys redundant WSN nodes. Here WSN virtualization can play a pivotal role by fulfilling the requirements of each user.

B. Requirements

In this section we present a list of eight requirements, derived from the scenarios mentioned above. In Table IV we indicate if the existing solutions meet our identified requirements, and to what degree.

The *first* requirement is the availability of node-level virtualization. This is a fundamental requirement which ensures that the sensor nodes can support the concurrent execution of multiple applications.

The *second* requirement is network-level virtualization, which concerns the ability of sensor nodes to dynamically form groups to perform the isolated and transparent execution of multiple application tasks in such a way that each group belongs to a different application.

The *third* requirement is support for application/service priority. It is our observation that most WSNs are deployed for mission-critical situations like security, fire monitoring, battlefield conditions and surveillance. In such situations, mission-critical applications/services should have prioritized execution mechanisms.

The *fourth* requirement is that any WSN virtualization solution should be platform-independent and thus should not depend on a particular hardware or software platform.

The *fifth* requirement is that the proposed solution should have a resource discovery mechanism, for both neighbor discovery and service discovery.

The *sixth* requirement is based on the applicability of the proposed solution to resource-constrained sensor nodes, including early generation sensor nodes. Mechanisms to allow legacy sensor nodes to become part of a WSN virtualization solution are also covered by this requirement.

The *seventh* requirement is heterogeneity, which means that the solution should be applicable to a variety of WSN platforms with different capabilities (e.g., processing power, memory). These platforms would include MICAZ, MICA2, Atmel AVR family, and MPS430 among others.

The *eight* requirement is the ability to select sensor nodes for application tasks. When multiple applications concurrently utilize a deployed WSN, selection of proper sensor nodes is very important because applications may have spatial and temporal requirements [21].

IV. State-of-the-Art

In this section we present the state-of-the-art and analyze it critically. We categorize the existing work as Node-level virtualization, Network-level virtualization and Hybrid solutions. Hybrid solutions combine both node- and network-level virtualization. Each category is further classified based on the approaches used.

A. Node-Level Virtualization

We group the Node-level virtualization approaches under two umbrellas: sensor operating system (OS) based solutions and Virtual Machine-/Middleware (VM/M) based solutions. In sensor OS-based solutions, the node-level virtualization is part of the sensor OS. In VM/M-based solutions, the node-level virtualization is performed by a component running on top of the sensor's OS.

Node-level virtualization solutions use two types of programming models; event-driven and thread-based. Event-driven programming model is simple to implement in sensors. Event-driven programs have a main loop that listens for the events, e.g., the temperature value going above a threshold. When the event occurs a callback function is called to handle the event, using an event-handler. When a program is blocked, by an I/O event, its event-handler simply returns the control without involving context switching. Thread-based model is more difficult to implement in sensors, due to limited resources and use of common address space. Each program consist of multiple threads, and when a thread is blocked, context switching is required to execute other threads [22].

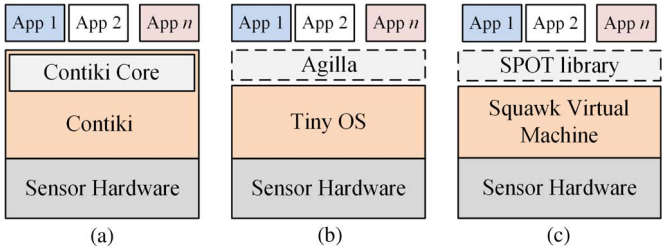


Fig. 3. Example node-level virtualization solutions. (a) OS-based solution (e.g., Contiki). (b) Middleware-based solution (e.g., Agilla). (c) Virtual machine-based solution (e.g., Squawk VM).

Fig. 3 shows the node-level virtualization types while Table I illustrates the characteristics of the existing works addressing node-level virtualization.

1) Sensor Operating System-Based Solutions: SenSmart [23] is a recent multitasking sensor OS that supports the execution of concurrent application tasks in very resource-constrained sensor nodes. It is designed to tackle the issues associated with the execution of concurrent application tasks. Normally, application tasks have their associated predefined stack space, but in SenSmart the stack allocation is managed dynamically at run time. Initially, each application task gets its default (stack) memory region and time slice, but during its execution SenSmart manages the size and location of the allocated stack in a transparent way. Each application task uses logical addresses at runtime, managed by the OS and mapped onto the physical memory. Stack space can be reclaimed from those tasks that no longer require it. When a new task is scheduled to run, the context of the current task is compressed and saved in a circular buffer for its resumption. The system architecture consists of a base station that compiles the code, links it and eventually distributes it to the sensor node. There is no mention of support for network layer support (6LoWPAN) or any radio protocol.

The support for node-level virtualization is provided by compiling and linking multiple application task codes together in a single code image. The application task codes are programmed in nesC and the compiled binary code of each task is then modified by a rewriter, combined with other binary codes and finally linked with the precompiled kernel runtime. The kernel runtime ensures that the application tasks, when instantiated, follow the multitasking semantics (stack management, context switching) and run concurrently. Once a final executable code is generated, it can be disseminated to the sensor node using any wireless reprogramming approach. The strategy of first compiling and linking all the binary codes together means that there is no separation of OS and application tasks, and, whenever a new application task is contemplated, all of the software of the sensor node is updated. The OS uses an event-driven programming model and follows a sense-and-send workflow model [24].

SenSmart has been implemented in Mica2/MicaZ hardware platforms and evaluated for overhead of common system functions, application benchmarking, and task scheduler performance when concurrent tasks are executed. The overhead of common system functions is within acceptable range especially for important functions such as context saving, restoring

TABLE I
CHARACTERISTICS OF NODE-LEVEL VIRTUALIZATION SOLUTIONS

Solution (Year)	Programming Model	Programming Language	Separation between OS and application tasks	Protocols Supported at different layers	Real-time Applications
SenSmart (2013)	Event-driven	nesC	No	Not discussed	No
RIOT (2013)	Thread-based	ANSI C/C++	Yes	6LoWPAN & RPL	Yes
PAVENET (2012)	Thread-based	C	No	Not discussed	Yes
SenSpire (2011)	Event- and thread-based	CSpire	No	CSMA, CSMA/CA, B-MAC & X-MAC	No
Nano-CF (2011)	Event-driven	Nano-CL	Yes	DSR, TDMA & B-MAC	Yes
UMADE (2010)	Event-driven	nesC	Yes	Not discussed	No
Agilla (2009)	Mobile agent and tuple-space -based	Assembly-like	Yes	Not discussed	No
LiteOS (2008)	Event- and thread-based	C	Yes	Not discussed	No
Squawk VM (2006)	Thread-based	J2ME	No	CTP, 6LoWPAN, AODV, LQRP	No
VMSTAR (2005)	Thread-based	Java	No	Not discussed	No
MANTIS (2005)	Thread-based	C	Yes	TDMA	No
TinyOS (2005)	Event-driven	nesC	No	Geographic routing, flooding, unicast	No
Contiki (2004)	Event- and thread-based	C	Yes	HTTP, COAP, UDP, TCP, RPL, 6LoWPAN	Only for event-driven applications
Maté (2002)	Event-driven	TinyScript	No	Not discussed	No

and switching. All these functions take between 127 μs to 316 μs. For application benchmarking it was found that the same applications use more CPU cycles in SenSmart than in TinyOS. For concurrent tasks, the evaluation found that delays recorded during execution of multiple tasks has same order of magnitude as context switching.

RIOT [25] is the latest attempt to address the challenges of designing a flexible OS for diverse hardware in the IoT. The concept of RIOT is based on the fact that none of the existing OSs, traditional or resource-constrained, are capable of supporting diverse hardware resources in the IoT. The focus of RIOT is to provide features such as real-time multithreading support, a developer-friendly programming model and POSIX-like API based on C/C++, as well as full TCP/IP network stack support for resource-constrained devices using 6LoWPAN and RPL. RIOT is based on microkernel architecture and requires only 1.5 kB of RAM and 5 kB of ROM for a basic application. RIOT can run on 8-bit, 16-bit and full 32-bit processors, and thus has the potential to become unique operating system for diverse hardware devices in the IoT paradigm. This adaptability is achieved by using a hardware abstraction layer. Overall, RIOT takes a modular approach and the system services and the user application tasks run as threads. The scheduler is designed to minimize context switching between threads to few clock cycles. The kernel is based on FireKernel [26] providing maximum reliability and real-time multithreading. System tasks have static memory allocation, but for application threads dynamic memory management is used. RIOT is a work in progress and so far there are no performance results or comparisons with existing OSs, but the code is available on their website.

In the context of WSN virtualization, RIOT uses a real-time thread-based programming model where various system services and application tasks are coded in standard ANSI C/C++ and run in parallel. Threads can be preempted based on their priority. Application tasks are coded independently of the hardware and software, which makes it possible to run them on different devices. In large-scale deployments such as Smart Cities, sensor nodes and other IoT devices (e.g., surveillance cameras) can be programmed conveniently.

So far there are no performance results regarding RIOT OS however, in [27] the authors do present a theoretical comparison of their approach against existing competition without any qualitative or quantitative comparison.

SenSpire OS [28] is another recent effort that supports both event-driven and thread-based programming models. Their work has four main features: predictability—to guarantee that sensor nodes respond to control messages, availability—the nodes remain available for data forwarding when needed, programming mode—which is hybrid, and efficiency—so that the OS can be used on very resource-constrained sensor nodes. Another contribution of SenSpire is a multi-layer (*radio, resource and sensornet layers*) abstraction to develop networked applications. The radio layer makes it possible to write device drivers using different MAC protocols. The resource layer exposes the lower layer and allows different application tasks to use it concurrently. A new object-oriented language (CSpire) is provided to program user application tasks using a hybrid programming model. SenSpire uses static optimizations, meaning that application tasks, their states, and the kernel structures should be known beforehand. This limits its flexibility, a requirement for the real-world deployment of WSNs. The kernel

of SenSpire is written in C and the application tasks are written in CSpire. The paper describes extensive results based on the implementation of SenSpire on Mica2, MicaZ, and TelosB nodes. Its performance at various benchmarks is compared to that of MANTIS [29] and TinyOS [30]. Overall findings indicate that SenSpire offers a performance comparable to those OSs.

For WSN virtualization, SenSpire incorporates both event-driven and thread-based programming models. Tasks can be programmed as events or as threads. Event tasks have higher priority than thread tasks. System tasks are usually implemented as event tasks because they are predictable and easier to maintain. Application tasks are implanted as thread tasks with varying priority levels. A thread task is preempted either by a higher-priority thread task or when it goes to sleep. This set up is unlike other OSs where thread tasks are executed in a time-sliced manner. In SenSpire the threads follow run-to-completion model unless they are preempted by a higher priority thread. The execution of threads is sequential (First-in First-out) when they have the same priority level. The use of CSpire language to program application tasks means a learning curve for developers. Despite using a layered-approach, application tasks are tightly integrated with the OS and so when new application tasks are contemplated, all of the sensor node software is updated.

The performance results of SenSpire OS show that its interrupt latency is less than TinyOS. The overhead of task scheduling is compared against MANTIS OS [29] showing more delay in case of SenSpire. The energy consumption of various tasks including radio and CPU are almost similar to TinyOS.

MANTIS [29] is a thread-based embedded operating system supporting simultaneous execution on sensor nodes. The OS kernel and threads are programmed in C language and are portable across different hardware platforms. There are system-level threads and user-level threads. The OS kernel, scheduler and underlying hardware are exposed as APIs for the user-level threads. MANTIS supports preemptive multithreading by assigning priorities to threads, thereby allowing the interleaving of tasks and avoiding delays. Long-running threads can be preempted by short-running threads. Simultaneous execution of these threads is achieved by context switching. When execution of a thread is suspended, all its current states are stored in its own stack and later retrieved to resume execution. Every thread has an entry in a thread table managed by the kernel. Its size is fixed, hence only a predefined number of user-level threads can be created. The other main features of the OS include a dynamic reprogramming mechanism for deployed sensor nodes, a remote debugging mechanism and an x86-based prototype platform. Dynamic reprogramming options are, the wireless re-flashing of the entire OS, the re-programming of single threads and changing the variables of a thread. The wireless re-flashing of the OS and reprograming of a single thread is mentioned as work-in-progress. A command server is used for remote debugging. The sensor nodes run the client part of the command server. Any user can login to the sensor node and modify its setting, execute or stop threads or restart them. The authors implemented several demanding tasks with MANTIS on MICA2 nodes, including AES and RC5 encryption algorithms, compression/decompression algorithms using arithmetic code, and a 64-bit FFT algorithm. These tasks took low execution time in MANTIS. Normally the concurrent execution of threads leads to context switching overhead and the need for additional stack space. In MANTIS, it was found that while context switching does not incur much performance loss, a stack estimation tool would be helpful.

MANTIS is an interesting option for node-level virtualization, as it is completely thread-based and easier to program without having to manage low-level details of stack/memory. The time-sliced multithreading approach makes it possible to run application tasks simultaneously without using a run-to-completion model. The application threads are coded in C and are independent of the OS. Although MANTIS support dynamic reprogramming but it has not been fully explained in the paper. Currently it is not clear whether the work on MANTIS is underway or not as the project page [31] has quite old information.

The performance results presented in [29] are very limited. No comparison is provided in against other competing solutions. The execution times of some complex tasks (compression/ decompression and RC5 and AES encryption) and power consumption using MICA-2 platform are presented.

LiteOS [32] is a Unix-like OS designed for sensor nodes. It provides rich features, such as a hierarchical file system, a command shell that works wirelessly, kernel support for dynamic execution of multi-threaded applications, debugging support and software updates. LiteOS maps a WSN as a UNIX-like file system where different commands can be executed by the user in familiar UNIX-like manner. There are three components: *i)* LiteShell, *ii)* LiteFS and *iii)* Kernel. LiteShell is a command shell that resides in a base station and is used to communicate with sensor nodes to execute file, process, debugging, environment and device related commands. Within the wireless range, sensor nodes can be mounted by LiteShell, similar to how a USB is connected to a computer. However, this process cannot be achieved via the Internet or by multi-hop communication. The sensor nodes do not maintain any state regarding LiteShell and simply respond to the commands.

LiteFS is a hierarchical file system partitioned into three modules that use RAM, EEPROM and Flash memory, respectively. The RAM holds the open files, and their allocation and data information is in EEPROM and Flash memory, respectively. EEPROM holds the hierarchical directory information and the actual data is stored in Flash memory. The LiteOS programming model supports both event-based and thread-based approaches. The scheduling mechanism is also hybrid and supports priority-based and round-robin based scheduling. User applications are multithread-based, and concurrent threads do not have memory conflicts because there is no memory sharing between them. Overall, LiteOS's architecture is inspired by UNIX and works in a distributed manner. The memory consumption of LiteOS applications is larger than that of TinyOS because LiteOS applications are multithreaded whereas TinyOS applications are singe threaded.

LiteOS offers a flexible approach to implement node-level virtualization. It uses a hybrid programming model hybrid that allows the concurrent execution of application threads and handles events through a call-back mechanism. The application tasks can be programmed in C language. Installing and running application tasks is very simple and can be accomplished by dynamically copying user applications. Another advantage of LiteOS is its separation between applications and the OS through callgates. Callgates are pointers and act as application access points to they can access system software and resources. This means that new applications can be simply loaded on a sensor node without reprogramming the sensor node from scratch.

The performance results of LiteShell show the average response time of commands sent using the LiteShell. The average delay of common network commands is under 500 ms. The delay to send file in the network using copy command depends on the file size. The delay for 4 KB file copy is around 3 seconds to 7.5 seconds for single-hop and two-hop transfer respectively. The length of source code is compared against TinyOS and it is found that the same application can be written in LiteOS using few lines than TinyOS, however because of multi-threading support LiteOS applications take more memory than TinyOS counterparts.

PAVENET [33] OS is a thread-based OS designed to exclusively handle the issues related to the preemption of multithreaded application tasks. However, PAVENET has one major drawback—its non-portability. It only works with PIC18 microchip, and unlike other sensor OSs it cannot be used on other hardware platforms such as MICAZ. Two types of multithreading are provided: preemptive and cooperative. The former is used for real-time tasks (e.g., radio access, sensor sampling) and the latter for best-effort tasks (e.g., routing). PAVENET makes three contributions that deal with the issues of preemption overhead and stack/memory space management; it offers a real-time task scheduler, a best-effort task scheduler and a wireless communication stack to abstract lower layers. To mitigate the effects of switching overheads, the PIC18 chip's functions are used for a real-time task scheduler. One of the functions is the fast return stack that automatically saves the context of a task. The best-effort task scheduler makes use of cooperative task switching to avoid stack/memory issues. The wireless communication stack includes MAC, network and socket layers between the physical and application layers. A buffer is shared by the MAC, network and socket layers to handle the data flow. Tasks with equal priority are grouped together and executed as single task, which leads to code size that is smaller than that of TinyOS. The average clock cycles required to execute an application are better than those required for TinyOS. The support for multithreading means that for complex tasks, PAVENET uses more RAM and ROM than TinyOS.

For WSN virtualization, PAVENET provides a thread-based programming model and uses C language. It is possible to program multithreaded applications with varying priority levels, but their execution will be sequential and not simultaneous because time-sliced execution is not provided. There is also no separation of application tasks from the OS. The main drawback of PAVENET is its lack of portability, although it is an interesting approach that shows how a better hardware design can lead to an efficient sensor OS.

The performance results of PAVENET show that it uses more RAM than TinyOS for sample applications. The execution times of sample applications is comparable to TinyOS. The task switching overhead is found to be 5 times less than MANTIS and comparable to TinyOS. Another aspect is the comparison of lines of codes needed to code sample applications in PAVENET and TinyOS. PAVENET uses twice as less as TinyOS (even more for complex applications).

Contiki [34] is by far one of the most popular systems for WSNs, and over the years has grown to become a leading platform for the IoT and low-powered embedded networked systems. It has a kernel based on an event-driven model, but preemptive multithreading is also provided as an option in the form of a library and exposed as an API for applications to call the necessary functions. Preemption is implemented using a timer interrupt. All threads have their own execution stack.

The concept of protothreads [35] was introduced to combine the concepts of event-driven and thread-based approaches. Protothreads borrows the block-wait approach of threads and combines it with the stack-less approach of events. The advantage of protothreads is that they have lower stack requirement than traditional threads and can be preempted, unlike events. Contiki makes it possible for applications and services to be dynamically uploaded/unloaded wirelessly on sensor nodes. This is made possible by incorporating relocation information in the application binary and later performing runtime relocation.

The OS is written in C language and can be ported to many hardware platforms. CPU multiplexing and an event handling mechanism are the two major functionalities provided by the kernel. The rest of the system-related functionalities are provided as system libraries that can be used by applications when needed. There is no hardware abstraction layer and applications can directly utilize the underlying hardware. Since the OS is event-driven, once an event handler is called, it can only be preempted by an interrupt—otherwise it must run to completion. A simple over-the-air protocol is used to dynamically load/unload applications in a WSN. Binary images of the new application code are sent to selected network nodes using point-to-point communication; the remaining sensor nodes receive the application code as broadcast from them. The current version of Contiki includes several features like full IP support [36], including IPv6 [37], CoAP [38], RPL, 6LowPAN, Cooja, a network simulator to test applications on emulated devices before actual deployment, the Coffee flash file system [39] for sensors that have external flash memory, and a command-line shell for debugging applications.

For node-level virtualization, Contiki is one of the better choices available. It supports multiple applications that are independent of the OS and run on top of it. Applications can be programmed in C language and updated/installed without reinstalling the whole OS. It provides a hybrid programming model. With protothreads, it is possible to create efficient multithreaded applications that share a common stack. Contiki supports many different hardware platforms.

The original Contiki paper used in this work does not provide any systematic performance results. However some insights regarding the performance were presents. For example, reprogramming of a sensor node with a new code (6 KB size) took around 30 seconds, whereas the reprogramming of 40 nodes with the same code took around 30 minutes. It is found that code size of similar applications in Contiki is larger than TinyOS but smaller than MANTIS.

TinyOS [30] is another notable effort to provide OS solution for sensor nodes. It is an application-specific, component-based OS based on two characteristics: being event-centric and offering a flexible platform for innovation. It is written in nesC, a dialect of C language, and has a component-based modular design using an event-driven programming model. Three main abstractions are used in TinyOS: commands, events and tasks. Commands are requests to perform a service, events are generated as responses when services are executed, and tasks are functions posted by commands or events for the TinyOS scheduler to execute at a later time. TinyOS components are sets of services, specified by the interfaces that are offered to applications. There are two type of components: modules and configurations. Modules are code snippets written in nesC for calling and implementing commands and events. Configurations connect components through their interfaces. Only components used by the applications are included in the final binary image.

The TOSThreads [40] library was introduced to combine the event-based approach with a thread-based approach, similar to the protothreads in Contiki. Event-based code runs in a kernel thread and user applications run in application threads. Application threads can only run when kernel thread becomes idle. Static optimizations are used during compilation to ensure the removal of any issues in the final code. The OS and the applications are bundled together at compile time in a single file. A component called Deluge [41] is used for over-the-air network-wide reprogramming. The new application code is distributed as composite binaries. Many protocols can be implemented as components. The current version of TinyOS is portable to many hardware platforms.

TinyOS is not the most suitable OS for WSN node-level virtualization. First of all, the programming mode is event-driven and it is often difficult to program event-driven applications. In the context of WSN virtualization, it may not be feasible to bundle applications with the OS at the time of deployment. New application tasks can only be installed by propagating the entire OS image over a virtual machine [42]. TinyOS also has tight coupling between the applications and the OS. The task scheduler in TinyOS is sequential (FIFO based) and executes tasks in run-to-completion mode, meaning a weak form of WSN virtualization.

The performance results of TinyOS highlight important features of the OS. For example, code optimization reduces code size of the programs as much as 60%. The timer component reduces CPU utilization by 38%. The interrupt and task switching also takes very less time as compared to SenSmart.

2) Virtual Machine-/Middleware-Based Solutions: Maté [42] is a tiny virtual machine that supports sequential execution and uses a stack-based binary code interpreter. It was designed to work on the early-generation, resource-constrained WSN nodes using TinyOS. The main purpose of Maté is to enable energy efficient code propagation in WSN with minimal overhead required to re-task sensors. To achieve this, application programs are broken into small code capsules and propagated throughout a WSN with a single command. Only predefined applications with predefined instruction sets are possible. There are fixed sets of instructions divided into three classes: basic, s-class and x-class. Basic instructions include arithmetic operations and the activation of sensors/LEDs, s-class instructions perform memory access, and x-class instructions perform branch operations. Up to eight user-defined instructions are also allowed. These user-defined instructions need to be fixed when Maté is installed and cannot be changed afterwards. Each program capsule contains up to 24 instructions. Larger programs consist of multiple capsules. The instructions in the capsules are executed in sequence until the halt instruction is reached. New application code is propagated in the network in the form of code capsules, using a viral code distribution scheme. Each capsule contains a version number which is used by a sensor node to determine if it needs to install new application code. Network-wide code propagation occurs when a sensor node forwards the code capsule to its local neighbors, which in turn forward it to their neighbors. Maté maintains two stacks, one for normal instructions and the other for instructions that control the program flow. When an instruction is under execution, a new instruction cannot be executed. This allows for simpler programming options. Maté incurs the cost of byte code interpretations before instructions can be executed.

Regarding node-level virtualization, Maté supports the sequential execution of tasks and tries to address the main drawback of the original TinyOS implementation. New application code can be injected without replacing the OS on a sensor node. However, applications are still tightly coupled. Maté is more suitable for simple event-driven networks where it is possible to define events and their outcomes. To end on a positive attribute, Maté does provide a simple mechanism to automatically reprogram a WSN using code capsules.

The performance results of Maté are collected by implanting an ad-hoc routing protocol which is also implemented in standard TinyOS release with Maté. The implementation of simple operations (such as AND, rand, sense, sendr) take more CPU cycles than native TinyOS, worst-case taking 33 times more CPU cycles and best case taking 1.03 times. A setup of 42 sensor nodes (in a grid pattern) is used to see the propagation of code using Maté. It is found that Maté takes little over 120 seconds to reprogram all sensor nodes with the new code. Overall Maté incurs overhead because its each instruction is executed as a TinyOS task.

VMSTAR [43] is a Java-based software framework for building application-specific virtual machines. It also allows for the updating of WSN applications as well as the OS itself. VMSTAR provides a rich programming interface that allows developers to develop new applications which can be portable to a variety of hardware platforms. VMSTAR generates compact code files rather than regular Java class files. It supports both the sequential and simultaneous execution of thread-based applications. The framework is comprised of three

parts: a component language called BOTS [44], a composition tool and an updating mechanism. The component language is used to specify software systems. The composition tool selects/composes the required components and determines the dependencies between them to satisfy specific constraints.

The updating mechanism uses an incremental update technique [45] to take actual coding changes rather than structural changes into account in the program code file like change in number of lines. For simple applications, sequential thread execution is supported, but for complex applications requiring input from external events, two event-based programming models are defined. One is the select model, in which an application subscribes to an event, acquires the corresponding event handle and executes it when the event occurs. In the case of multiple events, the respective handling methods are executed sequentially. The second model is known as action listener, in which applications define event handlers by extending the default handler class from the library—they do not register for events. When an event occurs, the registered callback method is invoked. The action listener model allows for the simultaneous execution of threads, but in the paper only the select model is implemented. A base station is used as a repository for application code and as an orchestrator for deployment and update purposes. A native interface is also provided to allow access to the underlying resources of a MICA platform.

For node-level virtualization, VMSTAR does support the concurrent execution of multi-threaded application tasks but the implementation presented only supports single-threaded Java applications. The programming model is thread-based and applications can be coded in Java language, making it easier for developers. Concurrent events can be handled using action listeners. Although VMSTAR discusses the distinction between the user applications and the OS, for the implementation example both are tightly coupled.

The performance results of VMSTAR show that it performs better than Maté but not so well against native TinyOS. For example, its memory consumption is almost double as compared to TinyOS. The same is true for CPU utilization, where VMSTAR sits between TinyOS and Maté.

Squawk [46] is a small Java virtual machine that runs on sensor hardware. Compared to VMSTAR, Squawk does not require an operating system to run; it provides the required functionalities by itself. These include interrupt handling, networking functions, resource management, support for the migration of applications from one SunSpot to another and an authentication mechanism for deployed applications. Applications in Squawk are represented and treated as objects. Since multiple, isolated objects can reside in a virtual machine, concurrent applications can be executed easily. Squawk VM runs on a specific device platform, Sun Small Programmable Object Technology (SunSpot) which has more processing, memory and storage capability than MICA /MICAZ and other WSN platforms. Squawk VM can use many standard Java features, such as garbage collection, exception handling, pointer safety, and thread library. It is written in Java, in compliance with J2ME CLDC [47]. The device drivers and the MAC layer are also written in Java. Squawk VM supports split VM architecture, where the class file loading is performed on a desktop machine to generate its representation file. The representation file is then deployed and executed on SunSpots. The size of these files is much less than standard Java class files. Green threads are used to emulate multi-threaded environments. The threads are managed, executed and scheduled in user space. An application's status, including its temporary state, can be serialized to a stream for storage. When another Squawk VM, on another SunSpot, reads that stream it can effectively reconstitute the application along with its complete state information. This allows for live-migration of applications from one SunSpot to another. This is quite useful in situations when a SunSpot device is about to run out of battery power.

For node-level virtualization, Squawk VM takes quite a different approach than its competitors. A robust and efficient application isolation mechanism is provided, which allows multiple applications to be represented and treated as Java objects. These objects are instance of the Isolate class and can be started, paused and resumed using available methods. Applications can have multiple threads which are managed by the JVM. The programming model is thread-based and applications can be coded in J2ME. There is also an option for Over-The-Air (OTA) programming which can be used to load, unload, stop and migrate applications on SunSpots.

The performance results of Squawk are presented using some benchmark suits and a math application to measure integer and long computation. For memory footprint, Squawk is compared with KVM for CLDC which shows that Squawk VM with debugging support uses less memory than KVM equivalent. The benchmark suits for Squawk and KVM were run of different sets of ARM platforms with different CPU and memory sizes. The KVM ran on better hardware and hence exhibited better results than Squawk VM. The suits files of applications generated in Squawk have around 37% less size than standard java class files and JAR files.

Agilla [48] is a mobile agent-based middleware that runs on top of TinyOS and uses a VM engine to sequentially execute multiple applications in a round-robin fashion. It uses a mobile agent and tuple-space programming models. The middleware is designed to support self-adaptive applications in WSNs. Application programs are coded as mobile agents that can migrate themselves to other sensor nodes in response to changes in the network or in the physical phenomenon that is being monitored. Each sensor node can run several autonomous mobile agents. These mobile agents may perform a strong migration, i.e., transfer application code and its state to another sensor. Weak migration only transfers application code, which means that at its new destination, a migrated mobile agent will restart the application. Agents are injected in the WSN from a base station and propagated one hop at a time. Each mobile agent arrives at a new destination, starts its execution and then migrates to the next-hop sensor node. This process can take quite some time to propagate a new application in the WSN. Each sensor node has a tuple space and a local memory. In a tuple space, data is accessed using pattern-matching techniques. This approach allows mobile agents to be oblivious of each other's memory addresses. Mobile agents have a stack space, a heap and three registers, which are used to store ID of the agent, program

code and condition code. Every agent, including the clones, has a unique ID. The program code register holds the address of the next instruction and the condition code register holds the execution status.

For node-level virtualization, Agilla relies on TinyOS to provide concurrency, and thus mobile agents are executed in a round-robin fashion. However, this is an OS issue, since a multithreaded OS can execute mobile agents in parallel allowing better concurrency. Mobile agents work independently of the TinyOS. The use of tuple-space and locally-stored agent states allows for quick migration, but still much work is left to the programmers to deal with issues such as stalled migration. In a highly dynamic WSN where applications utilize sensor nodes on the fly, such as the IoT, the migration of agents might lead to performance issues. The programming language of Agilla is another difficulty, as the agents are programmed in low-level assembly-like language.

A test-bed of 25 sensor nodes is used to gather the performance results. Agent migration is evaluated by varying number of hops between source and destination sensor nodes. The migration is 99% successful for up to 3 hops but after that it starts decreasing. Also more hops mean more latency, a 5-hop migration can take more than 1.1 second. The latency experienced for remote operations is under 300 ms.

The authors in [49] present an integrated system, UMADE, to promote the utilization of a deployed WSN among multiple contending applications. The main contribution of UMADE is a mechanism to allocate sensor nodes to improve overall Quality of Monitoring (QoM) for the applications. UMADE is implemented on TelosB motes and uses Agilla VM on top of TinyOS. The proposed systems consist of several components such as, specification of QoM attributes, application deployment and relocation of applications to deal with the network changes, as well as QoM-aware application allocation algorithm. QoM attributes are specified by variance reduction and detection probability attributes. A variance reduction QoM attribute exploits the correlation of sensor readings using probabilistic methods to predict sensor readings. For the detection probability QoM attribute, a stochastic model is used to find the probability of an event's detection by a group of sensor nodes. It is not clear from the paper whether QoM attributes can only be specified before the deployment of UMADE or if it is an evolving process. A simple greedy heuristic is used in a QoM-aware application allocation algorithm to maximize the overall WSN utility. Applications are deployed using an application allocation engine and an application deployment engine. The allocation engine runs in a base station and uses an allocation algorithm to find the suitable sensor nodes for an application. The deployment engine, present in both the base station and the sensor node, is used to wirelessly send a sensor application to the selected sensor nodes. The applications run concurrently in the Agilla VM. Both preemptive and non-preemptive allocation is used to deal with network dynamics and sensor node failures. In preemptive allocation existing applications are relocated to new sensor nodes to increase the overall utility, whereas in non-preemptive allocation no application is relocated to new sensor nodes. The base station side code is written in Java and the sensor node code is written in nesC.

UMADE uses Agilla VM for node-level virtualization. Agilla VM is extended to provide dynamic memory management for concurrent applications. UMADE has event-driven programming model and uses nesC language to code application tasks.

Application specific results are presented in the paper (i.e., applications that are implemented for evaluation purposes). For example, an increase in weight of a temperature monitoring application resulted in increase in its utility by 60%. The time to execute multiple application over a set of nodes increases linearly. Since UMADE uses Agilla over TinyOS its performance is highly dependent on those two solutions.

A macro-programming framework, Nano-CF, for the in-network programming and execution of multiple applications over a deployed WSN is presented in [50]. Nano-CF runs over the Nano-RK operating system [51] and allows several applications to utilize a common WSN infrastructure. Using Rate-Harmonized Scheduling (RHS) [52], Nano-CF realizes the coordinated delivery of data packets from multiple application tasks that run on sensor nodes. RHS also allows for data aggregation and ensures that small data packets are combined together before being sent to their respective applications. Nano-CF is a three-layer architecture consisting of a Coordinated Programming Environment (CPE) layer, an integration layer and a runtime layer. The CPE layer is present at the user/programmer side and allows them to write application programs in the Nano-Coordination Language (Nano-CL). Nano-CL is descriptive language with a C-like syntax. Its programs have two sections: service descriptor and job descriptor. The service descriptor section has tasks that are executed by the sensor nodes, as services. The job descriptor section has multiple services along with a set of nodes which will execute them. The programmer has to specify the timing and the periodic rate at which the services (tasks) will be executed at each sensor node. The program code is parsed to byte-code and sent to the sensor nodes by a dispatcher module in the CPE layer. The integration layer is responsible for handling the data and control packets. It consists of a sender module in the gateway and a receiver module in the sensor nodes to deliver the application task in byte-code. The runtime layer resides in each sensor node and consists of a code interpreter module which translates the received task byte-code for the underlying Nano-RK OS. It also provides routing functionality using DSR protocol. A data aggregation module collects aggregated data from the sensor nodes and sends it to the user applications using RHS. The proposed architecture is evaluated using a university campus multi-application sensing test-bed called sensor Andrew [53].

Nano-CF makes several contributions to node-level virtualization. It allows independent application developers to write application tasks for a common WSN infrastructure. Each application task runs independently and is not coupled with the sensor OS. The proposed framework is suitable for data collection applications and for sensor nodes that have multiple on-board sensors. The programming model is event-driven and applications are programmed using their descriptive language, Nano-CL.

The performance results of the solution cover the energy and overhead of code interpreter. Using RHS allows energy savings especially using multiple applications since packets are aggregating first and then transmitted. However, the packet size

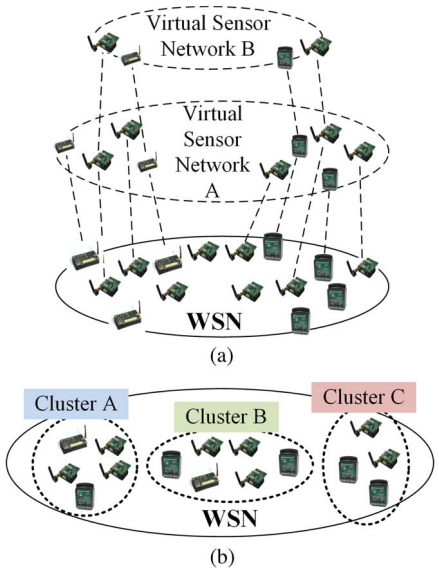


Fig. 4. Network-level virtualization solutions. (a) Virtual network-based solutions. (b) Cluster-based solutions.

has an impact on this because bigger packets means they cannot be aggregated due to size issues. When code interpreter is used, the extra-overhead is around 55%.

B. Network-Level Virtualization

We group the network-level virtualization approaches under two umbrellas: virtual network/overlay-based solutions and cluster-based solutions. Virtual network/overlay-based solutions utilize the concept of virtual networks and application overlays to achieve network-level virtualization. Virtual network/overlay are logical networks created on top of physical network(s). In cluster-based solutions, the nodes in a physical network are grouped to work together in connected groups, i.e., clusters. Unlike virtual network/overlays, clustering is more like the physical partitioning of the network where one part of the network is used to one application and another part is used by a different application. Nodes inside a cluster have specific roles, such as cluster-head and cluster-member. Typically cluster-based solutions in WSNs are used to monitor dynamic events.

Fig. 4 shows the network-level virtualization types while Table II illustrates the characteristics of the existing work dealing with node-level virtualization.

1) Virtual Network/Overlay-Based Solutions: The work in [9] uses overlays to create application-specific virtual networks on top of the deployed WSN. The overlay is used to allow data exchange between sensor nodes in different administrative domains. This work is more suitable for situations where it is difficult to bundle applications during the deployment of a WSN. A three-layer architecture is presented to allow multiple end-user applications to utilize sensor nodes concurrently. The bottom layer has new-generation sensor nodes like Java SunSpots, as well as older and less capable ones. To allow older and less capable sensor nodes to participate in overlays, another entity called Gates-to-Overlay (GTO) nodes is incorporated.

TABLE II
CHARACTERISTICS OF NETWORK-LEVEL VIRTUALIZATION SOLUTIONS

Solution (year)	Network Formation Mechanism	Protocols or Algorithms used	Results in the Paper are based on
Khan et al. (2013)	Application Overlays	JXTA	Not discussed
IoT-VN (2012)	Virtual Links	AODV	Implementation
MENO (2011)	Virtual Links	Not discussed	Not discussed
Taynan et al. (2008)	Virtual Network using Embedded Agents	Coverage Configuration Protocol (CCP) and Interpolation-based Redundancy Identification and Sensor Hibernation (IRISH)	Simulation
Jayasumana et al. (2007)	Virtual Links	Not discussed	Not discussed
Dilum et al. (2008)	Cluster Tree	Hop-ahead Hierarchical Clustering (HHC)	Simulation
Han et al. (2008)	Cluster Tree	PHenomena AwaRE clustering in wireless sensor networks (PHRE) and DRAGON	Implementation

The functionality of these GTO nodes can be implemented in gateways and sink nodes, as well as more powerful sensor nodes. The middle layer abstracts the simultaneous tasks executed by the physical sensors as virtual sensors. This is the basic assumption of the work, that the sensor nodes are capable of executing multiple application tasks concurrently. The top layer consists of applications implemented as overlays. These independent applications utilize the data sent by their respective tasks running on the sensor nodes. Each application has an independent overlay with virtual sensors as members of that overlay. This logical grouping allows data exchange even when sensors are physically located in different administrative domains. The architecture has separate paths for data and control messages. A fire monitoring scenario is used as an example, in which the sensor nodes in private homes are used to monitor the progress of fire eruption using a fire contour algorithm. Since sensor nodes are in private homes they cannot send data to each other directly. An overlay network is created to facilitate such data exchange and execute the fire contour algorithm. The authors assume the prior publication of sensor nodes to a registry which the end-user applications use to select the required sensors. The paper does not provide any implementation details. However, certain protocols are suggested for data, control interfaces and for overlays.

For network-level virtualization this work makes use of application-specific overlays to provide a robust and efficient mechanism for sensors to communicate. There have been some efforts to utilize DHT overlays in WSNs e.g., [54]–[57]. Each sensor can be part of several overlays at the same time and can execute their tasks. In the absence of any implementation details, it is difficult to determine the effectiveness of this solution, but it is quite relevant to IoT, where WSNs will be utilized by different users to provide new applications and services that were not envisioned during their initial deployment. Even geographically dispersed WSNs can be combined to provide data for new applications.

No performance results are presented in this work.

The work in [58] discusses the "Managed Ecosystems of Networked Objects" (MENO) concept, with its broader scope to connect sensor nodes as well as other IP-smart objects to the Internet for end-to-end communication without the use of traditional gateway-based approaches. The idea behind MENO is to create a virtual network on top of physical networks and thereby allow communication with different types of devices, including sensor nodes. Within each virtual network, end-to-end communication is possible using different protocols. Once end-to-end communication is enabled, it becomes possible for application developers to write new applications that utilize sensors, actuators and other devices. This work is still at the conceptual level, without any implementation details or results. It appears to be on track to use a clean-slate approach to integrate the physical world with the Internet in a seamless way. Some motivational scenarios are presented to make a case for integrating WSNs to the Internet.

The concept utilized by MENO is used to develop the Internet of Things Virtual Network (IoT-VN) [59]. That study presents some implementation details by applying the concept of the IoT-VN to constrained and non-constrained environments. For constrained environments, the IDRA framework [60] is used to implement neighbor detection and a tunneling mechanism to create virtual links between the members of the virtual network. For non-constrained environments, the Click Router [61] is used, which is a C++ based framework capable of realizing network packet processing functionality. Routing the data over virtual links is accomplished by means of the AODV protocol. They have extended the AODV header to include IoT-VN ID header and a network header. A simple ping application implements basic request and reply messages to demonstrate data exchange inside a virtual network.

For network-level virtualization, the work in [58] and [59] uses the concept of virtual links built over either layer 3 or layer 2 in traditional networks, and over IEEE 802.15.4 in WSNs. Not much detail about the actual protocols is provided, but the researchers do mention some motivational scenarios to open up WSN deployments and connect them to the Internet. Overall, the focus here is on connecting different devices (resource-constrained and non-resource constrained) together and allowing end-to-end communication for the deployment of new applications and services.

The work in [58] does not provide any performance results, however [59] presents early results using a simple two sensor test-bed setup. Round trip times of a ping command are shown which was sent from one sensor to another. Overall the results do not give much insight in to the solution.

An embedded agent-based approach is presented in [62] to create and maintain Virtual Sensor Networks (VSNs). This agent-based solution is built on top of Java SunSpot devices, as they offer Java programming support and are easier to program. The authors first provide an analysis of the layered approach normally used to create and maintain a VSN. In this approach a new VSN layer is introduced to create and maintain a VSN, but it is not flexible when the sensor nodes' sleep and wake patterns are taken into account. A sensor node that is part of more than one VSN at a time cannot sleep abruptly without first coordinating with other sensor nodes to inform them about its unavailability. Since the layers in sensor nodes are tightly coupled and cannot be changed without affecting the other layers, an agent-based solution is proposed in this work. Agent Factory Micro Edition (AFME) [63] library is used to create agents. Each agent resides on a sensor node and is responsible for creating and maintaining a VSN, as well as for communicating with the agents working for the same VSN on other sensor nodes. These agents can communicate with each other to optimize performance. AFME allows communication between agents for easy message exchange. AFME also allows the migration and cloning of agents in the network, which makes it easy for new sensor nodes to join a VSN. Using the agent-based approach has obvious benefits, not least because a sleep broker can make intelligent decision about the sleep and wake duration of sensor nodes.

For network-level virtualization the work in [62] considers independent VSNs created over a WSN for different applications. To create such VSNs, mobile agents create a virtual topology linking sensor nodes together for an application. Although the agents are implemented using AFME, there are no details about VSN formation and its operation.

Interestingly the work does not provide any performance results of the agent-based approach instead it present simulated results of layered approach showing their obvious drawbacks.

Pioneering work regarding network-level virtualization was first presented in [19] and extended in [64] and [65]. In [19], a subset of WSN nodes dynamically forms a VSN. Applications with attributes or situations such as being geographically dispersed, using heterogeneous WSN nodes with different capabilities and that monitor dynamic phenomenon are particularly suited to take advantage of VSNs. Each independent subset executing an application is a VSN. In this approach, it is clear that different applications can execute sequentially, due to the dynamic VSN formation by different node subsets. However, the authors do not give any information about how these applications might eventually be executed simultaneously. Two illustrative applications are presented. One is a geographically overlapped application which works in scenarios where heterogeneous WSN nodes are deployed to monitor two different events spread over a large area. Each WSN needs to be deployed without using resource sharing even in those areas where there is no event of interest, to provide communication and routing. With resource sharing however, other WSNs can help, resulting in a more efficient use of resources.

The second application illustrates the concept of monitoring a dynamic event with a subset of WSN nodes. This subset can expand or reduce depending on the dynamics of the event. The work discusses the management issues of these VSNs and describes functions to create VSNs. WSN nodes that are not part of any subset help in the overall WSN operation, with data routing for example, or remain asleep to conserve energy.

For network-level virtualization the authors in [19] present the basic motivation to create VSNs. Example applications are discussed. However, the paper presents high-level details and does not include any technical details, e.g. how to realize these VSNs. The paper provides the basic concept of multiple applications sharing a WSN and using multiple WSNs for new applications without additional deployments.

No performance results are presented in this work.

2) Cluster-Based Solutions: A self-organizing tree-based solution is presented in [64] to facilitate the creation, operation and maintenance of VSNs. When an event has been detected, a dynamic cluster tree is formed, ensuring that nodes will join a VSN to monitor the event in a reactive manner. In this approach the sequential execution of applications is possible, since VSNs are formed dynamically, but it is not clear if (or how) it is supported by the WSN nodes. This approach uses cluster heads and child cluster heads inside VSNs to carry out different functions. This structural organization provides logical connectivity among WSN nodes and ensures that two different notifications of the same event are detected and treated as one; meaning that no event in the deployed WSN remains unknown. Once an event is detected, a dynamic cluster tree is formed by exchanging VSN formation messages.

VSNs provides unicast, broadcast and multicast communication. For unicast communication, a hierarchical addressing scheme like DNS is used while broadcast and multicast communication use a list. This list is used by each cluster head to keep track of the child cluster heads it serves. A new hierarchical clustering algorithm is proposed to create VSNs. A simulation-based performance analysis of the proposed algorithm is presented using a custom-built simulator in C language. However, advanced VSN functions like the merging and splitting of VSNs are not implemented.

A cluster tree mechanism is used to group the sensor nodes that work for an application, as a way to realize network-level virtualization. This work is an extension of the work in [19]. Dynamic trees are formed and communication between the sensor nodes is also supported. There is no discussion about the actual implementation of the proposed scheme.

For performance results a discrete-event simulator is used. Three scenarios are implemented to detect events in different regions and use sensor nodes to monitor them. The results show a linear increase in number of hops similar to the increase in sensor nodes monitoring the event. When an event occurs, with source and destination node in the same region, more unicast messages are exchanged but these messages are not affected by the network size. On the other hand, when an event occurs in another region more multicast messages are exchanged and are affected by network size.

A proof-of-concept study that monitors an underground plume is presented in [65]. It is based on a single application, and so it is difficult to find a link with sequential or simultaneous execution. The authors also discuss a phenomena-aware clustering algorithm to create and maintain VSNs. Using this algorithm, clusters are comprised of groups of WSN nodes that are close to dynamic phenomenon and report on it frequently throughout their lifetimes. With these reports, the algorithm is able to select those WSN nodes which are relevant for clusters and that are close to the dynamic phenomenon, allowing less-relevant WSN nodes to save their energy for other applications. This technique considerably reduces the required data reporting since only relevant data is sent. As the deployed WSN is event-based and not always on, sudden bursts of data are avoided whenever an event of interest occurs. The algorithm is also resilient to WSN node and link failures. To adapt to the dynamics of an event, i.e., a merger or a split, another algorithm, called DRAGON, is presented. When an event is detected, DRAGON ensures its location is found and used as a reference point to track its movement. Sensor readings and the relative positions of WSN nodes are then used to make decisions about whether two events should logically remain distinct or be merged into a single event.

For network-level virtualization this work is based on [19] and [64]. The proof-of-concept prototype is used to demonstrate the viability of the concepts presented in earlier papers, however only one application is demonstrated.

There are not much performance results of the prototype except that the sensors were able to track a plume similar to the conductivity probes.

C. Hybrid Solution

Hybrid solutions combine both node- and network-level virtualization mechanisms. We group the Hybrid solutions under three types: middleware and cluster-based solutions, middleware and virtual network/overlay-based solutions and virtual machine and dynamic grouping-based solutions.

In middleware and cluster-based solutions, a middleware handles node-level virtualization, while network-level virtualization is achieved by grouping sensor nodes into clusters. In middleware and virtual network/overlay-based solutions a middleware handles node-level virtualization while network-level virtualization is achieved using virtual network/overlays. In virtual machine and dynamic grouping-based solutions, node-level virtualization is achieved using a virtual machine, and a tailored, sensor node grouping scheme is used for network-level virtualization.

Fig. 5 shows the hybrid virtualization solution while Table III shows the characteristics of hybrid solutions.

1) Middleware and Cluster-Based Solutions: In [66], a middleware solution, Sensomax, for Java SunSpot [67] devices is presented. Sensomax follows a components-based approach and provides several operational paradigms such as data-driven, event driven, time-driven and query-driven, to offer more flexibility. The main contributions of Sensomax are support for multi-tasking, dynamic task modification and re-programming at runtime. At node-level, user applications are coded as

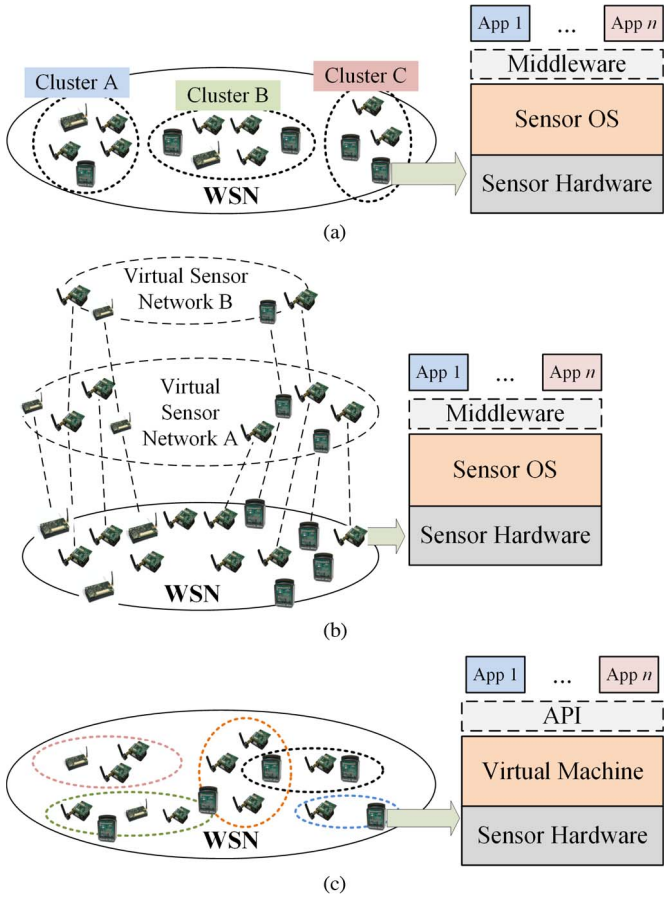


Fig. 5. Hybrid virtualization solutions. (a) Middleware and cluster-based solutions. (b) Middleware and virtual network-based solutions. (c) Virtual machine and dynamic grouping-based solutions.

application-specific agents. Concurrency is implemented using a main Monolithic Kernel, abstracting the sensor resources. Applications act as Microkernels running atop the Monolithic Kernel and access underlying resources in a uniform way. When an application starts its execution in a sensor node, its corresponding agent is loaded to an execution space and queued for execution. A resource-algorithm is used for allocating resources to multiple agents in the execution space. However, no details of such allocation algorithms are discussed. Application agents can be data-driven, event-drive, time-driven, query-driven or hybrid models.

At the network level, the deployed WSN is divided into multiple clusters consisting of sensor nodes. Each cluster is dedicated to a single or multiple applications and treated as a single entity by the application programmers. The applications can span over multiple clusters by running application-specific agents in each cluster. Each cluster consists of a sensor node acting as the cluster-head and several sensor nodes acting as cluster members. Sensor nodes can have dual roles, i.e., a sensor node can act as cluster-head for an application while at the same time it can be a cluster member for a different application. Such roles depend on the application agents residing in a sensor node. The agent-based approach is used for network-level communication in Sensomax. The global agents enable different network entities to communicate with each other. The local agents are used for intra-cluster communication, allowing the cluster-heads to communicate with their cluster-members and vice-versa. The system agents are used by the base-station to send configuration instructions to cluster members via cluster heads. The system agents are used to reprogram or update sensor nodes on the fly. The WSN resources are divided into three main classes: global, local and system resources. Global resources include sensors, actuators and processes that are shared among different network entities. Local resources include resources found inside a cluster and can only be shared between members of that particular cluster. System resources include items such as system properties where resource states are defined. A one-hop broadcasting of agents is used to propagate application-specific agents in the WSN.

For node-level virtualization, Sensomax uses Java SunSpot devices and exploits their ability to run concurrent application tasks. Each user application is programmed as an agent, and multiple agents can reside on a single sensor node. Agents are submitted via a base station and propagated into the WSN using a one-hop broadcast. The network-level virtualization uses the clusters concept. The WSN is divided into multiple clusters, each with its own cluster head. Different types of communication modes are provided to enable communication between different network entities.

The performance results are collected by means of a test-bed consisting of 12 sensor nodes and a simulator. The processing time of each agent is found to be around 200 ms when the sensor node is executing 30 concurrent applications. The simulation results follow the same trend. The sample applications report temperature and light level with various conditions. The dynamic update processing time is under 100 ms for the same number of applications.

The work in [68] presents a multi-set architectural model to allow the execution of multiple applications over a deployed WSN. This work is based on the concept of agents, similar to Agilla. The agents are not application-specific, instead they are used to control the node- and network-level functionality. The overall design goal is the ability to run multiple applications in a pre-defined execution order and to be able to adjust their functional parameters. A configuration agent (C-Agent) is used to modify the functional parameters of an application running on a sensor node, e.g., to change its sampling interval. The C-Agent is first propagated in the WSN from the base station to the cluster-heads and then from cluster-heads to the sensor nodes in their clusters. Before the deployment of a WSN, the applications and their order of execution are defined. This step limits flexibility, as new applications cannot simultaneously use the deployed WSN. At node-level, TinyOS is used to provide concurrent execution of application tasks on a sensor node using a middleware that runs on top of TinyOS. The solution inherits the drawbacks of TinyOS; making applications to be executed in their predefined order.

At the network-level, the scoping building block concept [69] is used to divide a WSN into subsets. Within these subsets, nodes can be grouped as clusters according to the application requirements. Each subset is dedicated to execute only one application, hence a WSN with n subsets will execute n number of applications. The role of cluster-head is performed by powerful

TABLE III
CHARACTERISTICS OF HYBRID SOLUTIONS

Solution (year)	Programming Model	Programming Language	Separation between OS & Application Tasks	Real-time Applications	Protocols or Algorithms used	Network Formation Mechanism	Results in the Paper are based on
Sensormax (2013)	Thread-based	Java	Yes	No	Market-based resource allocation algorithms	Clusters	Simulation & Implementation
ShenShare (2012)	Event-driven	nesC	Yes	Yes	Collection Tree Protocol (CTP)	Network-level Overlays	Implementation
VITRO (2012)	Not discussed	Not discussed	Yes	No	Not discussed	Not discussed	Not discussed
Majeed et al., (2010)	Event-driven	nesC	No	No	Not discussed	Clusters	Not discussed
Melete (2006)	Event-driven	TinyScript	No	Yes	Trickle	Connected Graph	Simulations & Implementation

sensor nodes, so there is no selection of cluster-heads on the fly. When the WSN is deployed initially, only one application begins its execution, according to a pre-defined sequence. The sensors in other subsets sleep to conserve their energy until it is their turn to execute their application. A switching agent (S-Agent) is used to switch from one application to another by putting awake sensor nodes into sleep mode and vice-versa. There is no information about how S-Agent is propagated in the network.

For node-level virtualization, the solution works similar to the TinyOS and provides a weak form of virtualization. Predefining applications and their execution sequences does not make this solution very attractive. For network level virtualization, the WSN is divided into subsets that have multiple clusters. At any given time the sensor nodes in one subset are active while others sleep to save their energy.

No performance results are presented in this work.

2) Middleware and Virtual Network/Overlay-Based Solutions: The authors in [70] discuss SenShare, a platform to execute multiple applications over a WSN. This is the first significant effort to tackle the issue of allowing open access WSN deployments running multiple applications concurrently. Two roles, those of WSN infrastructure owners and application developers, are considered. This separation opens up the possibilities for new business models, innovative applications, improved utilization of WSN resources, and flexibility, along with cost benefits. At node-level a hardware abstraction layer (HAL) and a node runtime layer is used in each sensor node to support multiple applications. Each application is a TinyOS program which runs on top of a multi-tasking OS that allows the simultaneous execution of multiple application tasks. The HAL is shared by each application and is used to break the tight coupling between TinyOS applications and the sensor hardware and to allow shared access to the sensor hardware. Each application contains virtual hardware controllers (e.g., access to LEDs, sensors, timers and network I/O) that are linked to all TinyOS application at compile time. When an application requires access to, e.g., a sensor, the corresponding virtual hardware controller passes the request to a runtime layer between the applications and the multi-tasking OS. The runtime layer is OS-specific and all of the TinyOS applications use it to access the sensor hardware. It runs as a separate process inside every sensor node and mediates between the applications and the sensor hardware. The sensor I/O and network I/O are two components in the runtime layer that allow managed access to sensing components and to the network interface, respectively. This access is allowed asynchronously to multiple applications. Each application in SenShare, has a unique ID which is used to manage it. To deploy an application, SQL-like commands are used to select the target nodes according to the application's requirement. Afterwards the application's binary code is sent to the selected nodes using a modified version of the Deluge protocol [71]. Once the application is up and running, the virtual topology is formed to provide isolation from other data/control traffic. The WSN is globally synchronized using the TPSN protocol [72].

At the network level, a network-level overlay is created to group WSN nodes that execute similar application, using the Collection Tree Protocol (CTP) [73]. Physically scattered groups executing similar applications can be joined into a single overlay network. CTP is also used to route data and control messages in the WSN. To provide isolation between the traffic from multiple applications, each application packet is modified to include the application ID along with sequence number, origin and destination addresses. The runtime layer attaches and removes this information at the source and destination nodes, respectively.

An application could be executed by physically scattered sensor nodes. Linking these scattered sensor nodes (clusters) into a single virtual connected network requires an overlay formation protocol that utilizes the underlying CTP topology to connect clusters together in a virtual connected network. The protocol works by making each sensor node route its packets to the closest cluster.

For node-level virtualization, SenShare implements application tasks as TinyOS programs over a multi-tasking OS. The

programming model is similar to TinyOS. Incorporating virtual hardware controllers with the applications makes the solution less flexible, as developers need to be aware of the type of hardware each sensor node has. The runtime layer between the OS and the applications does not expose the sensor hardware to the developers, so they cannot write applications on the fly. For network-level virtualization, SenShare uses the concept of overlays and uses CTP protocol to create independent overlays for applications.

The performance results of this work cover the application isolation penalty and overlay management. With more concurrent applications in a sensor node, it is observed that sampling rate decreases by 28% as compared to a single application sampling the same phenomenon. The CPU utilization also increases linearly and has less impact on the SenShare runtime. The same is observed for memory usage. The extra overlay traffic is found to be decreasing over the period of time to around 10% of the network traffic.

The work in [10] discusses the node- and network-level virtualization of sensor nodes in the context of the VITRO project. The goals of this work are *i)* to design a middleware to act as a bridge between applications and the sensor nodes, and *ii)* to design advanced sensor node architecture. Node-level virtualization is achieved by instantiating various instances of routing and of MAC layers. There is a Node Virtualization Manager (NVM) inside every sensor node which is responsible for managing the available resources and fulfilling the requests to utilize those resources [74]. NVM interacts with each layer to ensure the optimal, secure and energy-efficient utilization of sensor nodes. Each sensor node has a middleware which is responsible for its discovery and the services it provides. This middleware sits on top of the network layer. The network layer uses routing protocols that can support multiple routing instances. A trust-aware routing protocol [75] is used to route the data, and delay-tolerant network mechanisms are suggested to counter the connectivity issues. For each application, a newly configured MAC layer is instantiated.

A reference architecture is presented at the network level, consisting of several autonomous WSN domains. Each of these domains is connected to VITRO service providers through a gateway node. The gateway node plays a major role in providing network-level virtualization. It consists of modules that help in the creation and management of VSNs. The gateway node uses several registries to create and manage a VSN. In VITRO, only gateway nodes can be part of the VSN, which can be realized by creating a routing link between them using protocols such as RPL. Individual sensor nodes can only be part of the VSN, on their own, if they support the functionalities of the gateway node, otherwise they can only join a VSN with the help of a gateway node. Details such as sensor selection and task dissemination are not discussed. A VSN manager is responsible for service negotiation, session establishment and monitoring. Functional architectures of gateway nodes and advanced sensor nodes are also presented, along with the details of the interfaces between system components. No implementation details are discussed and no protocol recommendations are given for interfaces or functions such as service registration or service negotiation.

For node-level virtualization, VITRO relies on advanced sensor nodes that enable the efficient utilization of resources and concurrent access. However, there is no discussion regarding the OS that will provide such functionalities, nor is there any information on the hardware platform in the paper. Most of the details are at the conceptual level; no technical details such as programming model, programming language, and OS are provided. For network-level virtualization, this work only connects already VSN-aware/legacy/proprietary WSNs through a gateway node. The mechanisms for creating a VSN-aware network are not discussed, nor is there any mention of protocols to be used.

No performance results are presented in this work.

3) Virtual Machine and Dynamic Grouping-Based Solution: Melete [18] provides both node- and network-level support for the concurrent execution of applications in WSNs. At the node-level, Melete supports simultaneous execution by enhancing Maté, supporting the interleaved execution of multiple applications on a single WSN node. Application code images are stored, each with its own dedicated execution space. Applications do not share variables with each other to ensure that an application failure does not affect other applications executing on the same WSN nodes. The number of concurrent applications that can be executed by WSN nodes depends on the available RAM; the implementation in the paper supports up to five applications. Melete uses an event-driven programming model. Another contribution of Melete is that it supports application task code dissemination. Task code dissemination has two main goals. One is to select the sensor nodes which are part of a group, and send new code to them. The second is to reactively send code to the sensor nodes that require it. Both goals allow the task code of the relevant sensor nodes to be sent while discouraging its unnecessary dissemination. Actual code forwarding is done region-wise using multi-hop communication.

At a network-level, Melete supports the dynamic grouping of deployed WSN nodes to execute multiple applications simultaneously. The supported network topology is a connected graph. It is possible for WSN nodes to be part of more than one logical group at a time. Each logical group is dedicated to a single application, and the implementation supports up to 16 groups coexisting in a WSN. A new application code is disseminated passively between members of the group using the above-mentioned design goals. All WSN nodes maintain the version information of the applications, and advertise it in the group, hence making WSN nodes aware of when to update their application codes. This saves energy by reducing unnecessary communications, but at a cost of the delay incurred. Sensor nodes in a logical group execute a single application at a time, hence each application cannot be influenced by the run-time error of another application. The paper presents extensive simulation-based as well as actual implementation results.

For node-level virtualization, Melete improves on Maté, but since application tasks have their own data and execution space, only a limited number of application tasks can run concurrently. The programming model is based on the event-driven approach of TinyOS. The application programs are written in TinyScript. A dynamic grouping scheme is provided for network-level

TABLE IV
SUMMARY OF THE STATE-OF-THE-ART

Solution	Requirements								
	Type	Node-level Virtualization	Network-level Virtualization	Application Priority	Platform Independent	Resource Discovery	Application to Resource-Constrained Nodes	Heterogeneity	Sensor Selection for Application Tasks
SenSmart [23]	OS-based	Yes	No	No	Yes	No	Yes	Yes	No
RIOT [25]	OS-based	Yes	No	Yes	Yes	Yes	Yes	Yes	No
PAVENET [33]	OS-based	Yes	No	Yes	No	No	Yes	No	No
SenSpire [28]	OS-based	Yes	No	Yes	Yes	No	Yes	Yes	No
Nano-CF [50]	VM-based	Yes	No	No	Yes	No	Yes	Yes	No
UMADE [49]	VM-based	Yes	No	No	No	No	Yes	No	Yes
Agilla [48]	VM-based	Yes	No	No	No	Yes	Yes	Yes	No
LiteOS [32]	OS-based	Yes	No	Yes	No	No	Yes	Yes	No
Squawk VM [46]	VM-based	Yes	No	Yes	Yes	No	No	No	No
VMSTAR [43]	VM-based	Yes	No	Yes	No	No	Yes	No	No
MANTIS [29]	OS-based	Yes	No	Yes	No	No	Yes	No	No
TinyOS [30]	OS-based	Yes	No	No	Yes	No	Yes	Yes	No
Contiki [34]	OS-based	Yes	No	Yes	Yes	No	Yes	Yes	No
Maté [42]	VM-based	Yes	No	No	No	No	Yes	Yes	No
Khan et al. [9]	Overlay-based	No	Yes	Yes	Yes	Offline publication	Yes	No	No
MENO [58]	VN-based	No	Yes	–	–	–	–	–	–

virtualization. By default, all sensor nodes are members of a parent group, with its code stored in them. How a sensor node will join a new group depends on the task code it is executing. The programmer needs to be aware of the many situations that may arise in the network and program the responses, and this approach is not flexible at all.

The performance results of Melete include mathematical analysis of the impact of parameters on the task code dissemination scheme. The code size and memory consumption of Melete was compared to Maté. The code size of Melete is bigger than Maté even when there was only one application. Similarly Melete exhibits higher memory consumption than Maté. The result pertaining to dynamic grouping show delays in the order of seconds for a motion tracking application in an office setting.

D. Summary

Table IV illustrates the evaluation of the existing work based on the requirements identified in section 2.4. We have found several capable node-level virtualization solutions. In the early-generation sensor nodes, the programming model of choice was event-driven, as it was simple to implement, but once its limitations were found, the thread-based approach was used to implement more complex and concurrent tasks in sensor nodes. Of all these works, TinyOS and Contiki have become extremely popular and have good community support. Contiki is now considered as a platform for the IoT [76] and has incorporated many innovative features over the last decade. RIOT [25] is a new work to design a capable OS to run C/C++ applications on heterogeneous sensor platforms.

For network-level virtualization, the early work used the concept of clusters but managing clusters itself is quite challenging. The majority of work on cluster-based solutions in WSNs is focused on improving routing, energy efficiency and security. We need solutions that facilitate the creation of application-specific clusters that adapt to the dynamics of the network and of the monitored events. Recently overlay solution are being used for network-level virtualization but it is still largely unexplored territory. We have works like [54] discussing, quite convincingly, that it is not '*mission impossible*' to use overlays in WSNs. Most recent research work has focused on providing hybrid solutions for WSN virtualization. A few recently-concluded research projects have addressed WSN virtualization, but their solutions are embryonic and multiple issues remain. For example, some solutions are platform dependent, others are theoretical and at conceptual level.

TABLE IV
(Continued). SUMMARY OF THE STATE-OF-THE-ART

Solution	Requirements								
	Type	Node-level Virtualization	Network-level Virtualization	Application Priority	Platform Independent	Resource Discovery	Application to Resource-Constrained Nodes	Heterogeneity	Sensor Selection for Application Tasks
IoT-VN [59]	VN-based	No	Yes	No	Yes	Yes	Yes	No	No
Taynan et al [62]	VN-based	No	Yes	No	No	No	No	No	No
Jayasumana et al [19]	VN-based	No	Yes	No	Yes	No	Yes	No	No
Dilum et al [64]	Cluster-based	No	Yes	No	Yes	No	Yes	No	Yes
Han et al [65]	Cluster-based	No	Yes	No	Yes	No	Yes	No	Yes
Sensomax [66]	Middleware- and cluster-based	Yes	Yes	No	No	No	No	No	No
SenShare [70]	Middleware- and overlay-based	Yes	Yes	Yes	No	Yes	No	No	Yes
VITRO [10]	Hypervisor- and VM-based	Yes	Yes	No	Yes	Yes	Yes	No	No
Majeed et al. [68]	Middleware- and cluster-based	Yes	Yes	Yes	No	No	Yes	No	Predetermined
Melete [18]	VM- and Dynamic grouping-based	Yes	Yes	No	No	Yes	Yes	No	Yes

V. WSN VIRTUALIZATION RESEARCH PROJECTS

In this section we introduce some relevant projects that envision the utilization of WSNs by multiple applications. Table V lists these projects and provides their summary based on the following characteristics.

1) Project Aim: Provides the holistic aim of the overall project. FRESnel and VITRO are the only two projects that are aimed directly at WSN virtualization. The remaining projects have more extended scopes, such as smart city realization, smart health in the context of IoT, or aim to provide a large-scale test bed for network research.

2) Project Scope: Indicates if a project is a part of academic or industrial research, or is being developed as a multi-partner effort. VITRO, Smart Santander, iCore and Butler are all European FP7 projects involving large consortiums of industrial, telecom and academic partners. FRESnel is a joint project between Cambridge and Oxford Universities, UK.

3) Virtualization Level: Indicates the type of WSN virtualization. FRESnel and VITRO are the two projects that aim to provide both node- and network-level virtualization. CitySense, iCore, Butler and ViSE do not explicitly address WSN virtualization, but they do consider the utilization of sensors by multiple applications.

4) Virtualization Type: The true realization of WSN virtualization does not involve any gateway node managing the virtualization-related tasks; instead, sensor nodes themselves handle such tasks. On the other hand the gateway-based virtualization solutions make WSNs act as capillary networks connected to the Internet or to other networks through a single node. It is important to mention that the presence of a gateway node for communication is difficult to rule out since sensor nodes may use sense and sleep mechanisms.

5) Network Devices: Another important characteristic of these projects is the type of devices they use in their work. CitySense, Butler and ViSE use high-end devices. While sensors are considered, they are usually connected to high-end PCs/nodes that compliment them for processing, data storage, power supply and connectivity. FRESnel and VITRO utilize the usual/normal sensor nodes, which is more relevant to WSN virtualization.

6) Evaluation Setup: All of the projects discussed here evaluate their contributions using real test bed setups; however the size of these setups varies considerably. For example, the Smart

TABLE V
WSN VIRTUALIZATION-RELATED PROJECTS

Project (Year)	Project Aim	Project Scope	Virtualization Level	Virtualization Type	Network Devices	Evaluation Setup
CitySense [77]	Provide city-wide test bed for distributed & networking research	Academic research	Network-level	Gateway-based virtualization	Embedded PCs with Linux acting as gateways	100+ PCs distributed over an urban area
FRESnel [78] (2010 - 2012)	Provide a federated WSN framework for multiple applications	Academic research	Node- and Network-level	Sensor node-based virtualization	iMote2 nodes using embedded Linux	35 iMote2 nodes distributed in an academic building
VITRO [74] (2010 - 2013)	Develop architectures, algorithms to provide VSNs.	Academic research + Industry	Node- and Network-level	Gateway-based virtualization	TelosB, IRIS, iSENSE, xbee, TmoteSkye, AdvanticSys kit	Simulations + 5 test bed setups by project partners
Smart Santander [79] (2010 - 2013)	Provide a city-wide IoT experimentation platform for smart city applications	Academic research + Industry	Network-level	Gateway-based virtualization	Sensor nodes, IoT devices, RFID tags, GPRS devices	About 20,000 sensors deployed in four European cities
iCore [80] (2011 - 2014)	Provide a cognitive framework consisting of virtual objects, composite virtual objects & business perspectives	Academic research + Industry	Abstract representation of sensors	Gateway-based virtualization	Sensors, ICT devices, everyday objects	Will utilize the Smart Santander test bed
Butler [81] (2011 - 2014)	Provide a secure, pervasive, energy-efficient & context-aware architecture	Academic research + Industry	Abstract representation of sensors	Gateway-based virtualization	Smart objects, mobile devices and smart servers	Several field-trials and proof-of-concepts are planned
ViSE [82] (2008 - 2011)	Provide public access to a WSN test bed using the GENI framework for multiple users	Academic research	Abstract representation of sensors	Gateway-based virtualization	High-end nodes running Linux and acting as gateway nodes	Three nodes deployed in a town near a forested area

Santander project will use around 20 000 nodes deployed over four European cities, providing a massive platform for research and evaluation purposes. This gigantic setup will also be used by the iCore project. In comparison, Fresnel's in-campus test bed has 35 nodes while ViSE test bed has only 3 nodes.

The ViSE and CitySense projects were not designed to provide solutions for WSN virtualization, but they do incorporate the important virtualization concept, i.e., to allow multiple applications to utilize the deployed WSN infrastructure. The Smart Santander, iCore and Butler projects are aimed to realize the IoT, and consider sensors and devices of different types. VITRO and FRESnel are focused on WSN virtualization, but VITRO provides gateway-based virtualization, which is not a true realization of WSN virtualization. The FRESnel project however, considers the true realization of WSN virtualization, but it provides platform-specific solutions. Overall it is clear that the idea of WSN virtualization is receiving considerable attention, not only from academic quarters but also from major industrial and telecom players.

VI. RESEARCH ISSUES

We identify some important research issues that need to be addressed to provide innovative WSN virtualization solutions.

1) Advanced Node-Level Virtualization: Node-level virtualization has attracted considerable attention from the research community. In many ways, it is provided as part of the sensor OS. Multi-threaded OSs and application-specific virtual machines (VM), working on top of an OS, can support the concurrent execution of application tasks. As the trend moves towards more powerful IP-WSNs, more efforts are required to virtualize the individual components of sensor nodes, such as MAC and routing layers. The VITRO project has put forth the concept [10], but there are no real implementations to date. PAVENET OS [33] takes advantage of capable hardware to design efficient OSs but is tied to a single platform. To exploit the recent advances in sensor hardware, a fresh approach like RIOT OS [25] can be taken to come up with new and general purpose solutions. Some new solutions provide separation between the sensor OS and the user application tasks but we still need functions like OTA installation/updating

of new user tasks without disturbing the existing ones. One possible solution to tackle this issue is to design an abstraction layer that works on top of sensor OS to provide application portability like in [83]. A modular-based approach will work much better since it will be applicable to heterogeneous OSs, programming languages and models.

2) Network-Level Virtualization: Not much work has been done in the area of network-level virtualization to support multiple applications over a deployed WSN, hence there is a tremendous opportunity to make valuable contributions. Overlay networks can provide an efficient solution as they are robust and can work efficiently without changes in the underlying network. Some solutions like those in [54], [56], and [57] do exist, but they are still embryonic in nature and do not consider the requirements of multiple applications utilizing a WSN concurrently. As multiple overlays may need to coexist, preventing them from interacting with each other in a harmful way remains a challenge. Cluster-based approaches have traditionally been used in WSN's for improving routing, energy-efficiency, management and security. Managing clusters in a virtualized WSN is not trivial, however, cluster-based solutions can be quite useful in scenarios where a deployed WSN is used to monitor dynamic events. These solutions can also be helpful in mobile WSNs, Robotic and Vehicular Ad hoc Networks.

3) Discovery and Publication: The discovery and publication of resources and services in WSN is already challenging, but it becomes more sophisticated in virtualized WSNs. For example, it will be interesting to find whether certain kind of relationships exist between physical and virtual sensors and whether they can be exploited to provide quick publication and discovery solutions. As virtual sensors are created on-demand and destroyed when no longer required, their publication and discovery needs to be efficient, robust, scalable and manageable. Discovery and publication of resources and services on the fly are very important functions, especially in the context of IoT. A P2P based architecture can be a solution like [84] that does not rely on any central mechanism to discover the services. However, no such solution exists for virtualized WSNs. Similarly a service recommendation system can be developed, for virtualized WSNs, which allows context-aware discovery of resources and services. Recent IETF service discovery protocols like CoAP resource discovery [85], [86] and DNS-SD [87] can be used to design efficient discovery and publication solutions in resource-constrained environments. Moreover, new algorithms that adapt to evolving WSN conditions and nodes' mobility or failures are required, to ensure service continuity.

4) Service Composition: Service composition using virtual sensor nodes is another important research challenge. In our view, future WSN deployments will involve multiple actors, such as WSN providers, virtual sensor providers, service providers, third-party application/services providers and end-user applications. A cloud-based approach could be a solution [88]. WSN resources could be offered as Infrastructure-as-a-Service (IaaS) and used by Platform-as-a-Service (PaaS) to offer services to end users. In this regard, existing projects like [79], [80], and [81] can be used for inspiration about end user services. Using semantics and ontologies to compose services based on application requirements and the capabilities of sensor nodes can provide improved solutions. It is also important to note that the service composition may also use existing or third-party services on the fly. Location and mapping services are typical examples of such services.

5) Sensor Node Selection and Task Assignment: The issues of sensor selection and task assignment are very much related to each other. Selecting the right set of sensor nodes according to the temporal and spatial requirements of applications is crucial [21] to improving the overall Quality of Monitoring (QoM) systems. A more detailed task assignment problem formulation and its solutions are presented in detail in [89], but it does not consider the possibility of multiple applications using a single sensor node at the same time. In [90] cost-effective market-based algorithms are used for task allocation and resource management. But the proposed algorithms are OS specific (Sensomax) and require more work to determine their suitability. A QoS-aware task allocation algorithm in [91] brings a new dimension into the sensor node selection while satisfying QoS requirements of multiple applications at the same time. New algorithms that not only consider the QoS requirements of the applications but also take into account the properties of the events being monitored by the sensor nodes are needed to advance in this area.

6) Application Task Dissemination: When new applications are being contemplated, it is not unrealistic to assume that a new algorithm or application task will need to be sent for the sensor node(s) to execute. Sending the new task code (or updating an existing one) in a seamless way, with no disruption of existing tasks, is quite a challenge. Much of this will depend on the sensor OS and its ability to install and update user tasks without disturbing the existing ones or requiring the reboot of the sensor node. Another issue is how to get the user input, program it, and compile it to generate executable code. In the context of IoT, the user may not have technical expertise to code the required program. There needs to be a clear separation between the WSN infrastructure and the user. This can be achieved by having an entity, like service provider, to allow a user to provide her requirements in an easy manner, e.g., in a web-form. This way only some aspects of the (re)programming a sensor nodes can be exposed to the user. Once the input is gathered, the service provider can send it to the physical WSN provider to generate executable code for the selected sensor node(s) and reprogram them. Such a system will have two benefits: one is that the sensor nodes not able to fulfill a task, due to some reason, can be filtered out. Second, based on previous usage patterns of the user, a recommendation system can be devised that makes use of the historical data to recommend and (re)program the sensor nodes. An alternative approach would be to develop a cloud-based PaaS solution and provide toolkits specifically designed to develop, compile, verify, test and deploy sensor application tasks for different sensor platforms.

7) Reference Designs and Architectures: A comprehensive virtualization platform for WSNs is required, one that covers all aspects: data acquisition from the sensors, end-to-end communication (including data management and computation),

as well as service composition for end-user applications. Such a platform will allow a deeper and complete search space exploration to find the optimal solution for any given WSN application. Furthermore, this complete framework will ensure that all the relevant aspects can be modeled and evaluated comprehensively. Decentralized architectures are required that will enable robust and objective-based solutions depending on application requirements like time sensitivity, QoS, and QoM. Another important aspect is that most of the existing work focuses on the fixed WSNs but in the context of IoT, we can expect more and more deployments of mobile WSNs and even spontaneous ad hoc WSNs. These ad hoc WSNs will be created when large number of sensors communicate together to provide on-demand services for a certain time period and then cease to exist. Participatory sensing and crowed-based sensing, using smart phones, are two forms of the ad hoc WSNs. There is an early work in this area [92] that aims to utilize external sensors with the smart phones. This is achieved by means of a sensor virtualization module developed for the android platform. Still we require more solutions that focus on mobile, ad hoc WSNs and even hybrid variations.

8) New Protocols, Algorithms and Simulation Tools: As mentioned in the introduction, recently WSN virtualization is getting attention from the research community and we're now seeing some new contributions in this area. For example, in [93] a harmonized transmission protocol is presented that combines transmissions from a sensor node when it is being used by multiple concurrent applications. References [94] and [95] put forth a reconfiguration scheme and a management scheme, respectively, to manage concurrent applications over a deployed WSN. It will be a good idea to have a capable simulation tool to analyze and evaluate proposed protocols and solutions, simply because initially it may not be possible to have a sizeable WSN deployment for such purposes. A new simulation tool is presented in [96] which simulates multiple concurrent applications over a WSN. While it is a good start, more effort is required to integrate such support in already well-known and established simulation tools.

9) WSN Virtualization Business Model & Standardization: A viable business model is required to allow broader (and more commercial) acceptance of WSN virtualization. This can be accomplished easily if WSN entities are decoupled into distinct roles of WSN providers, virtual sensor providers, service providers and third-party applications/service providers. Allowing third-party applications will allow for the rapid development of applications and solutions, since the existing components will be reusable. Another benefit of such business model is that it will pave the way for standardization activities in this area. In our review of WSN virtualization area we strongly felt the need for harmonization between different protocols, data formats, encoding schemes, and consortium-led efforts such as Sensor Web Enablement (SWE) [97]. Currently these incompatibilities act as major roadblocks for proposing generic and open solutions.

10) Energy Efficient Solutions: Energy efficiency will remain a key research area in WSNs, even more so when WSN virtualization is involved. While we can safely predict that future sensor nodes will be more capable and resourceful, energy efficient communications, discovery, routing and applications will still be required. So far the main focus has been on making a sensor node sleep for maximum duration possible so that it utilizes less energy. This strategy has worked reasonably well for simple applications but this trend is not sustainable in emerging IoT paradigm. Energy harvesting mechanisms need to be incorporated with WSN platforms as main or alternative source of energy. This will ensure that sensor nodes have a continuous power supply in addition to their batteries. Example of energy harvesting mechanisms are, use of ambient energy like vibrations or solar energy to generate energy [98]. There is considerable research work in this area [99] but commercial platforms are missing.

11) Access Control, Authentication, and Accounting: Another important area is to provide a controlled access to deployed WSN resources. In the context of the IoT, sensors deployed by entities like city administrations will probably allow for public access, but they will still require access control, authentication and authorization. For example, such deployments will also be used for monitoring or security applications along with public applications, hence providing access according to users will be challenging. Another aspect is that it may not be feasible for a single authority to deploy a WSN on a large scale. For areas where WSN deployments are not possible, participatory sensing can be used as an alternative. Motivating private owners to share their deployed sensors and allow remote access is a challenge. Incentives like tax rebates or reduced utility rates need to be devised to encourage voluntary participation. Using a WSN deployment for monetary benefits brings in the accounting issue—how to charge users in accordance with service contracts.

12) WSN Virtualization Application Scenarios and Test-Beds: Applications from domains such as smart cities, smart health, smart homes, green computing and pervasive computing can potentially use the WSN virtualization concept for cost effective solutions. New trends like mobile WSNs, participatory/crowd-based sensing, cloud-based remote sensing and vehicular networks can also benefit from this concept. The availability of test-bed setups like Smart Santander [79] provides a massive basis for prototyping and evaluation purposes.

VII. Conclusion

We have presented a detailed overview of WSN virtualization, as well as the current state of the art. First we categorized state-of-the-art into node-level, network-level and hybrid solutions, and explained them. We then provided a critical analysis of the existing state-of-the-art in each category and evaluated them based on a set of requirements derived from the motivating scenarios. Several research projects pertinent to this topic were also presented. We outlined several important research challenges and their possible solutions. WSN virtualization is very much relevant in the context of the IoT, in which small-scale devices, at an unprecedented scale, are expected to provide services to multiple applications concurrently, but we have yet to find a comprehensive solution that meets this challenge.

REFERENCES

[1] M. A. Feki, F. Kawsar, M. Boussard, and L. Trappeniers, "The Internet of Things: The next technological revolution," *Computer*, vol. 46, no. 2, pp. 24–25, Feb. 2013.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Comput. Netw.*, vol. 38, no. 4, pp. 393–422, Mar. 2002.

[3] S. Loveland, E. M. Dow, F. LeFevre, D. Beyer, and P. F. Chan, "Leveraging virtualization to optimize high-availability system configurations," *IBM Syst. J.*, vol. 47, no. 4, pp. 591–604, 2008.

[4] N. M. M. K. Chowdhury and R. Boutaba, "Network virtualization: State of the art and research challenges," *IEEE Commun. Mag.*, vol. 47, no. 7, pp. 20–26, Jul. 2009.

[5] Z. J. Chong *et al.*, "Autonomy for Mobility on Demand," in *Intelligent Autonomous Systems*, 12th ed., S. Lee, H. Cho, K.-J. Yoon, and J. Lee, Eds. Berlin Germany: Springer-Verlag, 2013, pp. 671–682.

[6] G. Cardone, A. Cirri, A. Corradi, and L. Foschini, "The participact mobile crowd sensing living lab: The testbed for smart cities," *IEEE Commun. Mag.*, vol. 52, no. 10, pp. 78–85, Oct. 2014.

[7] H. Ma, D. Zhao, and P. Yuan, "Opportunities in mobile crowd sensing," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 29–35, Aug. 2014.

[8] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by Internet of Things," *Trans. Emerging Tel. Technol.*, vol. 25, no. 1, pp. 81–93, Jan. 2014.

[9] I. Khan, F. Belqasmi, R. Glitho, and N. Crespi, "A multi-layer architecture for wireless sensor network virtualization," in *Proc. 6th Joint IFIP WMNC*, Dubai, UAE, 2013, pp. 1–4.

[10] L. Sarakis, T. Zahariadis, H.-C. Leligou, and M. Dohler, "A framework for service provisioning in virtual sensor networks," *J. Wireless Commun. Netw.*, vol. 2012, no. 1, pp. 1–19, Dec. 2012.

[11] A. Merentitis *et al.*, "WSN Trends: Sensor infrastructure virtualization as a driver towards the evolution of the Internet of Things," in *Proc. 7th Int. Conf. UBICOMM*, Porto, Portugal, 2013, pp. 113–118.

[12] R. Ramdhany and G. Coulson, "Towards the coexistence of divergent applications on smart city sensing infrastructure" in *Proc. 4th Int. Workshop CONET/UBICITEC*, Philadelphia, PA, USA, Apr. 8, 2013, pp. 26–30

[13] E. Patouni, A. Merentitis, P. Panagiotopoulos, A. Glentis, and N. Alonistioti, "Network virtualisation trends: Virtually anything is possible by connecting the unconnected," in *Proc. IEEE SDN4FNS*, 2013, pp. 1–7.

[14] S. Abdelwahab, B. Hamdaoui, M. Guizani, and A. Rayes, "Enabling smart cloud services through remote sensing: An Internet of everything enabler," *IEEE Internet Things J.*, vol. 1, no. 3, pp. 276–288, Jun. 2014.

[15] C. Liang and F. R. Yu, "Wireless network virtualization: A survey, some research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 358–380, 2015.

[16] M. M. Islam, M. M. Hassan, G.-W. Lee, and E.-N. Huh, "A survey on virtualization of wireless sensor networks," *Sensors*, vol. 12, no. 2, pp. 2175–2207, Feb. 2012.

[17] M. M. Islam and E.-N. Huh, "Virtualization in wireless sensor network: Challenges and opportunities," *J. Netw.*, vol. 7, no. 3, pp. 412–418, Mar. 2012.

[18] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun, "Supporting concurrent applications in wireless sensor networks," in *Proc. 4th Int. Conf. Embedded Netw. Sensor Syst.*, New York, NY, USA, 2006, pp. 139–152.

[19] A. P. Jayasumana, Q. Han, and T. H. Illangasekare, "Virtual sensor networks—A resource efficient approach for concurrent applications," in *Proc. 4th ITNG*, 2007, pp. 111–115.

[20] M. Ceriotti *et al.*, "Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment," in *Proc. Int. Conf. Inf. Process. Sensor Netw.*, Washington, DC, USA, 2009, pp. 277–288.

[21] X. Wang, J. Wang, Z. Zheng, Y. Xu, and M. Yang, "Service composition in service-oriented wireless sensor networks with persistent queries," in *Proc. 6th IEEE CCNC*, 2009, pp. 1–5.

[22] W. Dargie and C. Poellabauer, *Fundamentals of Wireless Sensor Networks: Theory and Practice*. Hoboken, NJ, USA: Wiley, 2010.

[23] R. Chu, L. Gu, Y. Liu, M. Li, and X. Lu, "SenSmart: adaptive stack management for multitasking sensor networks," *IEEE Trans. Comput.*, vol. 62, no. 1, pp. 137–150, Jan. 2013.

[24] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," *ACM Trans. Sen. Netw.*, vol. 4, no. 2, pp. 7:1–7:40, Apr. 2008.

[25] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt "RIOT OS: Towards an OS for the Internet of Things" in *Proc. 32nd IEEE INFOCOM Poster*, 2013, pp. 79–80.

[26] H. Will, K. Schleiser, and J. Schiller, "A real-time kernel for wireless sensor networks employed in rescue scenarios," in *Proc. IEEE 34th Conf. LCN*, 2009, pp. 834–841.

[27] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt, "OS for the IoT—Goals, challenges, and solutions," in *Proc. WISG*, Troyes, France, 2013, pp. 1–6.

[28] W. Dong *et al.*, "SenSpire OS: A predictable, flexible, and efficient operating system for wireless sensor networks," *IEEE Trans. Comput.*, vol. 60, no. 12, pp. 1788–1801, Dec. 2011.

[29] S. Bhatti *et al.* "MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms," *Mobile Netw. Appl.*, vol. 10, no. 4, pp. 563–579, Aug. 2005.

[30] P. Levis *et al.*, "TinyOS: An operating system for sensor networks," in *Ambient Intelligence*, W. Weber, J. M. Rabaey, and E. Aarts, Eds. Berlin, Germany: Springer-Verlag, 2005, pp. 115–148.

[31] (Accessed 27/10/2014). [Online]. Available: www.cs.colorado.edu/~rhan/sensornets.html

[32] Q. Cao, T. Abdelzaher, J. Stankovic, and T. He, "The LiteOS operating system: Towards unix-like abstractions for wireless sensor networks," in *Proc. Int. Conf. IPSN*, 2008, pp. 233–244.

[33] S. Saruwatari, M. Suzuki, and H. Morikawa, "PAVENET OS: A compact hard real-time operating system for precise sampling in wireless sensor networks," *SICE J. Control, Meas., Syst. Integr.*, vol. 5, no. 1, pp. 24–33, 2012.

[34] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki—A lightweight and flexible operating system for tiny networked sensors," in *Proc. 29th Annu. IEEE Int. Conf. Local Comput. Netw.*, 2004, pp. 455–462.

[35] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, "Protothreads: Simplifying event-driven programming of memory-constrained embedded systems," in *Proc. 4th Int. Conf. Embedded Netw. Sensor Syst.*, New York, NY, USA, 2006, pp. 29–42.

[36] D. Yazar and A. Dunkels, "Efficient application integration in IP-based sensor networks," in *Proc. 1st ACM Workshop Embedded Sens. Syst. Energy-Efficiency Buildings*, New York, NY, USA, 2009, pp. 43–48.

[37] M. Durvy *et al.*, "Making sensor networks IPv6 ready," in *Proc. 6th ACM Conf. Embedded Netw. Sensor Syst.*, New York, NY, USA, 2008, pp. 421–422.

[38] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A low-power CoAP for contiki," in *Proc. IEEE 8th Int. Conf. MASS*, 2011, pp. 855–860.

[39] N. Tsiftes, A. Dunkels, Z. He, and T. Voigt, "Enabling large-scale storage in sensor networks with the coffee file system," in *Proc. Int. Conf. Inf. Process. Sensor Netw.*, Washington, DC, USA, 2009, pp. 349–360.

[40] K. Klues *et al.*, "TOSThreads: Thread-safe and non-invasive preemption in tinyos," in *Proc. 7th ACM Conf. Embedded Netw. Sensor Syst.*, New York, NY, USA, 2009, pp. 127–140.

[41] J. Hui, "Deluge 2.0-TinyOS network programming," 2005. [Online]. Available: http://www.cs.berkeley.edu/jwhui/research/deluge/deluge-manual.pdf

[42] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," in *Proc. 10th Int. Conf. Rec. IEEE IAS Annu. Meeting ASPLOSX*, San Jose, CA, USA, 2002, pp. 85–95.

[43] J. Koshy and R. Pandey, "VMSTAR: Synthesizing scalable runtime environments for sensor networks," in *Proc. 3rd Int. Conf. Embedded Netw. Sensor Syst.*, New York, NY, USA, 2005, pp. 243–254.

[44] R. Pandey and J. Wu, "BOTS: A constraint-based component system for synthesizing scalable software systems," in *Proc. ACM SIGPLAN/ SIGBED Conf. Language, Compilers, Tool Support Embedded Syst.*, New York, NY, USA, 2006, pp. 189–198.

[45] J. Koshy and R. Pandey, "Remote incremental linking for energy-efficient reprogramming of sensor networks," in *Proc. 2nd Eur. Workshop Wireless Sensor Netw.*, 2005, pp. 354–365.

[46] D. Simon *et al.* "Java on the bare metal of wireless sensor devices: The squawk java virtual machine," in *Proc. 2nd Int. Conf. Virtual Execution Environ.*, New York, NY, USA, 2006, pp. 78–88.

[47] J. W. Muchow, *Core J2ME Technology and MIDP*. Englewood Cliffs, NJ, USA: *Prentice-Hall*, 2001.

[48] C.-L. Fok, G.-C. Roman, and C. Lu, "Agilla: A mobile agent middleware for self-adaptive wireless sensor networks," *ACM Trans. Autonom. Adapt. Syst.*, vol. 4, no. 3, pp. 16:1–16:26, Jul. 2009.

[49] S. Bhattacharya, A. Saifullah, C. Lu, and G. Roman, "Multi-application deployment in shared sensor networks based on quality of monitoring," in *Proc. 16th IEEE RTAS*, 2010, pp. 259–268.

[50] V. Gupta *et al.*, "Nano-CF: A coordination framework for macro-programming in wireless sensor networks," in *Proc. 8th Annu. IEEE Commun. SECON*, 2011, pp. 467–475.

[51] A. Eswaran, A. Rowe, and R. Rajkumar, "Nano-RK: An energy-aware resource-centric RTOS for sensor networks," in *Proc. 26th IEEE Int. RTSS*, 2005, pp. 10–265.

[52] A. Rowe, K. Lakshmanan, H. Zhu, and R. Rajkumar, "Rate-harmonized scheduling and its applicability to energy management," *IEEE Trans. Ind. Informat.*, vol. 6, no. 3, pp. 265–275, Aug. 2010.

[53] A. Rowe *et al.*, "Sensor Andrew: Large-scale campus-wide sensing and actuation," *IBM J. Res. Develop.*, vol. 55, no. 1.2, pp. 6:1–6:14, Jan. 2011.

[54] A. Muneeb and K. Langendoen. "A case for peer-to-peer network overlays in sensor networks" in *Proc. Int. WWSNA*, 2007, pp. 56–61.

[55] G. Fersi, W. Louati, and M. B. Jemaa, "Distributed hash table-based routing and data management in wireless sensor networks: A survey," *Wireless Netw*, vol. 19, no. 2, pp. 219–236, Feb. 2013.

[56] H. V. Luu and X. Tang. "Constructing rings overlay for robust data collection in wireless sensor networks" *J. Netw. Comput. Appl.*, vol. 36, no. 5, pp. 1372–1386, Sep. 2013.

[57] A. A.-B Al-Mamou and H. Labiod, "ScatterPastry: An overlay routing using a DHT over wireless sensor networks," in *Proc. Int. Conf. IPC*, 2007, pp. 274–279.

[58] J. Hoebeke, E. D. Poorter, S. Bouckaert, I. Moerman, and P. Demeester, "Managed ecosystems of networked objects," *Wireless Pers. Commun.*, vol. 58, no. 1, pp. 125–143, May 2011.

[59] I. Ishaq, J. Hoebeke, I. Moerman, and P. Demeester, "Internet of things virtual networks: Bringing network virtualization to resource-constrained devices," in *Proc. IEEE Int. Conf. GreenCom*, 2012, pp. 293–300.

[60] E. De Poorter, E. Troubleyn, I. Moerman, and P. Demeester, "IDRA: A flexible system architecture for next generation wireless sensor networks," *Wireless Netw.*, vol. 17, no. 6, pp. 1423–1440, Aug. 2011.

[61] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[62] R. Tynan, G. M. P. O'Hare, M. J. O'Grady, and C. Muldoon, "Virtual sensor networks: An embedded agent approach," in *Proc. ISPA*, 2008, pp. 926–932.

[63] C. Muldoon, G. M. P. O'Hare, and J. F. Bradley, "Towards reflective mobile agents for resource-constrained mobile devices," in *Proc. 6th Int. Joint Conf. Autonom. Agents Multiagent Syst.*, New York, NY, USA, 2007, pp. 141:1–141:3.

[64] H. M. N. D. Bandara, A. P. Jayasumana, and T. H. Illangasekare, "Cluster tree based self organization of virtual sensor networks," in *Proc. IEEE GLOBECOM Workshops*, 2008, pp. 1–6.

[65] Q. Han, A. P. Jayasumana, T. Illangaskare, and T. Sakaki, "A wireless sensor network based closed-loop system for subsurface contaminant plume monitoring," in *Proc. IEEE IPDPS*, 2008, pp. 1–5.

[66] M. Haghighi and D. Cliff, "Multi-agent support for multiple concurrent applications and dynamic data-gathering in wireless sensor networks," in *Proc. 7th Int. Conf. IMIS*, 2013, pp. 320–325.

[67] R. B. Smith, "SPOTWorld and the sun SPOT," in *Proc. 6th Int. Conf. Inf. Process. Sensor Netw.*, New York, NY, USA, 2007, pp. 565–566.

[68] A. Majeed and T. A. Zia, "Multi-set architecture for multi-applications running on wireless sensor networks," in *Proc. IEEE 24th Int. Conf. WAINA*, 2010, pp. 299–304.

[69] J. Steffan, L. Fiege, M. Cilia, and A. Buchmann, "Towards multi-purpose wireless sensor networks," in *Proc. Syst. Commun.*, 2005, pp. 336–341.

[70] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, "SenShare: Transforming sensor networks into multi-application sensing infrastructures," in *Wireless Sensor Networks*, G. P. Picco and W. Heinzelman, Eds. Berlin, Germany: Springer-Verlag, 2012, pp. 65–81.

[71] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proc. 2nd Int. Conf. Embedded Netw. Sensor Syst.*, New York, NY, USA, 2004, pp. 81–94.

[72] J. Lu *et al.*, "The smart thermostat: Using occupancy sensors to save energy in homes," in *Proc. 8th ACM Conf. Embedded Netw. Sensor Syst.*, New York, NY, USA, 2010, pp. 211–224.

[73] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proc. 7th ACM Conf. Embedded Netw. Sensor Syst.*, New York, NY, USA, 2009, pp. 1–14.

[74] M. Navarro, M. Antonucci, L. Sarakis, and T. Zahariadis, "VITRO architecture: Bringing virtualization to WSN world," in *Proc. IEEE 8th Int. Conf. MASS*, 2011, pp. 831–836.

[75] T. Zahariadis, P. Trakadas, H. C. Leligou, S. Maniatis, and P. Karkazis, "A novel trust-aware geographical routing scheme for wireless sensor networks," *Wireless Pers. Commun.*, vol. 69, no. 2, pp. 805–826, Mar. 2013.

[76] P. Levis, "Experiences from a decade of tinyos development," in *Proc. 10th USENIX Conf. Oper. Syst. Des. Implementation*, Berkeley, CA, USA, 2012, pp. 207–220.

[77] R. N. Murty *et al.*, "CitySense: An urban-scale wireless sensor network and testbed," in *Proc. Conf. Technol. Homeland Security*, 2008, pp. 583–588.

[78] C. Efstratiou, I. Leontiadis, C. Mascolo, and J. Crowcroft, "A Shared Sensor Network Infrastructure," in *Proc. 8th ACM Conf. Embedded Netw. Sensor Syst.*, New York, NY, USA, 2010, pp. 367–368.

[79] L. Sanchez *et al.*, "SmartSantander: IoT experimentation over a smart city testbed," *Comput. Net.*, vol. 61, pp. 217–238, Mar. 2014.

[80] F. Berkers *et al.*, "Constructing a multi-sided business model for a smart horizontal IoT service platform," in *Proc. 17th Int. Conf. ICIN* 2013, pp. 126–132.

[81] A. Andrushevich *et al.*, "Leveraging multi-domain links via the Internet of Things," in *Internet of Things, Smart Spaces, and Next Generation Networking*, S. Balandin, S. Andreev, and Y. Koucheryavy, Eds. Berlin, Germany: Springer-Verlag, 2013, pp. 13–24.

[82] D. Irwin *et al.*, "Towards a virtualized sensing environment," in *Testbeds and Research Infrastructures. Development of Networks and Communities*. Berlin, Germany: Springer-Verlag, 2011, pp. 133–142.

[83] R. S. Oliver, I. Shcherbakov, and G. Fohler, "An operating system abstraction layer for portable applications in wireless sensor networks," in *Proc. ACM Symp. Appl. Comput.*, 2010, pp. 742–748.

[84] J. Mäenpää, J. J. Bolonio, and S. Loreto, "Using RELOAD and CoAP for wide area sensor and actuator networking," *J. Wireless Commun. Netw.*, vol. 2012, no. 1, pp. 1–22, Dec. 2012.

[85] Z. Shelby, "Embedded web services," *IEEE Wireless Commun.*, vol. 17, no. 6, pp. 52–57, Dec. 2010.

[86] Z. Shelby *et al.*, "Constrained Application Protocol (CoAP)," work in progress, Internet Eng. Task Force-IETF, Fremont, CA, USA, Draft-ietf-core-coap-18, Jun. 2013.

[87] S. Cheshire and M. Krochmal, "Multicast DNS," IETF, Fremont, CA, USA, RFC 6762, Feb. 2013.

[88] R. Glitho, M. Morrow, and P. Polakos, "A cloud based—Architecture for cost-efficient applications and services provisioning in wireless sensor networks," in *Proc. 6th Joint IFIP WMNC*, 2013, pp. 1–4.

[89] H. Rowaihy *et al.*, "Sensor-mission assignment in wireless sensor networks," *ACM Trans. Sensor Netw.*, vol. 6, no. 4, pp. 36:1–36:33, Jul. 2010.

[90] M. Haghighi, "Market-based resource allocation for energy-efficient execution of multiple concurrent applications in wireless sensor networks," in *Mobile, Ubiquitous, and Intelligent Computing*, J. H. Park, H. Adeli, N. Park, and I. Woungang, Eds. Berlin, Germany: Springer-Verlag, 2014, pp. 173–178.

[91] W. Li, F. C. Delicato, P. F. Pires, and A. Y. Zomaya, "Energy-efficient task allocation with quality of service provisioning for concurrent applications in multi-functional wireless sensor network systems," *Concurrency Comput., Pract. Exp.*., vol. 26, no. 11, pp. 1869–1888, Aug. 2014.

[92] J. Ko, B.-B. Lee, S. G. Hong, and N. Kim, "Poster abstract: Virtualizing external wireless sensors for designing personalized smartphone services," in *Proc. 12th Int. Conf. Inf. Process. Sensor Netw.*, New York, NY, USA, 2013, pp. 353–354.

[93] V. Gupta, N. Pereira, E. Tovar, and R. Rajkumar, "Poster abstract: A harmony of sensors: Achieving determinism in multi-application sensor networks," in *Proc. 13th Int. Symp. Inf. Process. Sensor Netw.*, Piscataway, NJ, USA, 2014, pp. 299–300.

[94] C.-M. Hsieh, Z. Wang, and J. Henkel, "DANCE: Distributed application-aware node configuration engine in shared reconfigurable sensor networks," in *Proc. Conf. Des., Autom. Test Eur.*, San Jose, CA, USA, 2013, pp. 839–842.

[95] T. M. Cao, B. Bellata, and M. Oliver, "Design of a generic management system for wireless sensor networks," *Ad Hoc Netw.*, vol. 20, pp. 16–35, Sep. 2014.

[96] M. Haghighi, "An agent-based multi-model tool for simulating multiple concurrent applications in WSNs" in *Proc. 5th Int. Conf. Commun. Softw. Netw. JACN, Malaysia*, Jun. 2013, pp. 1–6.

[97] C. Reed *et al.*, "Ogc sensor web enablement:overview and high level achhitecture" in *Proc. IEEE Autotestcon*, 2007, pp. 372–380.

[98] E. Gelenbe, D. Gesbert, D. Gunduz, H. Kulah, and E. Uysal-Biyikoglu, "Energy harvesting communication networks: Optimization and demonstration (the E-CROPS project)," in *Proc. 24th TIWDC*, 2013, pp. 1–6.
[99] S. Sudevalayam and P. Kulkarni, "Energy harvesting sensor nodes: Survey and implications," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 3, pp. 443–461, Sep. 2011.

Imran Khan (S'11) received the B.S. degree in computer science from COMSATS Institute of IT, Pakistan, and the M.S. degree in multimedia and communication from M.A. Jinnah University, Pakistan. He was member of Center of Research in Networks and Telecom (CoReNeT) from 2008 to 2010 and worked on projects funded by French Ministry of Foreign Affairs and Internet Society (ISOC). Currently, he is pursuing the Ph.D. degree at Institut Mines-Télécom, Télécom SudParis jointly with Paris VI (UPMC). He is also a collaborating researcher at Concordia University, Montreal, Canada, working on a project funded by Cisco. He has contributed to the IETF standardization. His research interests are Internet of Things (IoT), virtualization, cloud computing, service engineering, and wireless sensor networks.

Fatna Belqasmi (M'09) received the M.Sc. degree in electrical and computer engineering and the Ph.D. degree from Concordia University, Canada. Previously, she worked as a research associate at Concordia University, Canada, and as a researcher at Ericsson Canada. She was part of the IST Ambient Network project (a research project sponsored by the European Commission within the Sixth Framework Programme -FP6-). She worked as an R&D Engineer for Maroc Telecom in Morocco. Currently, she is working as Assistant Professor at Zayed University, Abu Dhabi, UAE. Her research interests include next generation networks, service engineering, distributed systems, and networking technologies for emerging economies.

Roch Glitho (SM'97) received the M.Sc. degrees in business economics from University of Grenoble, France, and in pure mathematics and in computer science from University of Geneva, Switzerland. He also received the Ph.D. (Tekn. Dr.) degree in teleinformatics from the Royal Institute of Technology, Stockholm, Sweden. He works in Montreal, Canada, as an Associate Professor of networking and telecommunications at the Concordia Institute of Information Systems Engineering (CIISE) where he leads the telecommunication service engineering (TSE) research laboratory (http://users.encs.concordia.ca/~tse/). In the past, he has worked in industry for almost a quarter of a century and has held several senior technical positions at LM Ericsson in Sweden and Canada (e.g., expert, principal engineer, senior specialist). His industrial experience includes research, international standards setting (e.g., contributions to ITU-T, ETSI, TMF, ANSI, TIA, and 3GPP), product management, project management, systems engineering and software/firmware design. In the past, he has served as IEEE Communications Society distinguished lecturer, Editor-in-Chief of IEEE COMMUNICATIONS MAGAZINE and Editor-in-Chief of IEEE COMMUNICATIONS SURVEYS & TUTORIALS. His research areas are: virtualization and cloud computing; machine-to-machine communications (M2M) and Internet of Things; distributed systems (e.g., SOAP based—web services, RESTful web services); rural communications and networking technologies for emerging economies.

Noel Crespi (M'07–SM'08) received the master's degree from the Universities of Orsay (Paris 11) and Kent (U.K.), a diplome d'ingénieur from Telecom ParisTech, the Ph.D. and Habilitation from Paris VI University (Paris-Sorbonne). From 1993, he was with CLIP, Bouygues Telecom, and with Orange Labs in 1995. He took leading roles in the creation of new services with the successful conception and launch of Orange prepaid service and in standardization (from rapporteurship of IN standard to coordination of all mobile standards activities for Orange). In 1999, he joined Nortel Networks as telephony program manager, architecting core network products for the EMEA region. He joined Institut Mines-Telecom in 2002 and is currently Professor and Program Director, leading the Service Architecture Lab. He coordinates the standardization activities for Institut Mines-Telecom at ITUT, ETSI, and 3GPP. He is also an Adjunct Professor at KAIST, an Affiliate Professor at Concordia University, and is on the four-person Scientific Advisory Board of FTW (Austria). He is the Scientific Director the French-Korean laboratory ILLUMINE. His current research interests are in service architectures, services webification, social networks, and Internet of Things/Services.

Monique Morrow (SM'09) holds the title of CTO Cisco Services. Her focus is in developing strategic technology and business architectures for Cisco customers and partners. With over 13 years at Cisco, she has made significant contributions in a wide range of roles, from Customer Advocacy to Corporate Consulting Engineering. With particular emphasis on the Service Provider segment, her experience includes roles in the field (Asia-Pacific) where she undertook the goal of building a strong technology team, as well as identifying and grooming a successor to assure a smooth transition and continued excellence. She has consistently shown her talent for forward thinking and risk taking in exploring market opportunities for Cisco. She was an early visionary in the realm of MPLS as a technology service enabler, and she was one of the leaders in developing new business opportunities for Cisco in the Service Provider segment, SP NGN. She holds three patents, and has an additional nine patent submissions filed with US Patent Office. She is the co-author of several books, and has authored numerous articles. She also maintains several technology blogs and is a major contributor to Cisco's Technology Radar, having achieved Gold Medalist Hall of Fame status for her contributions. She is also very active in industry associations. She is a new member of the Strategic Advisory Board for the School of Computer Science at North Carolina State University. She is particularly passionate about Girls in ICT and has been active at the ITU on this topic—presenting at the EU Parliament in April of 2013 as an advocate for Cisco. Within the Office of the CTO, first as an individual contributor, and now as CTO, she has built a strong leadership team, and she continues to drive Cisco's globalization and country strategies.

Paul Polakos received the B.S., M.S., and Ph.D. degrees in physics from Rensselaer Polytechnic Institute and the University of Arizona. He is currently a Cisco Fellow and member of the Mobility CTO team at Cisco Systems focusing on emerging technologies for future Mobility systems. Prior to joining Cisco, he was Senior Director of Wireless Networking Research at Bell Labs, Alcatel-Lucent in Murray Hill, NJ, USA, and Paris, France. During his 28 years at Bell Labs he worked on a broad variety of topics in physics and in wireless networking research including the flat-IP cellular network architecture, the Base Station Router, femtocells, intelligent antennas and MIMO, radio and modem algorithms and ASICSs, autonomic networks and dynamic network optimization. Prior to joining Bell Labs, he was a member of the research staff at the Max-Planck Institute for Physics and Astrophysics (Munich) and visiting scientist at CERN and Fermilab. He is an author of more than 50 publications and 30 patents.

Annex B

Paper II

Wireless Sensor Network Virtualization: Early Architecture and Research Perspectives

Imran Khan, Fatna Belqasmi, Roch Glitho, Noel Crespi, Monique Morrow, and Paul Polakos

Abstract

WSNs have become pervasive and are used in many applications and services. Usually, deployments of WSNs are task-oriented and domain-specific, thereby precluding reuse when other applications and services are contemplated. This inevitably leads to the proliferation of redundant WSN deployments. Virtualization is a technology that can aid in tackling this issue, as it enables the sharing of resources/infrastructure by multiple independent entities. In this article we critically review the state of the art and propose a novel architecture for WSN virtualization. The proposed architecture has four layers (physical layer, virtual sensor layer, virtual sensor access layer, and overlay layer) and relies on a constrained application protocol. We illustrate its potential by using it in a scenario where a single WSN is shared by multiple applications, one of which is a fire monitoring application. We present the proof-of-concept prototype we have built along with the performance measurements, and discuss future research directions.

In the last few years, wireless sensor networks (WSNs) have become ubiquitous and are being used in a broad array of application domains, including healthcare, agriculture, surveillance, and security. These WSNs are composed of small-scale nodes that have the ability to sense, compute, and communicate [1]. While early sensor nodes were resource-constrained with limited capabilities, recent advances in sensor hardware technology have made it possible to produce sensor nodes that have more processing power and memory, and prolonged battery life.

Virtualization is a key technique for the realization of the future Internet, and it is indeed quite pertinent to explore it in the context of WSNs. Virtualization makes it possible to present physical computing resources by abstracting them into logical units, enabling their efficient usage by multiple independent users, including multiple concurrent applications [2]. Furthermore, it allows for the deployment of applications that were not even envisioned during an infrastructure's initial deployment.

To date, realizations of WSNs have been domain-specific and task-oriented. Applications are bundled with a WSN at the time of deployment, and it is next to impossible to use the same WSN for another application. This leads to redundant deployments and underutilization of these resources. There are two approaches to allow multiple applications to access deployed WSN resources. One is to allow multiple applications to share the data gathered from a WSN. In this approach, a sink/gateway node collects all the data from the WSN and shares it among multiple users. For example, in [3], WSNs are merged into the cloud by sending observed sensor data through a host manager that lies outside the WSN. The host manager simply collects the sensor data, profiles/aggregates it, and then allows multiple applications to use it for their own purposes.

The second approach is to use the capabilities of the individual sensor nodes to execute multiple application tasks concurrently, and allow applications to group these sensor nodes together according to their requirements. The key difference between the two approaches is that the former approach allows the sharing of WSN data among multiple applications, while the latter allows sharing of WSN nodes by multiple applications. This article is focused on the second approach because it makes it possible to provision more innovative applications over the deployed WSNs, even applications that were not envisioned a priori. This will greatly improve the efficiency of deployed WSNs and will also encourage new business models.

This article introduces the WSN virtualization concept, critically reviews the state of the art in WSN virtualization, and proposes a new early architecture that focuses on fixed WSNs. We illustrate the potential of the architecture by instantiating it for a fire monitoring scenario [4] in which multiple applications share the same WSN. We have built a prototype to demonstrate its feasibility and to measure its performance. We also identify further research directions.

The next section presents a critical overview of the state of

Imran Khan and Noel Crespi are with Télécom SudParis.

Fatna Belqasmi is with Zayed University.

Roch Glitho is with Concordia University.

Monique Morrow and Paul Polakos are with Cisco Systems.

This article is an extended version of a short paper presented at the 6th Joint IFIP Wireless and Mobile Networking Conference (WMNC '13), April 23–25, 2013, Dubai, UAE, under the title "A Multi-Layer Architecture for Wireless Sensor Network Virtualization."

0890-8044/15/$25.00 © 2015 IEEE

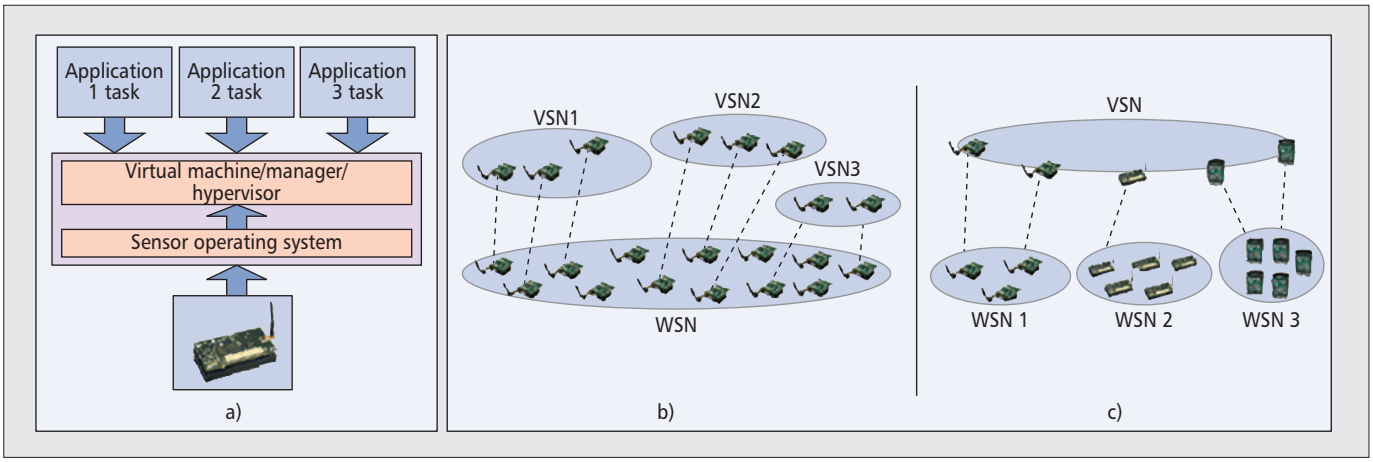


Figure 1. WSN virtualization categories: a) a general-purpose sensor node; b) multiple VSNs over a single WSN; c) a single VSN over multiple WSNs.

the art. The proposed architecture is presented in the third section. The fourth section discusses the implementation alternatives with the proof-of-concept prototype and the recorded performance measurements. The research directions are discussed in the fifth section. We conclude in the last section by discussing the lessons learned.

A Critical Overview of the State of the Art

There are two categories of WSN virtualization: node level and network level. Figure 1 shows a high-level view of WSN virtualization. WSN node-level virtualization allows multiple applications to run their tasks concurrently on a single WSN node [5] (Fig. 1a). This execution can be sequential (e.g., round-robin) or simultaneous, with context switching between application tasks.

In WSN network-level virtualization, a subset of sensor nodes belonging to a deployed WSN forms a virtual sensor network (VSN) to execute given application tasks at a given time [6], while the other sensor nodes remain available for other application tasks. WSN network-level virtualization can be achieved in two ways. Different VSNs can be created over the same underlying WSN infrastructure (Fig. 1b), or sensor nodes can form a single VSN over multiple WSNs in different administrative domains (Fig. 1c). The latter is possible when the sensor nodes can support the concurrent execution of application tasks. This is the case these days because many popular sensor operating systems (e.g., Contiki and Squawk VM) that run on resource-constrained devices enable node-level virtualization through the concurrent execution of applications' tasks on the same sensor node.

Motivating Example and Requirements

In this section we first present a motivating example and then draw requirements from it.

Motivating Example — A real-world deployment of a WSN is presented in [7], in which a WSN is used to monitor the impact of constructing a road tunnel under an ancient tower in Italy, as it was feared that the tower could lose its ability to stand on its own and might collapse during the construction. Now consider that there are three users interested in the fate of the tower. The first is the construction company, as it needs to make sure that the tower does not lose its ability to stand on its own; otherwise, it will have to pay a heavy fine. The second user is the conservation board, which routinely monitors all the ancient sites around the city. The third user is the local municipality, which has to plan emergency remedial/rescue actions in case the tower falls during the construction.

It is quite possible that the conservation board has already deployed its own WSN to monitor the health of ancient sites, including this tower. In this case the construction company and local municipality can reuse the existing sensor nodes during the construction period. In the absence of WSN virtualization, there are only two possible solutions. One is to rely on the information provided by the conservation board application. However, this information may not be at the required granularity level. Worse, some of the information that is needed might simply not be available because the requirements of the construction company and the local municipality were not considered when the conservation board application was designed and implemented. The second solution is that each user deploys redundant WSN nodes, but this is an inefficient approach.

Requirements — The *first* requirement is the support of node-level virtualization to allow the execution of multiple application tasks on the same sensor node. The *second* requirement is the ability of sensor nodes to dynamically form groups to execute isolated and transparent application tasks concurrently (i.e., support for WSN network-level virtualization). The *third* requirement is support of application priority. In some critical application scenarios such as fire monitoring, it is important that other tasks have less priority than the one reporting the fire event.

The *fourth* requirement is that the proposed solution should be applicable to a wide range of applications and not tailored for a particular scenario or domain, as is the case with most solutions. The *fifth* requirement is that the proposed solution should be platform-independent and not depend on specific operating systems or customized/tailored interfaces. The *sixth* and final requirement is that the solution should address heterogeneity, that is, cope with sensor nodes that have different capabilities (e.g., processing power, memory).

The State of the Art and Its Shortcomings

We divide the related work into three classes: node-level, network-level, and hybrid virtualization solutions. The hybrid solutions combine both node- and network-level virtualization.

Node-Level Virtualization — In order to achieve node-level virtualization, mechanisms must be in place to allow deployed WSN nodes to execute new application tasks as well as update existing ones. One solution is to reprogram WSN nodes individually, but that is neither feasible nor efficient. Wireless reprogramming, on the other hand, allows large numbers of WSN nodes to be updated with new application tasks with

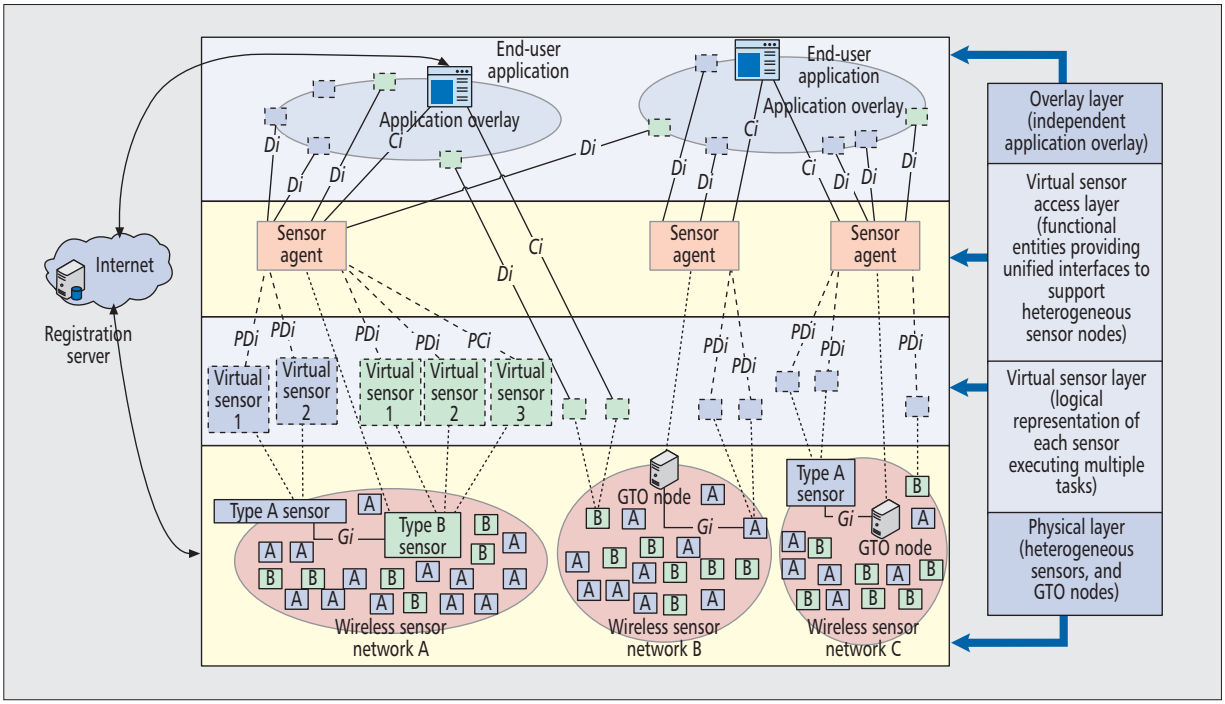


Figure 2. Multi-layer WSN virtualization architecture.

minimum effort. It is now the main mechanism used for node-level virtualization. Two examples of node-level virtualization based on wireless reprogramming are discussed below. Their main drawback is platform dependence.

Maté [5] is a pioneering work that provides sequential execution of application tasks on resource-limited early-generation sensor nodes. It is a tiny virtual machine consisting of a stack-based binary code interpreter and works on top of TinyOS. Application tasks are divided into code capsule(s) of up to 24 instructions and are executed one by one. A viral code distribution scheme is used to propagate code and reprogram the sensor nodes. As there is tight coupling between the application code and TinyOS, installing a new code requires the replacement of the whole OS. There is no support for application priority, and only a limited set of applications is supported. Furthermore, the approach is not platform-independent since it only works on TinyOS, but it does address heterogeneity.

MANTIS [8] is a thread-based embedded operating system. Programs are created as user-level threads with dedicated memory space and static data attached to them at compile time. Long-running threads can be preempted by short-running threads. The work on wireless reprogramming is ongoing according to the authors. The techniques used are the wireless reflashing of the OS and reprogramming of single threads. Unlike Maté, MANTIS does provide application priority. However, it is not platform-independent.

Network-Level Virtualization — In [6], sensor nodes form clusters to support applications that monitor dynamic phenomena. The sensor nodes within each cluster execute application(s) tasks, meaning a sensor node can be part of multiple clusters. With each cluster dedicated to an application, a WSN can be utilized by multiple applications concurrently, hence realizing network-level virtualization. Two illustrative applications are presented as motivation. Unfortunately, the work is poor in terms of technical details (e.g., how individual nodes execute application tasks). Furthermore, there is no discussion of how application priority, heterogeneity, and platform independence are tackled. This work was extended in [9] in order to facilitate the creation, operation, and maintenance of dynamic clusters to achieve network-level virtualization. Once an event is detected, sensor nodes are grouped as a dynamic cluster tree by exchanging VSN formation messages. However, in terms of our requirements, none of the drawbacks of [6] are addressed.

The authors in [10] introduce the problem of mission assignment in WSNs. The work can be related to network-level virtualization because the WSN is able to support multiple missions at the same time. Each mission uses a dedicated subset of sensor nodes that are not shared with other missions. A mission assignment problem is modeled as a weighted bipartite graph to optimally assign the sensor nodes to missions. Achieving a mission produces a profit, so the goal is to maximize profit by efficiently achieving as many missions as possible. Both centralized and distributed solutions are presented, using proofs and algorithms including an energy-aware solution. This solution does not consider any specific application domain. Heterogeneity is addressed along with platform independence. However, application task priority is not provided since each sensor node executes only one application task at a time.

Hybrid Solutions — The authors in [11] discuss the SenShare platform, which supports both WSN-node and network-level virtualization. They consider TinyOS applications with an embedded hardware abstraction layer. The underlying sensor node resources are then accessed using a runtime layer on top of TinyOS. Since TinyOS supports multiple tasks at the same time, node-level virtualization is achieved. For network-level virtualization, an overlay network using Collection Tree Protocol (CTP) is created to group sensor nodes executing the same application. The physically scattered sensor nodes executing the same application can be grouped into a single overlay network. SenShare is the first solution targeting comprehensive WSN virtualization. It supports node- and network-level virtualization, application priority, and heterogeneity, and it is independent of any application domain. However, it is not platform-independent, as only TinyOS applications are supported.

Melete [12] is an extension of Maté and supports both node- and network-level virtualization. Concurrent execution

of application tasks is achieved by making the following enhancements to Maté: dedicated storage and execution space for applications to allow concurrency, and a code dissemination protocol to allow selective and reactive (re)programming of sensor nodes. For network-level virtualization it uses a dynamic grouping technique of sensor nodes. A sensor node can be part of more than one logical group at the same time. The supported network topology is a connected graph. Melete does not support application priority and is not platform-independent. It only supports a limited set of applications, but it does tackle heterogeneity.

Proposed Architecture

In this section, we first present the architectural principles. We then present our multi-layer architecture based on overlays, followed by a discussion of the interfaces and the overlay creation procedure.

Architectural Principles

The first architectural principle is that new applications/services are deployed as new overlays on top of the physical WSN. Overlays have several advantages: they are distributed, lack central control, and allow resource sharing [13]. 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 application tasks at any given time.

The third principle is that not all WSN nodes perform the overlay-related operations, 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. This principle in effect makes it possible to address the heterogeneity requirement and enables network-level virtualization for early-generation resource-constrained sensor nodes.

The fourth principle is that within the architecture there are separate data and control paths. 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., changing application priority and overlay management) is sent over the control path. This separation of paths makes it easy to work on new protocols for each path independently.

The last principle is the use of emerging standards, aimed at resource-constrained devices, to tackle the platform independence challenge. These standards include protocols such as the Constrained Application Protocol (CoAP) [14] and DNS-Service Discovery (DNS-SD) [15], and standards such as Sensor Model Language (SensorML) [16], Observations & Measurements (O&M) [17] and Sensor Markup Language (SenML) [18]. This principle of course implies the need for converters/mappers for devices that do not support the standards.

CoAP is an application-layer transfer protocol, like HTTP, designed to work with resource-constrained devices. It has less overhead, memory, and processing requirements than HTTP. DNS-SD offers service discovery in resource-constrained networks and allows for the seamless integration of such architectures into the existing IP networks. SensorML provides standard models and XML-based encoding to describe sensor measurements and processes. It is able to provide interoperability, automatic discovery, utilization, and sensor sharing. O&M is a standard that defines encoding schemas for the observations made by sensors. SenML provides a data model for sensor measurements and simple metadata about sensors in JSON, XML, and EXI formats.

Abbreviation	Component	Remarks
—	Type A sensor	Legacy/resource-constrained sensor
—	Type B sensor	New-generation smart IP sensor node
GTO node	Gates-to-overlay node	Gateway/sink node capable of joining application overlays on behalf of type A sensors
—	Sensor agent	Functional entity providing a unified interface to provide platform independence
—	Registration server	Sensor repository
Di	Data interface	Interface to send sensor data to application overlay
PDi	Proprietary data interface	Proprietary interface to send virtual sensor data to sensor agent
Ci	Control interface	Interface to send/receive control data from end-user application
PCi	Proprietary control interface	Proprietary interface to send/receive control data from virtual sensor to sensor agent
Gi	Gates-to-overlay interface	Interface to send/receive control data between type A sensors and type B sensors/GTO nodes

Table 1. Components of the architecture.

Overall Architecture

Figure 2 shows our proposed multi-layer architecture, and Table 1 provides the list of components used. There are four layers (physical, virtual sensor, virtual sensor access, and overlay), two paths (data and control), five interfaces (data [*Di*], proprietary *Di* ([PD*i*], control [*Ci*], proprietary *Ci* [PC*i*], and gateway [*Gi*]), and a registration server.

At the physical layer we have independent WSNs that consist of two types of sensor nodes, that is, resource constrained (type A) and capable (type B) sensors. Each WSN also has specialized nodes, called GTO nodes. Their role is to help type A sensors join the application overlays and provide heterogeneity. Gateways, sink nodes, or type B sensors can act as GTO nodes when required. For example, in the motivating example in the previous section, if the existing sensors are of type A, either the existing gateway node or type B sensors deployed by the construction company can help those sensors become part of the construction company overlay. This might increase the complexity of the type B sensor nodes but does allow flexibility.

The virtual sensor layer consists of the logical representation of each sensor executing multiple application tasks concurrently. Each logical representation is called a virtual sensor in our architecture, which is an abstraction of an application task run by a sensor.

The virtual sensor access layer consists of sensor agents, which ensure platform independence. This is achieved by providing standardized interfaces (*Di* and *Ci*) to interact with the

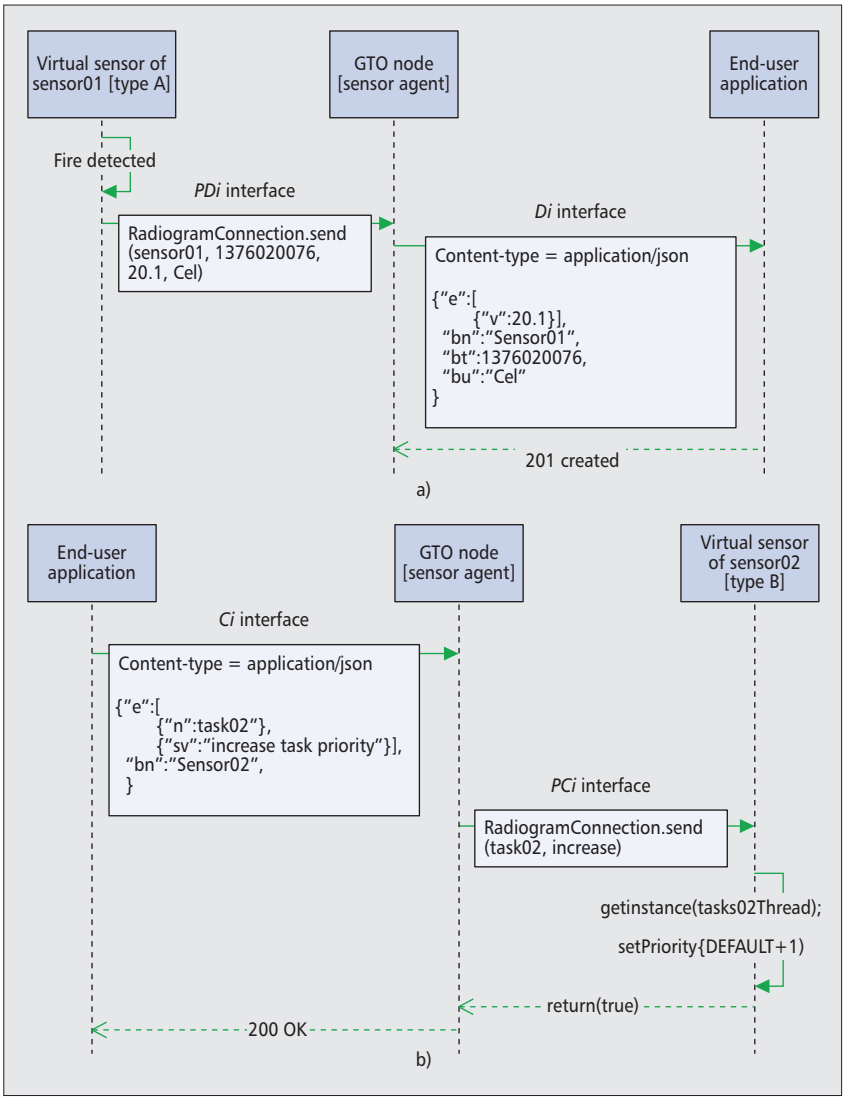


Figure 3. Example of communication over data and control interfaces: a) sending sensor data over *PDi* and *Di* interfaces; b) changing application task priority over *PCi* and *Ci* interfaces.

end-user applications, and are mapped onto platform-specific (proprietary) interfaces (*PDi* and *PCi*) for the underlying physical sensor nodes. Sensor agents can be implemented in either capable (type B) sensors or GTO nodes.

The overlay layer consists of independent application-specific overlays (two are shown in Fig. 2, but there could be many more). Each application overlay is created by the end-user application and consists of virtual sensors that run the overlay application tasks. An overlay protocol is used for message exchange inside an overlay. A registration server, which contains the details of the deployed sensor nodes, is used by end-user applications to find sensor nodes.

Interfaces

The data path uses the data interface (*Di*) supported by all of the sensor agents to send the data received from the virtual sensors executing the end user's application task to the application overlays. The control path uses the control interface (*Ci*) supported by all sensor agents to send/receive control data. Examples of control data include sending requests to change application priority and sampling frequency. The interfaces, *PDi* and *PCi*, are proprietary and are used by the sensor agent to communicate with WSNs. Figure 3 shows high-level examples of when sensor data is sent over *PDi* and *Di* interfaces (Fig. 3a) (when fire is detected) and when a request to change application task priority is sent over *Ci* and *PCi* interfaces (Fig. 3b). In this case it is the priority of the task running on sensor 02. The gates-to-overlay interface (*Gi*) is provided by all the sensors as well as the GTO nodes. Any communication from type B or GTO nodes with type A sensors is done using this interface.

Overlay Creation Procedure

This section describes the overlay creation procedure. The creation of the overlay is a three-step procedure initiated by the end-user application. The first step is dynamic resource discovery and overlay preconfiguration, allowing the discovery of the sensors and GTO nodes on the fly according to the requirements of the end-user application. The second step is the activation of the overlay. The selected sensor (type B) and GTO nodes receive an overlay join request (or advertisement) over the *Ci* interface. After joining the overlay, the type B sensors and GTO nodes (for type A sensors) may receive the application task with its desired priority level. The final step is the execution of the end-user application, which begins when each sensor starts executing the end-user application task. Depending on the application requirements, sensors may exchange messages among themselves in the overlay before sending any data to the end-user application over the *Di* interface.

Implementation Alternatives, Proof of Concept Prototype, and Measurements

Implementation Alternatives

Our proposed architecture consists of the data plane, the control plane, and several interfaces that belong to them. The *Di* interface, belonging to the data plane, carries the actual data. The *Ci* and *Gi* interfaces carry control messages and are part of the control plane.

There are several options for implementing a data plane interface. Both HTTP and CoAP can be used as application-layer protocols, but we chose CoAP as it will allow type A nodes to support the same protocol for *Di* and *Gi* interfaces. We use SenML specifications to encode the sensor data in standard JSON format. The combination of SensorML and O&M is another option, but we selected SenML since it is less complex.

For the control plane, one candidate protocol is JXTA [19], an open source peer-to-peer protocol specification that allows the creation of independent, robust, and efficient overlay networks. ScatterPastry [20] is another option. For our work we opted to use JXTA since its implementations are readily available.

Prototype

We implemented a simple brush fire scenario discussed in [4] as a prototype. In this scenario, the city administration is interested in the early detection of brush fire eruption and in its evolution, using a WSN and a fire contour algorithm

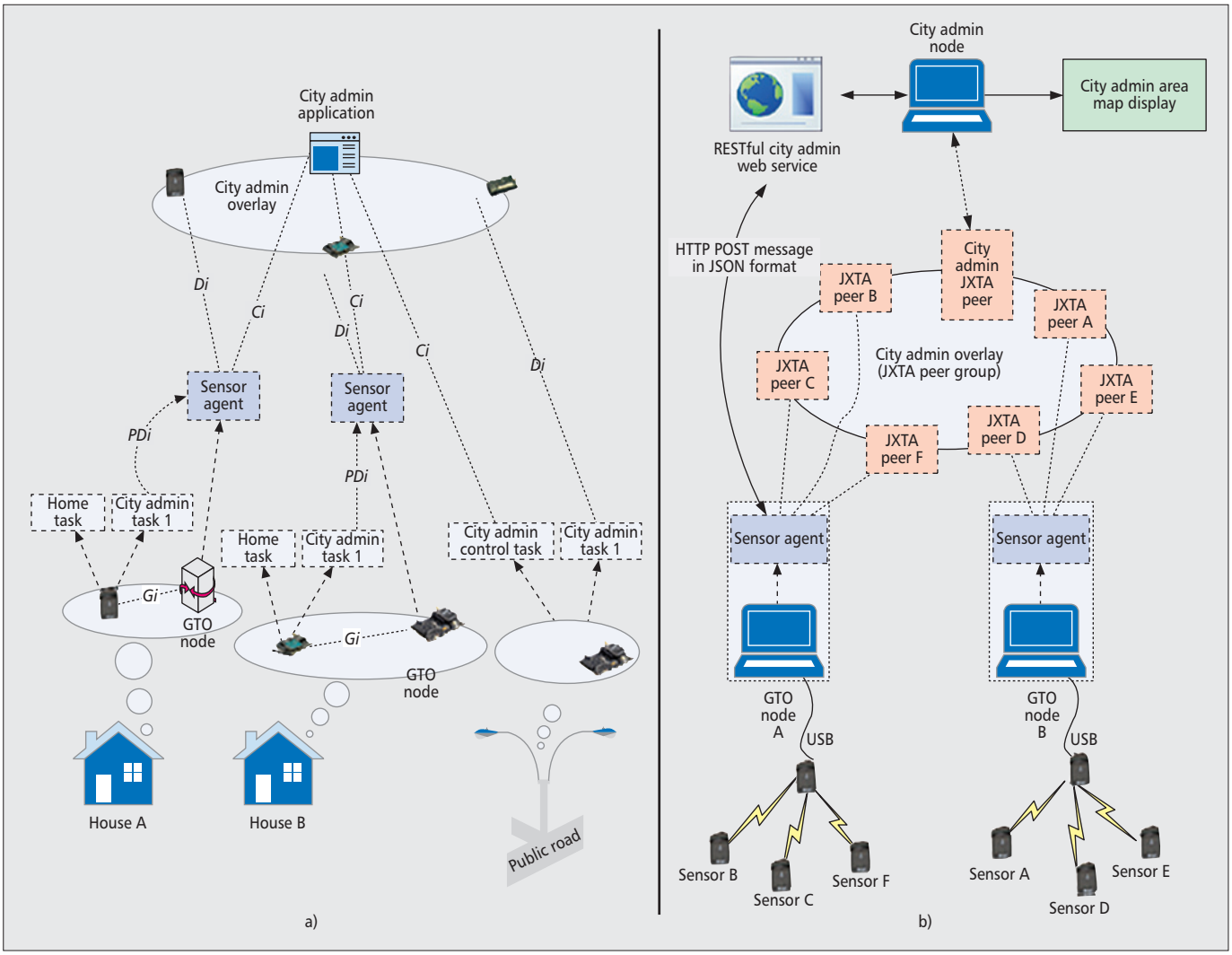


Figure 4. Instantiation of the architecture and prototype setup: a) instantiation of the architecture; b) prototype setup.

(FCA). Some houses in the area already have their own sensors to detect fire. To accelerate the deployment of its application and avoid redundancy, the city administration has opted to deploy sensors in areas under its jurisdiction (i.e., streets and parks) and incorporate the sensor nodes already deployed in private homes. The home owners get incentives like tax rebates for allowing the use of their sensors by city administration. The home gateways acts as GTO nodes. All of the privately owned sensors execute two application tasks one for the home owner and one for the city administration. Figure 4a shows the mapping of the scenario onto our architecture.

We make the following assumptions. First, we assume that the city administration has already discovered and sent its application task to each of these sensors. The second assumption is that all of the sensors in the prototype are type A sensors that need a GTO node for overlay-related tasks. Third, as it was not possible to generate a fire in a lab environment, the city administration application task periodically measured the temperature value in a sensor and sent it to the GTO node. We used six Java SunSpots sensors, each executing three application tasks concurrently. The application tasks were coded in Java 2 Platform Micro Edition (J2ME). J2ME is a robust, flexible Java platform that enables the development of applications for mobile and embedded devices. The city administration's overlay network was implemented using a Java-based implementation of JXTA, JXSE 2.6.

A RESTful web service is used by the city administration node to receive fire alerts. Each GTO node, upon receiving fire notification from its sensor, sends an HTTP POST message to a URI (*http:///FireContourService/events/fire/*) to create a fire event. The content type of the HTTP POST message is set to application/senml+json, and the event data received from Java SunSpot is mapped to JSON format according to SenML specifications. Once the event is created, the city administration node sends a fire notification message to the peers in the overlay.

The overlay is created by the city admin node, acting as rendezvous peer, by advertising its peer group (fire contour service) using JXTA pipe advertisements before the fire event. The GTO nodes join the fire contour service as edge peers by replying to the received pipe advertisement. The city admin node sends the fire notification message using the JXTA multicast socket, which provides efficient message exchange between members of the same peer group. After the execution of the fire contour algorithm, the reply message is sent directly to the city admin node instead of being multicast.

The prototype uses a simple probabilistic fire contour algorithm, considering that a distant house will send fire notifications less frequently than a nearby house because the fire is far from it. The city administration's application, created using JavaFX, receives the fire alert messages as well as the peers' replies, and displays the

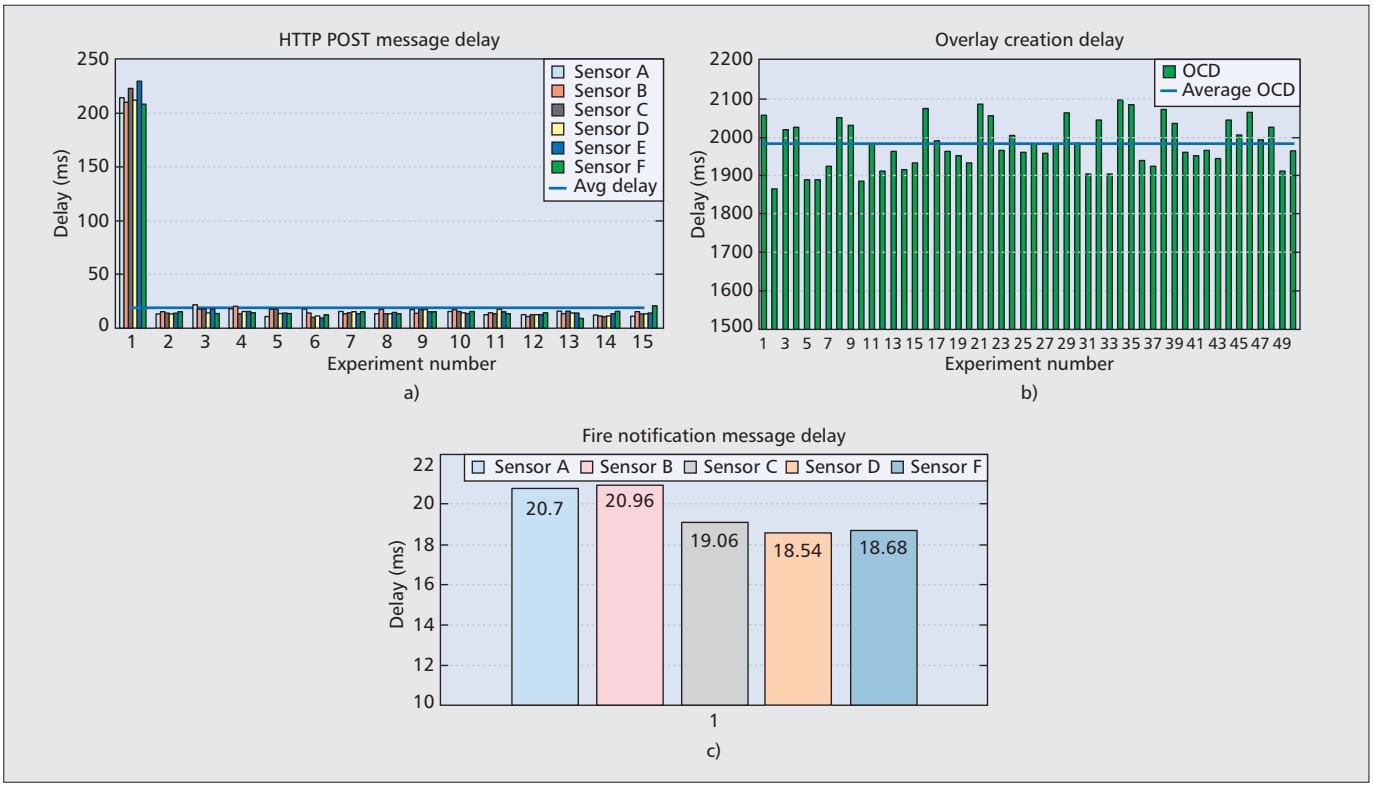


Figure 5. Results: a) HTTP POST message delay; b) overlay creation delay; c) fire notification message delay.

output using the area map. JavaFX is a set of Java libraries that allow developers to rapidly design, create, and deploy client applications that operate across diverse platforms.

The prototype setup is illustrated in Fig. 4b. The city administration application and its fire contour web service ran on a laptop with an Intel Core i5 CPU clocked at 2.67 GHz, and a 4 GB RAM with 32-bit Windows 7 Enterprise. The other two laptops acted as GTO nodes for Java SunSpots and ran three JXTA peers each. Their configurations were an Intel Core i7 CPU clocked at 2.70 GHz with 8 GB RAM, 64-bit Windows 7 Professional, an Intel Core i5 CPU clocked at 2.60 GHz, and a 4GB RAM with Windows 7 Enterprise. All three laptops used JVM version 1.7.0_21 and were connected to a private LAN.

Performance Measurements

Performance Metrics — The performance of the prototype was assessed in terms of the following delays: *HTTP POST delay* (HPD), *overlay creation delay* (OCD), and *fire notification delay* (FND).

HPD is the time difference between when the GTO node sends an HTTP POST request and when it receives the corresponding success code (201 created). HPD is calculated for each sensor. OCD is the time it takes to set up the city administration overlay from a nonexistent state to a ready state, when it advertises its fire contour service and is ready to accept join requests. We measured this delay inside the Java code to ensure that the OCD does not include the JVM start-up delay. FND is measured as the time it takes for the city admin node to multicast fire notification messages to JXTA peers and receive their replies after they execute the fire contour algorithm. For each experiment we restarted the JVM and cleared the previous JXTA configuration cache. All delays are measured in milliseconds and calculated at the sender side.

Performance Results — The HPD measurements are shown in Fig. 5a (for clarity, only 15 measurements are shown). The dark blue horizontal line shows the average delay for the *50* measurements, *18.96 ms*. It is observed that the delay for the first POST message is much larger than that for the subsequent messages. This long delay is due to the three-way handshake of TCP connection that takes place during the first POST message, whereas for subsequent requests a persistent HTTP connection (a.k.a. HTTP keep-alive) reduces delay considerably. Figure 5b shows the OCD of a city admin JXTA peer with an average value of *1983 ms* from 50 iterations indicated by the horizontal blue line. The delay includes the JXTA core startup, the creation of a fire contour service, the related pipe advertisement, a JXTA multicast socket, and the thread for accepting join requests from other JXTA peers. For each iteration a new JXTA cache was generated instead of using the old one. Figure 5c shows the average FND of five sensors that executed a fire contour algorithm in response to a notification message sent by a city admin JXTA peer. In this case sensor E reported the fire. The average FND of five sensors is *19.58 ms*.

In order to determine the overhead of WSN virtualization, we consider the scenario where sensors do not support node-level virtualization and only execute city admin tasks. There is also no network-level virtualization and no overlay network for message exchange. In this case, the fire counter algorithm will be executed by the GTO nodes after getting an HTTP POST message from the city admin node. For a simple comparison, if we consider that the FND without WSN virtualization is similar to HPD, that is, *18.96 ms*, and FND with WSN virtualization is *19.58 ms*, then with WSN virtualization we have approximately *3.27 percent* overhead. This overhead is due to the processing of XML-based JXTA messages. Our implementation demonstrates that WSN virtualization is indeed feasible and does not incur much overhead. Node-level virtualization is achieved with Java SunSpots with very little effort. Network-level virtualization is achieved using JXTA, and once JXTA is operational, the delays are mini-

mal. OCD is inevitable, but in the long run, using JXTA is beneficial as it provides a robust, highly scalable, and efficient solution.

Overall, the results show the typical delays experienced in a private LAN setting. The same JXTA pipe advertisement of the fire contour service was used to send and receive the fire notification messages over a JXTA multicast socket, which greatly improved the overall performance.

Research Directions

WSN virtualization is a very rich research area, and our proposed preliminary architecture has raised several interesting issues. This section provides a non-exhaustive sample. The first issue is a dynamic publication and discovery framework for sensor and GTO nodes. In this work, we assumed a static publication process where the sensor and GTO owners publish their nodes to a central repository. To automate the process of WSN virtualization, an on-the-fly publication and discovery mechanism would be required. A CoAP-based framework could be used as starting point. For a centralized solution, a CoAP resource directory (RD) mechanism can be used, while a CoAP resource discovery mechanism would be more appropriate for a distributed solution. Similarly, a DNS-SD mechanism can be used in combination with CoAP to provide new solutions.

The choice of data formats for various interfaces is another issue. The current OGC — O&M and SensorML specifications use the XML format, which is inefficient in resource-constrained environments. SenML addresses this issue by using JSON and EXI formats, and it works with both HTTP and CoAP, but it also has some open issues. For example, we can use it to specify simple metadata about measurements, but there is no mechanism to provide such data for describing the sensors, their capabilities, and their resources (memory, space, and battery life) at a particular time. The possibility of a lightweight mechanism for reporting a sensors' runtime status is very appealing. Similarly, a semantically enriched format would be of particular use for creating intelligent sensor-based systems in the context of the Internet of Things, which is currently not possible with SenML.

An important issue is optimal task assignment to sensors. The problem is essentially the mapping of end-user application requirements to the available resources, which is very challenging in a virtualized environment. Reference [10] proposes a solution, but it assumes that every sensor executes a single task, which is not the case in a virtualized environment. However, it could be used as starting point for further research. WSN-oriented overlay middleware is yet another issue to investigate. We need an efficient solution that prevents overlays from interacting in a harmful way when they compete for underlying resources. JXTA and similar protocols work well, but not in resource-constrained environments. Some early attempts like [20] exist, but they must be combined with the concept of WSN virtualization.

A signaling framework to support quality of service (QoS) and session management is also needed. Issues like handling application requests for setting/changing task priority will be tackled by such a general QoS framework. There are several signaling frameworks, such as SIP/RSVP, but they may not be suitable for sensors. Again, a CoAP-based signaling protocol is a potential solution. Using the virtualization concept for mobile WSNs is also interesting, since they are becoming more and more popular. Vehicular ad hoc networks, social networks, and crowd-based sensing can provide concrete application scenarios to motivate the virtualization of mobile WSNs.

Lessons Learned

In this article we have proposed a new preliminary multi-layer architecture for WSN virtualization and have identified several research directions.

We have learned several lessons. The first is that WSN node-level virtualization is still in its infancy, and very few WSN kits supporting node-level virtualization are readily available. This is certainly due to the challenges of designing hypervisors in resource-constrained environments. A second lesson is that most existing WSN standard specifications pertinent to our work are still embryonic. SenML, for instance, is very promising. However, in its present form, it is not suitable for control functions. On the other hand, SensorML is complex and not suitable for a general-purpose and efficient solution. A third lesson is that most existing overlay middleware is unsuitable for WSNs because it is usually not designed for resource-constrained devices. We used JXSE, which is one of the best choices available. However, its current open source implementation is rather old, and the future of the initiative is uncertain.

Acknowledgments

This work is partially supported by Cisco Systems through grant CG-576719, and by the Canadian National Science and Engineering Research Council (NSERC) through the Canada Research Chair in End-User Service Engineering for Communications Networks.

References

[1] I. F. Akyildiz *et al.*, "Wireless Sensor Networks: A Survey," *Computer Networks*, 38.4, 2002, pp. 393–422.
[2] S. Loveland *et al.*, "Leveraging Virtualization to Optimize High-Availability System Configurations," *IBM Sys. J.*, vol. 47, no. 4, 2008, pp. 591–604.
[3] M. Fazio *et al.*, "Huge Amount of Heterogeneous Sensed Data Needs the Cloud," *Proc. 9th IEEE Int'l. Multi-Conf. Systems, Signals and Devices*, Chemnitz, Germany, 20–23 Mar. 2012.
[4] I. Khan *et al.*, "A Multi-Layer Architecture for Wireless Sensor Network Virtualization," *Proc. 6th Joint IFIP Wireless and Mobile Networking Conf.*, Apr. 23–25, 2013, Dubai, UAE, pp. 1–4.
[5] P. Levis and D. Culler: "Maté: A Tiny Virtual Machine for Sensor Networks," *ASPLOSX: Proc. 10th Int'l. Conf. Architectural Support for Programming Languages and Op. Sys.*, San Jose, CA, 2002, pp. 85–95.
[6] A. P. Jayasumana *et al.*, "Virtual Sensor Networks a Resource Efficient Approach for Concurrent Applications," *Proc. 4th Int'l. Conf. Info. Tech.*, 2007, Las Vegas, NV, 2007, pp. 111–15
[7] M. Ceriotti *et al.*, "Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment," *Proc. 2009 Int'l. Conf. Info. Processing in Sensor Networks*, IEEE Computer Society, 2009.
[8] S. Bhatti *et al.*, "MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms," *Mobile Network Applications*, vol. 10, no. 4, 2005, pp.563–79.
[9] H. M. N. Dilum Bandara *et al.*, "Cluster Tree Based Self Organization of Virtual Sensor Networks," *Proc. IEEE GLOBECOM Wksp. Wireless Mesh and Sensor Networks*, New Orleans, LA, Nov. 2008.
[10] H. Rowaihy *et al.*, "Sensor-Mission Assignment in Wireless Sensor Networks," *ACM Trans. Sensor Networks*, 6.4, 2010, p. 36.
[11] I. Leontiadis *et al.*, "SenShare: Transforming Sensor Networks into Multi-Application Sensing Infrastructures," *Wireless Sensor Networks*, Springer, 2012, pp. 65–81.
[12] Y. Yu *et al.*, "Supporting Concurrent Applications in Wireless Sensor Networks," *Proc. 4th Int'l. Conf. Embedded Networked Sensor Systems*, Boulder, CO, 2006, pp.139–52.
[13] E. K. Lua *et al.*, "A Survey and Comparison of Peer-To-Peer Overlay Network Schemes," *IEEE Commun. Surveys and Tutorials*, vol. 7, no. 2, 2005, pp. 72–93.
[14] Z. Shelby "Embedded Web Services," *IEEE Wireless Commun.*, vol. 17, no. 6, Dec. 2010, pp. 52–57.
[15] A. J. Jara *et al.*, "Light-Weight Multicast DNS and DNS-SD (lmDNS-SD): IPv6-Based Resource and Service Discovery for the Web of Things," *Proc. Sixth Int'l Conf. Innovative Mobile and Internet Services in Ubiquitous Computing*, 2012, vol., no., 4–6 July 2012, pp. 731–38.
[16] M. Botts and A. Robin, "Sensor Model Language (SensorML)," Open Geospatial Consortium (OGC) doc. no. 07-000, Wayland, MA.
[17] S. Cox, Ed., "Observations and Measurements — Part 1 — Observation Schema," OGC doc. no. 07-002r1, Wayland, MA.

[18] Jennings *et al.*, "draft-jennings-senml-10," IETF Internet draft, Oct. 22, 2012; expired April 25, 2013.
[19] L. Gong, "JXTA: A Network Programming Environment.," *IEEE Internet Comp.*, vol. 5.3, 2001, pp. 88–95.
[20] AAl-B. Mamou and H. Labiod, "ScatterPastry: An Overlay Routing Using a DHT over Wireless Sensor Networks," *Proc. IEEE Int'l. Conf. Intelligent Pervasive Computing*, 2007, pp. 274–79.

Biographies

IMRAN KHAN [S] (imran@ieee.org) received his B.S. degree in computer science from COMSATS Institute of IT, Pakistan, and his M.S. degree in multimedia and communication from M.A. Jinnah University, Pakistan. He was a member of the Center of Research in Networks and Telecom (CoReNeT) from 2008 to 2010 and worked on projects funded by French Ministry of Foreign Affairs and Internet Society (ISOC). Currently, he is pursuing his Ph.D. degree at Institut Mines-Télécom, Télécom SudParis jointly with Paris VI (UPMC). He is also a collaborating researcher at Concordia University, Montreal, Canada, working on a project funded by Cisco. He has contributed to IETF standardization. His research interests are Internet of Things (IoT), virtualization, cloud computing, service engineering, and wireless sensor networks.

FATNA BELQASMI (fatna.belqasmi@zu.ac.ae) holds a Ph.D. and an M.Sc. degree in electrical and computer engineering from Concordia University, Canada. She is currently working as an assistant professor at Zayed University Abu Dhabi, United Arab Emirates. In the past, she worked as a research associate at Concordia University, Canada, and as a researcher at Ericsson Canada. She was part of the IST Ambient Network project (a research project sponsored by the European Commission within the Sixth Framework Programme -FP6-). She worked as an R&D engineer for Maroc Telecom in Morocco. Her research interests include next generation networks, service engineering, distributed systems, and networking technologies for emerging economies.

ROCH GLITHO [SM] (http://users.encs.concordia.ca/~glitho/) holds a Ph.D. (Tekn. Dr.) in tele-informatics (Royal Institute of Technology, Stockholm, Sweden) and M.Sc. degrees in business economics (University of Grenoble, France), pure mathematics (University Geneva, Switzerland), and computer science (University of Geneva). He works in Montreal, Canada, as an associate professor of networking and telecommunications at the Concordia Institute of Information Systems Engineering (CIISE) where he leads the telecommunication service engineering (TSE) research laboratory (http://users.encs.concordia.ca/~tse/). In the past he worked in industry for almost a quarter of a century, holding several senior technical positions at LM Ericsson in Sweden and Canada (e.g., expert, principal engineer, senior specialist). His industrial experience includes research, international standards setting (e.g., contributions to ITU-T, ETSI, TMF, ANSI, TIA, and 3GPP), product management, project management, systems engineering, and software/firmware design. In the past he has served as an IEEE Communications Society Distinguished Lecturer, Editor-In-Chief of *IEEE Communications Magazine*, and Editor-In-Chief of *IEEE Communications Surveys & Tutorials*. His research areas are virtualization and cloud computing, machine-to-machine communications (M2M) and Internet of Things; distributed systems (e.g., SOAP-based web services, RESTful web services); and rural communications and networking technologies for emerging economies.

NOEL CRESPI (noel.crespi@mines-telecom.fr) received his Master's degree from the Universities of Orsay (Paris 11) and Kent (United Kingdom), a Diplome d'Ingénieur from Telecom ParisTech, and his Ph.D. and Habilitation from Paris VI University (Paris-Sorbonne). From 1993, he was with CLIP, Bouygues Telecom, and with Orange Labs in 1995. He took leading roles in the creation of new services with the successful conception and launch of Orange prepaid service and in standardization (from rapporteurship of the IN standard to coordination of all mobile standards activities for Orange). In 1999, he joined Nortel Networks as telephony program manager, architecting core network products for the EMEA region. He joined Institut Mines-Telecom in 2002 and is currently a professor and program director, leading the Service Architecture Lab. He coordinates the standardization activities for Institut Mines-Telecom at ITU-T, ETSI, and 3GPP. He is also an adjunct professor at the Korea Advanced Institute of Science and Technology, an affiliate professor at Concordia University, and is on the four-person Scientific Advisory Board of FTW (Austria). He is the scientific director of the French-Korean laboratory ILLUMINE. His current research interests are in service architectures, services webification, social networks, and Internet of Things/Services.

MONIQUE MORROW (mmorrow@cisco.com) holds the title of CTO, Cisco Services. Her focus is on developing strategic technology and business architectures for Cisco customers and partners. With over 13 years at Cisco, she has made significant contributions in a wide range of roles, from customer advocacy to corporate consulting engineering. With particular emphasis on the service provider segment, her experience includes roles in the field (Asia-Pacific) where she undertook the goal of building a strong technology team, as well as identifying and grooming a successor to ensure a smooth transition and continued excellence. She has consistently shown her talent for forward thinking and risk taking in exploring market opportunities for Cisco. She was an early visionary in the realm of MPLS as a technology service enabler, and was one of the leaders in developing new business opportunities for Cisco in the service provider segment, SP NGN. She holds three patents, and has an additional nine patent submissions filed with the U.S. Patent Office. She is the co-author of several books, and has authored numerous articles. She also maintains several technology blogs, and is a major contributor to Cisco's Technology Radar, having achieved Gold Medalist Hall of Fame status for her contributions. She is also very active in industry associations. She is a new member of the Strategic Advisory Board for the School of Computer Science at North Carolina State University. She is particularly passionate about Girls in ICT and has been active at the ITU on this topic, presenting at the EU Parliament in April 2013 as an advocate for Cisco. Within the office of CTO, first as an individual contributor and now as CTO, she has built a strong leadership team, and she continues to drive Cisco's globalization and country strategies.

PAUL POLAKOS (ppolakos@cisco.com) is currently a Cisco Fellow and member of the Mobility CTO team at Cisco Systems focusing on emerging technologies for future mobility systems. Prior to joining Cisco, he was Senior Director of Wireless Networking Research at Bell Labs, Alcatel-Lucent in Murray Hill, New Jersey and Paris, France. During his 28 years at Bell Labs he worked on a broad variety of topics in physics and wireless networking research, including the flat-IP cellular network architecture, the base station router, femtocells, intelligent antennas and MIMO, radio and modem algorithms and ASICSs, autonomic networks, and dynamic network optimization. Prior to joining Bell Labs, he was a member of the research staff at the Max-Planck Institute for Physics and Astrophysics, Munich, Germany, and a visiting scientist at CERN and Fermilab. He holds B.S., M.S., and Ph.D. degrees in physics from Rensselaer Polytechnic Institute and the University of Arizona, is a Bell Labs and Cisco Fellow, and is an author of more than 50 publications and 30 patents.

Annex C

Paper III

Getting Virtualized Wireless Sensor Networks' IaaS Ready for PaaS

Imran Khan*†, Fatima Zahra Errounda†, Sami Yangui†, Roch Glitho† and Noël Crespi*
*Institut Minés-Télécom, Télécom SudParis, 91011 Evry Cedex, France
Email: imran@ieee.org, noel.crespi@it-sudparis.eu
†Dept. CIISE, Concordia University, H3G 2W1, Montreal, Canada
Email: {f_errou, s_yangui} @encs.concordia.ca, glitho@ciise.concordia.ca

***Abstract*—With the recent advances in sensor hardware and software, architectures for virtualized Wireless Sensor Networks (vWSNs) are now emerging. Through node- and network-level virtualization, vWSNs can be offered as Infrastructure-as-a-Service (IaaS) which can aid in realizing the true potential of Internet-of-Things (IoT). Cloud computing offers elastic provisioning of large-scale infrastructures to multiple concurrent users where Platform-as-a-Service (PaaS) interacts with IaaS in order to efficiently host and execute applications over these infrastructures. Amalgamating IoT with cloud computing potentially allows rapid application and service provisioning in an efficient, scalable and robust manner. However, interactions between vWSNs and PaaS are largely an unexplored area. Indeed, existing vWSN IaaS are not yet ready for PaaS. This paper proposes a vWSN IaaS architecture which is ready for interactions with PaaS. The proposed architecture is based on our previous works and is rooted in the fundamental differences between traditional IaaS and vWSN IaaS. We built a prototype using Java Sunspot as the WSN tool kit and made early performance measurements.**

Keywords*—*Wireless Sensor Networks; Internet of Things; Cloud Computing; Virtualization; IaaS; PaaS

I. Introduction

Since their mainstream introduction towards the end of 20th century, Wireless Sensor Network (WSN) deployments have been used as means to bridge the gap between the physical world and the virtual world. With their ability to sense, compute and communicate, WSNs provide their users with the ability to react to various physical phenomenon and take required actions [1]. WSNs are considered as basic building blocks of Internet-of-Things (IoT) paradigm [2] where sensors, along with multitude of everyday objects communicate, interact and share data on a massive scale [3].

Cloud computing [4] paradigm allows several inherent benefits (e.g., efficient usage of resources, scalability, elasticity, and rapid provisioning of new applications). It has three key facets: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS). Service providers use PaaS to provision applications and services as SaaS on a pay-per-use basis to the end-users. PaaS ease the provisioning process by adding levels of abstraction to the infrastructure. This abstraction is achieved by using the virtualization concept that allows sharing of resources by abstracting them into multiple logical units on the same physical node [5].

WSNs can be virtualized at node-level [6] as well as at network-level [7]. At node-level, multiple applications can run tasks concurrently on a single WSN node, either sequentially (round-robin) or simultaneously (context switching). At network-level, groups of WSN nodes form Virtual Sensor Networks (VSNs) to execute a given application task at a given time. There can be multiple such groups in a WSN deployment, each dedicated to a different application. A detailed survey discussing the basics, motivation, benefits and existing works on WSN virtualization can be found in [8].

Architectures that combine WSN node- and network-level virtualization are now emerging (e.g., [9], [10] and [11]). However, they are still not yet ready for PaaS. They lack the appropriate design and architectural details to enable proper interactions with the PaaS so that service providers are able to efficiently provision new WSN applications and services. The problem is challenging because vWSN IaaS are fundamentally different from traditional IaaS. For example, in traditional IaaS the concept of Virtual Machine (VM) is used, which is characterized by its operating system, unique global address, processing power and memory. On the other hand, in vWSNs the concept of Virtual Sensor (VS) is used, which is characterized by its sensor middleware, platform-dependent localized address and scarcity of processing power and memory. Moreover, issues like geospatial location and sampling rate impose additional constraints.

This paper proposes an architecture to offer competent vWSN IaaS, which is able to interact with PaaS to allow service providers to rapidly provision WSN-based applications and services. The proposed architecture is based on our previous work [10] and on the fundamental differences between the vWSN IaaS and traditional IaaS that we have identified. Unlike our previous work, this paper focuses on architectural design and details to enable interactions between vWSN IaaS and PaaS for dynamic provisioning of applications and services.

The paper is organized as follows. In Section II differences between traditional IaaS and vWSN IaaS are presented along with the requirements for a PaaS ready vWSN IaaS. Section III presents the proposed vWSN architecture. Details on the implementation and the results are presented in Section VI. Section V discusses the lessons learned and future work while Section VI concludes the paper.

II. Fundamental Differences Between WSN IaaS and Traditional IaaS

The fundamental differences between vWSN IaaS and traditional IaaS stem from the differences between WSNs and

traditional networks. In this section, we first briefly discuss how WSN and traditional networks differ before introducing the fundamental differences between vWSN IaaS and traditional IaaS. Our analysis will be structured around the concepts of VM (i.e., the fundamental element of traditional IaaS) and VS (i.e., the fundamental element of vWSN IaaS). Finally, we present a set of requirements for a PaaS ready vWSN IaaS.

A. Differences between WSN and Traditional Networks

WSNs are known to be resource-constrained environments whose nodes typically have limited processing capability, storage and are battery operated. The nodes have low duty cycle [12] and operate only at specific intervals [13]. This means that WSN nodes are not always available for applications. In traditional networks, nodes (server, computers) have considerable resources and potentially have unlimited power source allowing high duty cycle and high availability. This fundamental difference has led to numerous research efforts aimed at designing energy efficient protocols [14], simple data formats [15] and simple application design [16] for WSNs. Another important difference between the two network types is the availability of protocols. IP rules traditional networks whereas in WSN it is not much prevalent yet but there have been efforts to bring IP to the WSN world [17], [18] and [19]. HTTP is not as much useful in WSNs as in traditional networks but alternatives like CoAP [20] have emerged for WSNs. We observe that the advent of IoT paradigm has prompted many efforts to provide standard protocol support for WSNs [21].

B. Differences between VM and VS

A VM is defined as a logical unit that allows time and resource sharing of host machines by partitioning them into multiple dedicated execution environments [22]. Each VM has a guest operating system that can access underlying resources. On the other hand, a VS is a logical representation of the physical sensor to allow sharing of its sensing capabilities (e.g., temperature and light sensing capabilities) [10]. VSs execute multiple concurrent application tasks. On an abstract level, a VS is similar to a VM, i.e., both provide a mechanism to decouple physical resources from their host nodes in order to be used by multiple users. For example, in traditional IaaS, the resources of a host machine are represented by a VM Monitor (VMM) or Hypervisor that allows multiple VMs to access underlying resources [23]. In vWSN IaaS, if we consider the example of Java SunSpots, the Squawk virtual machine [24] provides a similar type of abstraction that allows multiple VSs to access the sensing resources of a sensor. Still, there are certain fundamental differences between the two. Table I lists seven such differences, which are explained below.

The *first* difference is that a VM allows for the sharing of resources (e.g., computing and storage) of the host machine, whereas a VS allows sharing of sensing capabilities (e.g., temperature, light, humidity) by executing multiple application tasks. The key difference is that a VM aims at sharing the host machine resources, whereas a VS may use the computing and storage of the host sensor, but it aims at sharing the sensing capabilities of the host sensor. In Java SunSpots, for instance, application tasks access the on-board sensors to sense the physical phenomenon, and send the data accordingly.

The *second* difference is that multiple heterogeneous VMs (in terms of operating systems) can be simultaneously deployed on the same host. For instance, a host can support a Linux-based VM and/or a Windows-based VM at the same time. However, VSs are tightly coupled with their sensor OS/middleware. For example, a sensor cannot support Contiki-based VS and TinyOS-based VS at the same time.

The *third* difference is that multiple VMs can be deployed in an isolated manner. The creation, deployment, and migration of VMs does not affect the execution of existing VMs. On the other hand, the deployment of new VS may disturb the execution of existing VS(s). This is due to the limited resources and the tight coupling between the VS and the sensor OS/middleware. Similarly, migrating VS from one physical sensor to another is not a standard feature yet. To the best of our knowledge, Java SunSpots is the only platform that provides support for VS migration (as serialized Java Isolates). There is additional work in which a mobile agent-based system for Java SunSpots is developed for VS migration [25].

The *fourth* difference is that VMs can be addressed by other entities that are similar to their host machines. Each VM can be assigned a public or private IP address and can be accessed accordingly. However, there is currently no standard mechanism for addressing a VS. Typically, a local ID is used and may vary depending on the platform. This necessitates some address mapping/translation mechanism to communicate with a VS. For instance, in Java SunSpots, each VS can be addressed by a MIDlet ID.

The *fifth* difference is that for a VM, there are no power/energy-related issues, whereas a VS inherits these issues from the host sensor nodes. This means that the creation, deployment, and operation of a VS are not only dependent on the capabilities/resources of the host sensor, but also on its available energy. The always-on/always-available concept is not applicable to WSN world.

The *sixth* difference is that for VMs, there are already some open source and proprietary solutions (e.g., KVM and VMware). However, no such solutions exist for VSs.

The *seventh and final* difference is that, at the IaaS level, the role of a VM is to maximize the use of a host machines resources (e.g., computing and storage), while the role of a VS is to use the sensing capabilities of the host sensor in an efficient manner. Therefore, to achieve cost-efficiency, traditional IaaS may create several VMs on a limited number of host machines. However, achieving cost-efficiency in vWSN IaaS may not lead to the creation of several VSs on a few host sensor nodes since the deployment of sensor nodes is strongly correlated to the desired coverage of a geographic area.

C. Requirements for a PaaS Ready vWSN IaaS

The *first* requirement is that vWSN IaaS should support standard interfaces for interacting with a PaaS API. These standardized interfaces will allow easy instantiation, operation and management of VSs from PaaS. RESTful interfaces are lightweight and can be useful in resource-constrained environment like vWSN.

The *second* requirement is that once created, the VS should be addressable similar to a VM in traditional IaaS. This

TABLE I. CONCEPTUAL DIFFERENCES BETWEEN VM AND VS

Virtual Machine	Virtual Sensor
Logical representation of host machine	Logical representation of sensing capabilities of host sensor
Deployment of multiple OS-heterogeneous VMs	Middleware-dependent deployment of VS
Isolated deployment	Non-isolated deployment
Standard IP-based address mechanism	No standard mechanism to address
Unlimited power supply (for the physical host, i.e. server)	Battery operated (for physical host; i.e., sensor)
Proprietary and open source solutions	Currently no solutions
Uses resources of host machine (computing, storage)	Uses sensing capabilities of host sensor

will allow PaaS to seamlessly manage these VSs (e.g. start, stop, migrate and/or delete). Similarly, depending on the PaaS requirements and vWSN IaaS capabilities, certain parameters could be dynamically adjusted to configure VSs, such as sampling rate, reporting interval or even task migration (e.g. when monitoring dynamic events). In traditional IaaS, VMs get IP address and are accessible from anywhere, whereas the addressing mechanism of VSs depends on the platform and can be either a task-ID, MIDlet-ID, or some variation of 64-bit IEEE hardware address. A mapping scheme at vWSN IaaS can be used to map global addresses to local ones.

The *third* requirement is that the vWSN IaaS should be able to publish available services provided by the deployed sensors. For application development, PaaS will need to discover services provided by sensors, for example it might look for temperature service at a particular location for a certain duration and upon finding appropriate sensor, proceed to create a VS on it. In this situation a static or simple service description will not suffice for publication, instead it should include the spatial/temporal characteristics while considering the current load on that particular sensor. A centralized or distributed repository can be used for this purpose.

The *fourth* requirement is the lifecycle management and monitoring of VSs by the vWSN IaaS. In resource-constrained environments, VSs may not be as stable as VMs in traditional IaaS. Energy deficiency coupled with low bandwidth and hardware issues make it difficult to have always-on or always-available VSs. A robust VS lifecycle management and monitoring will be useful, e.g. in releasing VS (deleting them) when they are no longer in use, map application requirements from PaaS to available sensors, and help in fault detection and solution in the vWSN IaaS. However, satisfying spatial and temporal requirements is not trivial.

The *fifth and final* requirement is the support for inter-vWSN IaaS interactions. Typical WSN deployments will span over a geographic area and may need to interact with each other according to the requirements of the applications. Such interaction needs to involve SLAs, policy enforcement and of course deal with privacy and security issues.

Reference [8] provides an exhaustive survey of vWSN solutions but none of them meets all these requirements.

III. PROPOSED WSN IAAS ARCHITECTURE

In this section, we first present our previous vWSN architecture since we use it as a starting point for this work. Later we discuss our proposed vWSN IaaS architecture.

A. vWSN Architecture

This work is based on our previous work [10] in which we proposed a vWSN architecture shown in Fig. 1. It is a multi-layer architecture that exploits the capabilities of individual sensor nodes to run concurrent application tasks at node-level and dynamically assembles such nodes at network-level for data sharing. The Physical layer has resource-constrained sensors (e.g., TelosB motes) and capable sensors (e.g., Java SunSpots). Since resource-constrained sensors may not support WSN network-level virtualization, they rely on capable Gate-to-Overlay (GTO) nodes (e.g., base station nodes, sink nodes and capable sensors) for this purpose.

Next, in the Virtual Sensor layer, we have VSs that are abstractions of the application tasks run by the physical sensors. For each application, there is one VS running its task. The third layer is the Virtual Sensor Access layer. It consists of Sensor Agents (SAs) that provide platform independence by using standardized north-bound interfaces and proprietary south-bound interfaces. The final layer is the Overlay layer, which consists of multiple application overlays that use the deployed WSN. There are separate interfaces for data and control messages. The architecture is platform independent, applicable to different types of sensors, and does not cater any specific application domain.

B. Proposed vWSN IaaS Architecture

The proposed vWSN IaaS architecture is shown in Fig. 2. The following is the detailed description of the architecture.

The bottom two layers (WSN Infrastructure and Virtual Sensors) are similar to the ones in the previous architecture and consist of heterogeneous sensors, GTO nodes and virtual sensors. The functionality of these two layers and the roles of their entities are same as described in the Section III-A.

We have added a new layer called Virtual Sensor Manager, which contains two new functional entities: The VS Manager and VS Communicator. VS Manager receives requests to instantiate, start, stop, delete, and migrate VS. Tasks such as VS migration can only be accomplished if supported by the vWSN IaaS. The task code, which is to be run by the VS, is also disseminated through the VS Manager.

The VS Communicator supports platform-specific protocols to interact with different sensor platforms to promote platform heterogeneity. Examples of these protocols include IEEE 802.15.4, Bluetooth, Cellular, and RESTful.

Next, we renamed the Virtual Sensor Access layer from our previous architecture to the Virtualized WSN Infrastructure Management layer to make it more appropriate for this work. It now contains several new entities in addition as well as SAs. SAs interact with the WSN PaaS components on behalf of the VS in order to provide platform independence. The additional entities are described as follows.

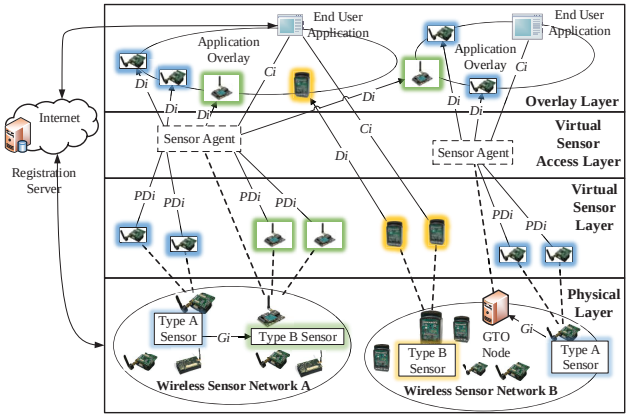


Fig. 1. Original vWSN Architecture

The Sensor Description Repository contains all relevant information about the deployed sensors, including their type, properties (i.e., protocol, data format, supported sampling intervals, physical location and supported units) and capabilities (i.e., sensing abilities). The repository can be distributed or centrally located and it is the responsibility of the WSN infrastructure owner to keep it up-to-date.

The Sensor Discovery entity, interacts with the repository to search for the required sensors using any criteria, e.g., sensor type, location and its availability.

The VS Provider is the main entity that receives VS creation requests from the WSN PaaS. Based on these requests, sensors are selected from the repository. The VS Provider also makes decision about when to create, start, or stop a VS by communicating with the VS Manager. There is also a small cache of the most recent sensors used by applications to prevent the need to search for sensors every time a request comes from the PaaS.

The VS Configurator entity prepares task codes based on the requests received from the VS Provider. These tasks will be run by a given VS. VS Configurator uses platform-specific code templates that allow for configurable parameters. A code template is a skeleton code file that does nothing useful on its own but can read from a parameter list and run a desired task. An example is the skeleton code that reads a manifest file (i.e., used in Java SunSpot platform) to initialize parameters such as sensor type, sampling interval, desired unit, and an end-point address to send data output. Creating these manifest files on the fly is programmatically simple and can be easily achieved.

The VS Configurator should ideally be implemented in a modular fashion to allow for the possibility of adding future code templates when new types of sensors are deployed. Additionally, VS Configurator compiles and generates the final executable code (e.g., jar file for Java SunSpot).

The role of VS Scheduler entity is to create, start, stop, and disseminate task codes either right away or at a later time, depending on the application requirements. It interacts with VS Manager to accomplish this.

The final layer is the Cloud Management layer, which includes an entity called the IaaS Access/Control Interface. This interface exposes a RESTful API that allows multiple users (i.e. PaaS) to interact with the deployed vWSN IaaS through a set of REST-based operations.

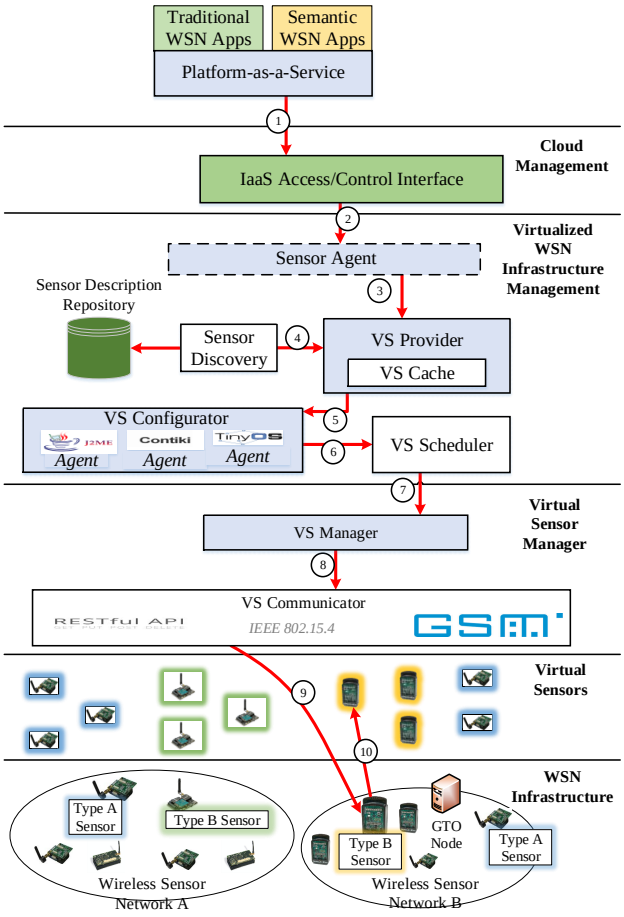


Fig. 2. Proposed vWSN IaaS Architecture

IV. An Early Implementation and Results

In this section, we first discuss a simple scenario used for implementation. Then, we present our implementation choices and prototype setup. Next, we discuss performance metrics and finish off this section with a discussion on the results.

A. Implementation Scenario

A smart home application is required that allows home owners to configure the use of their appliances when environmental conditions change. For example, the A/C should start automatically when temperature exceeds a given threshold. Similarly, the deck lights should be turned-on automatically when natural light drops below a given threshold.

The developer first discovers the light and temperature services to design and create the smart home application. When the application is deployed, the PaaS allocates an application container along with two REST-Based interfaces. One interface is for the VS corresponding to the light sensor and the other for the VS corresponding to the temperature sensor.

B. Implementation Choices and Prototype Setup

The WSN infrastructure consists of Java SunSpots, which have multiple on-board sensing capabilities. Unlike the earlier generation of sensor nodes, Java SunSpots are quite capable and are based on Java 2 Micro Edition (J2ME), which makes them easier to program. The Squawk VM supports multi-threading, making them suitable for our work. We used two Java SunSpot kits: two base station nodes and four SunSpots with on-board sensors. The vWSN IaaS layers were implemented as a standalone application.

We programmed a simple PaaS, as a standalone Java application. Eclipse IDE and JDK 1.7 were used for the application development. The application code was annotated with a description of the VS services and was given to the developers beforehand. The smart home application was developed as a simple Java application.

We used two laptops for the prototype. The first one had the PaaS, and the second one had the vWSN IaaS. The two laptops were connected via Ethernet and established as a LAN network. The vWSN IaaS laptop was connected to the Java SunSpot base stations to communicate with the remote SunSpots Over-the-Air (OTA).

C. Performance Metrics

The performance of the prototype was assessed in terms of the following metrics: VS Creation Delay (VSCD) and VS Start Time (VSST). The time spent between the moment the developer sends the application code to the PaaS for deployment and the moment the PaaS sends the creation requests to the vWSN IaaS was found to be negligible.

VSCD is the time spent between the moment the WSN infrastructure receives the VS creation request from the PaaS and the moment the VS is successfully created. Because it is required to create a shared base station instance to communicate with remote Java SunSpot OTA, we measured two types of VSCD. In the first type, the shared base station instance is created once and used repeatedly for VS creation, hence it only shows VS creation delay. In the second type, a shared base station is created every time a VS creation request is received from the PaaS, hence it shows VS creation delay plus the delay to create the shared base station instance.

VSST is the time spent when the WSN infrastructure receives the VS start request from the PaaS and when the corresponding VS is successfully started. All experiments were repeated 50 times with a confidence interval of 95%.

D. Results

Fig. 3 shows the values of both types of VSCD over 50 iterations. On average, it took about 14.973 seconds to create a VS on a remote Java SunSpot when the shared base station was created once. However, for the second type of VSCD, the average value increased by around 62%, to 24.282 seconds. One reason for this increase is that the shared base station instance spends some time probing for the available remote SunSpots. This delay is unavoidable and is not related to our architecture. Another reason for both of these high values is the fact that we used Ant build tool (as required by Java SunSpot platform) to first build, compile and create the executable file

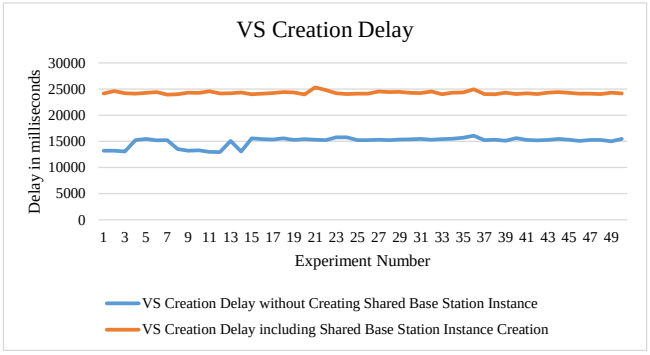


Fig. 3. VS Creation Delay

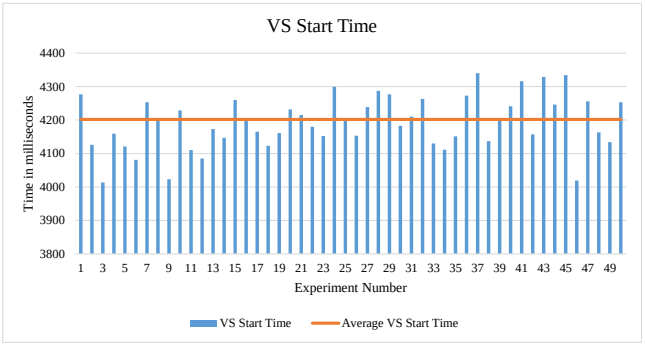


Fig. 4. VS Start Time

and then send it to remote Java SunSpots OTA. The last step included the delay to synchronize the target Java SunSpot. The actual dissemination of the VS code to the remote SunSpot took the very less time.

Fig. 4 shows the VSST of the 50 iterations. On average, it took 4.2 seconds to start the newly created VS after receiving the request from the PaaS. Again, this delay included the remote SunSpot synchronization delay before the newly created VS was started. Overall, these results are promising and prompt us to explore the problem area further in order to provide more optimized solutions.

V. Lessons Learned and Future Work

In this work, we have learned several lessons and have also identified many research issues to further pursue.

The *first* lesson learned is that while RESTful interfaces provide an easy way to access VSs, however, integrating them with existing open source PaaS (e.g., CloudFoundry) will be quite challenging. The *second* lesson is that there are other capable sensor kits in addition to Java SunSpots, such as Preon32 sensor kits from Virtenio GmbH [26] (Java-based and similar to SunSpots) and Phidgets kit [27]. The *third* lesson learned is that during the creation of VS on a Java SunSpot, the execution of existing VSs is not disturbed. This feature is very useful for ensuring that existing applications do not suffer when new ones utilize a SunSpot. Similarly, the VS migration feature is also supported by SunSpots, and we

intend to work on this in the future. The *fourth* lesson is that the delay associated the creation of VSs will largely depend on the platform. Java SunSpots need Ant build tool whose performance heavily depends on the installed Java version and the workload on the host machine.

As for the future work, *first* we plan to work towards the complete implementation of the architecture as presented in Section III-B and satisfy all the requirements mentioned in Section II-C. To this end, we intend to incorporate additional sensor platforms to allow for the heterogeneity of sensor nodes. The Preon32 and Phidgets kits are two possible candidates. *Second* we plan to work on exploiting the capabilities of available Java SunSpot kits by implementing the full features (e.g., VS stop, delete and migration to another remote SunSpot on the fly) they offer. *Third* we plan to provide the VS reservation mechanism by implementing a VS Scheduler entity, which would be very useful for a business model wherein a vWSN IaaS could be leased to users against certain incentives [28].

While this work focuses on the vWSN IaaS, we also felt the need to have a capable PaaS for vWSNs IaaS, because existing PaaS solutions do not consider the possibility of using VSs for application and service provisioning. For example, there is a need to discover and manage VSs and their details at the PaaS level but currently there is no solution for this. Instead most solutions simply receive sensor data and use it without taking full advantage of a vWSN IaaS.

VI. Conclusion

In this paper we have presented an architecture for a competent vWSN IaaS that is able to interact with the PaaS to support the concurrent VS-based applications and services deployment on-demand. The architecture uses the principles of cloud computing and the basics of WSN virtualization to offer WSN deployments as IaaS. Using a capable sensor kit, an early implementation has demonstrated its feasibility. We have also identified several interesting and potent research issues and plan to tackle them in future contributions.

Acknowledgment

This work was supported in part by the Natural Science and Engineering Council of Canada (NSERC) Canada Research Chair in End-User Service Engineering for Communications Networks and by an NSERC Discovery Grant.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393422, Mar. 2002.

[2] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy, "Smart objects as building blocks for the Internet of things, *IEEE Internet Computing*, vol. 14, no. 1, pp. 4451, Jan. 2010.

[3] M. A. Feki, F. Kawsar, M. Boussard, and L. Trappeniers, The Internet of Things: The Next Technological Revolution, *Computer*, vol. 46, no. 2, pp. 2425, 2013.

[4] L. M. Vaquero, L. Rodero-Merino, J. Caceres and M. Lindner, "A break in the clouds," *ACM SIGCOMM Computer Communication Review*, vol. 39, pp. 50, 2008.

[5] A. Khan, A. Zugenmaier, D. Jurca and W. Kellerer, "Network virtualization: a hypervisor for the Internet?," *IEEE Communications Magazine*, vol. 50, pp. 136-143, 2012.

[6] P. Levis and D. Culler: "Mat: A tiny virtual machine for sensor networks, *In ASPLOSX: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, 2002, pp. 85-95.

[7] A. P. Jayasumana, et al., "Virtual sensor networks a resource efficient approach for concurrent applications, *In Proc. Information Technology, 2007. ITNG07. Fourth International Conference on*, Las Vegas, 2007, pp. 111-115

[8] I. Khan, et al., "Wireless Sensor Network Virtualization: A Survey," *Communications Surveys & Tutorials, IEEE*, vol.PP, no.99, pp.1,1 2015, doi: 10.1109/COMST.2015.2412971

[9] I. Leontiadis, et al. "SenShare: transforming sensor networks into multi-application sensing infrastructures, *Wireless Sensor Networks*, Springer Berlin Heidelberg, 2012, pp. 65-81.

[10] I. Khan, et al., "Wireless Sensor Network Virtualization: Early Architecture and Research Perspectives", Accepted, *IEEE Network*. 2015

[11] I. Khan, et al., "A Data Annotation Architecture for Semantic Applications in Virtualized Wireless Sensor Networks", *in proceedings of 14th IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2015) Technical Session*, 2015, May 11-15, Ottawa, Canada.

[12] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292-2330, Aug. 2008.

[13] R. C. Carrano, D. Passos, L. C. S. Magalhaes and C. V. N. Albuquerque, "Survey and Taxonomy of Duty Cycling Mechanisms in Wireless Sensor Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 181-194, 2014.

[14] N. Pantazis,, S. Nikolidakis, and D. Vergados. "Energy-efficient routing protocols in wireless sensor networks: A survey," *Communications Surveys & Tutorials, IEEE*, 15.2 (2013): 551-591.

[15] A. Gyrard, C. Bonnet, and K. Boudaoud, "A machine-to-machine architecture to merge semantic sensor measurements," in *WWW 2013, 22nd International World Wide Web Conference, Doctoral Consortium*, Rio de Janeiro, BRAZIL, 2013.

[16] H. Alemdar and C. Ersoy, "Wireless sensor networks for healthcare: A survey," *Computer Networks*, vol. 54, pp. 2688-2710, 2010.

[17] J. Rodrigues, and P. ACS Neves. "A survey on IPBased wireless sensor network solutions," *International Journal of Communication Systems* 23.8, pp. 963-981.2010

[18] X. Wang and H. Qian, "Research on all-IP communication between wireless sensor networks and IPv6 networks," *Computer Standards & Interfaces*, vol. 35, pp. 403-414, 2013.

[19] Hui, J., Ed., and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", *RFC 6282*, September 2011.

[20] Z. Shelby, "Embedded web services," *IEEE Wireless Communications*, vol. 17, pp. 52-57, 2010.

[21] M. Kamio, T. Yashiro, and K. Sakamura. "6LoWPAN framework for efficient integration of embedded devices to the Internet of Things," *Consumer Electronics (GCCE), 2014 IEEE 3rd Global Conference on. IEEE*, 2014. in 2014, pp. 302-303.

[22] V. Medina and J. M. Garca, "A survey of migration mechanisms of virtual machines," *ACM Computing Surveys*, vol. 46, pp. 1-33, 2014.

[23] J-Y., Hwang et al. "Xen on ARM: System virtualization using Xen hypervisor for ARM-based secure mobile phones," *Consumer Communications and Networking Conference. 2008, 5th IEEE*. pp. 257-261.

[24] D. Simon, 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.

[25] R. Lopes, F. Assis, and C. Montez. "MASPOT: A mobile agent system for Sun SPOT., *In Autonomous decentralized systems (ISADS), 2011 10th international symposium on*, pp. 25-31. IEEE, 2011. Kobe, Japan

[26] Preon32 sensor kit – http://www.virtenio.com/en/products/evaluations-kits.html - [accessed 20-04-2015]

[27] Phidgets kit – http://www.phidgets.com/products.php?category=18 [accessed 20-04-2015]

[28] Y. Zhang, J. Wen, "An IoT electric business model based on the protocol of BitCoin", *in 2015 18th Int. Conf. Intelligence in Next Generation Networks: Innovations in Services, Networks and Clouds (ICIN 2015)*, Paris, France, Feb. 2015.

Annex D

Paper IV

Design and Analysis of Virtualization Framework for Wireless Sensor Networks

Imran Khan
Institut Minés-Télécom, Télécom SudParis,
91011 Evry Cedex, France
imran@ieee.org

***Abstract*—Wireless Sensor Networks (WSNs) are used in many application areas including health, agriculture and gaming. New advances in sensor technology make it pertinent to consider sharing a deployed WSN infrastructure by multiple applications, including applications which are designed after the WSN deployment. For my PhD research I propose a novel WSN virtualization framework that allows multiple users to run their application tasks over underlying WSN resources in a transparent way. This paper presents the overview of the proposed WSN virtualization framework, related work, current status and future work.**

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

I. Introduction

Wireless Sensor Networks (WSNs) are combinations of micro-electro-mechanical systems, wireless communication systems and digital electronics nodes that sense, compute and communicate [1]. Up till now the real world deployments of WSNs have been tailor-made solutions where applications are bundled with a WSN at the time of deployment with no possibility for other applications to re-use the deployed WSN. Virtualization is a technique that presents physical resources logically and enables their sharing and efficient usage [2]. The new generations of sensors [3] encourage us to consider sharing them using virtualization.

WSN virtualization is a relatively new field and to the best of our knowledge there is no mechanism to discover and publish WSN resources for multiple, independent applications allowing them to access these resources concurrently according to their requirements. Also there are no WSN oriented protocols for signaling, resource reservation and management for this purpose. Overlays can be used to have multiple applications access the WSN resources but they require careful analysis because of the inherit WSN constraints.

The main contribution of this PhD work is to provide a platform independent WSN virtualization framework consisting of *i)* a general architecture for WSN virtualization, *ii)* a dynamic discovery and publication framework, *iii)* a middleware independent overlay protocol suitable for resource constraint WSN nodes, and *iv)* a resource efficient signaling protocol for resource reservation and session management. None of the existing works deals with these issues.

The rest of the paper is organized as follows. Related work is presented in Section II. Overview of the proposed WSN virtualization framework is presented in Section III. Current status and future directions are discussed in Section IV and Section V concludes the paper.

II. Related Work

In literature, the existing WSN virtualization solutions can be classified into three categories, node level virtualization [4], [5] network level virtualization [6] and hybrid solutions [7], [8]. Our work considers both node and network level virtualization, hence it is pertinent to compare it to the hybrid solutions only. The authors in [7] provide a platform dependent solution for WSN virtualization. The proposed solution works on specific OS and hardware. Each application program uses a hardware abstraction layer (HAL) to access underlying WSN resources, which means that the developer needs to be aware of the HAL, which in turn depends on the OS. The solution in [8] is one of the pioneering work but falls short of true WSN virtualization philosophy. The applications are preconfigured and decided before the deployment of the WSN. It is not possible to include new applications afterwards. In [9] a software architecture FLEXOR is presented which provides optimal implementation, and evaluation of protocols with focus on reusability, QoE and user friendliness in WSN development cycle. This platform, however, does not discuss WSN virtualization. There is no discussion on how multiple applications, developed using FLEXOR, will use the underlying WSN resources concurrently.

III. WSN Virtualization Framework

A. Basic Principles

The first principle for WSN virtualization is that any new application or a service is deployed as a new overlay on top of the physical WSN. The second principle is that any given physical sensor can execute (locally) task(s) for a given application deployed in the overlay. Existing sensor kits such as Java SunSpot [3] and operating systems like Contiki [10] support concurrent execution of multiple applications. The third and final principle is that some sensor may not have enough capabilities to support the overlay middleware. When this is the case, they will delegate such operations to more powerful sensors and even to other nodes.

B. Proposed WSN Virtualization Framework

Figure 1 shows the proposed WSN virtualization framework. In this framework heterogeneous sensor nodes with varying

978-1-4673-5828-6/13/$31.00

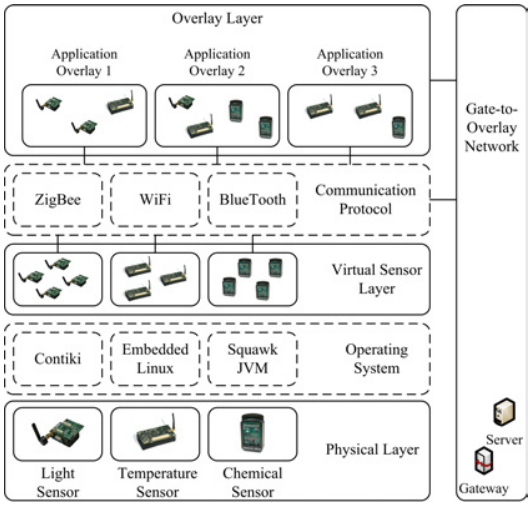


Fig. 1. WSN Virtualization Framework

sensing capabilities are considered. All sensor nodes have an OS that supports concurrent execution of the application tasks and can communicate using variety of communication protocols. There is a virtual sensor layer that consists of logical instances of sensors executing distinct application tasks. E.g. if a sensor executes two application tasks, it will have two virtual sensors each dedicated to a single task. The number of virtual sensors supported by a sensor node depends on its resources.

The end user applications create application specific overlays. The virtual sensors become part of these overlays as overlay nodes and participate in the execution of the end user applications as per the first principle of the framework. The creation of virtual sensors is according to the second principle. As for the final principle, there are certain nodes (e.g. gateways or servers) that enable resource deficient sensor nodes to join the application overlays. Such nodes are termed as Gate-to-Overlay (GTO) nodes making up a GTO network helping virtual sensors to join /leave application overlays.

IV. Current Status And Future Directions

A multi-layer general architecture for WSN virtualization is presented in [11]. At present an initial prototype is under development using Java SunSpot kit and JXTA middleware. Depending on the progress and acceptance of this extended abstract, initial results will be presented at the conference.

For future work, identified research issues include, a publication and discovery framework to allow different actors, including sensors, to publish and discover on the fly. This includes discovery of suitable sensors by the end user applications and discovery of GTO nodes by sensors to participate in the application overlays.

The second issue is to find suitable overlay protocols, especially as these protocols should be middleware-independent whenever possible. Another issue is to manage and prevent overlays from interacting in a harmful way as they compete for the underlying resources (WSN nodes in this case).

The third and final issue is regarding the signaling framework. There are several signaling frameworks like SIP/RSVP but they may not be suitable for resource-constrained devices. A CoAP [12] based signaling framework is a potential solution.

V. Conclusion

In this paper an overview of WSN virtualization framework and its related issues are presented and some key research issues are also identified. After proposing general architecture, currently work on initial prototype is in progress. WSN virtualization is an emerging area of research that can potentially help to realize the true potential of sensors. Much of this depends on the advancements in the sensor hardware technology but presently we have, in our hands, some capable devices that can be used to initiate research activities in this area.

Acknowledgment

This work is supervised by Prof. Noel Crespi, Institut Mines-Télécom, Télécom SudParis, France and Prof. Roch Glitho, CIISE, Concordia University, Canada. The author would like to thank Dr. Fatna Belqasmi for her comments.

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

[4] P. Levis and D. Culler: “Maté: A tiny virtual machine for sensor networks”, *In ASPLOSX: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, 2002, pp. 85-95.

[5] M. Navarro et al., “VITRO Architecture: Bringing Virtualization to WSN World”, *Mobile Ad-Hoc and Sensor Systems, IEEE 8th International Conference on*, Valencia, Spain, 2011, pp. 831-836

[6] A. P. Jayasumana, et al., “Virtual sensor networks a resource efficient approach for concurrent applications”, *In Proc. Information Technology, 2007. ITNG’07. Fourth International Conference on*, Las Vegas, 2007, pp. 111-115

[7] Leontiadis, Ilias, et al. “SenShare: transforming sensor networks into multi-application sensing infrastructures”, *Wireless Sensor Networks*, Springer Berlin Heidelberg, 2012, pp. 65-81.

[8] Y. Yu et al., “Supporting concurrent applications in wireless sensor networks”, *4th International Conference on Embedded Networked Sensor Systems, SenSys06*, Boulder, Colorado, 2006, pp.139-152

[9] Forster, Anna, et al. “Flexor: User friendly wireless sensor network development and deployment”, *World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2012 IEEE International Symposium on a. IEEE, San Francisco, CA, 2012, pp. 1-9

[10] Dunkels, Adam et al., “Contiki-a lightweight and flexible operating system for tiny networked sensors”, *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*. IEEE, Tempa, FL, 2004, pp. 455-462.

[11] Khan, Imran, et al., “A Multi-Layer Architecture for Wireless Sensor Network Virtualization”, in *6th Joint IFIP Wireless and Mobile Networking Conference (WMNC’13)*, April, 23-25, Dubai, UAE, 2013. To appear.

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

Annex E

Paper V

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 ©2013 IEEE

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.

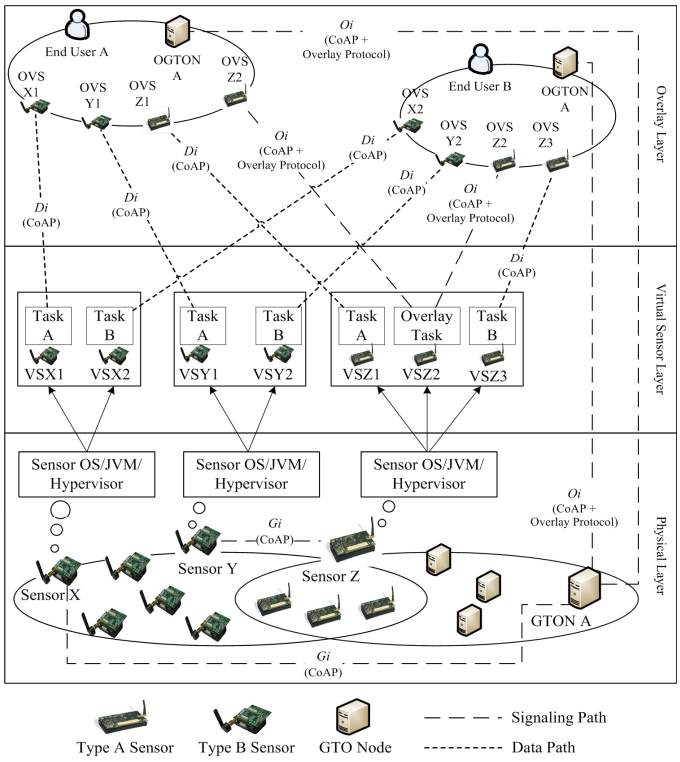


Figure 1. General architecture

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.

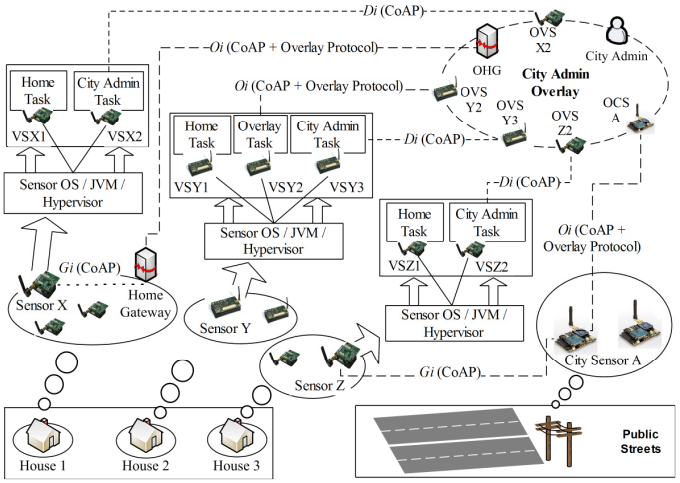


Figure 2. Fire monitoring problem

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.

Annex F

Paper VI

A Data Annotation Architecture for Semantic Applications in Virtualized Wireless Sensor Networks

Imran Khan*, Rifat Jafrin†, Fatima Zahra Errounda†, Roch Glitho†, Noël Crespi*,
Monique Morrow‡ and Paul Polakos‡
*Institut Minés-Télécom, Télécom SudParis, 91011 Evry Cedex, France
Email: imran@ieee.org, noel.crespi@it-sudparis.eu
†Dept. CIISE, Concordia University, H3G 2W1, Montreal, Canada
Email: {r_jafri, f_errou} @encs.concordia.ca, glitho@ciise.concordia.ca
‡CISCO Systems, Inc.
Email: {mmorrow, ppolakos} @cisco.com

***Abstract*—Wireless Sensor Networks (WSNs) have become very popular and are being used in many application domains (e.g. smart cities, security, gaming and agriculture). Virtualized WSNs allow the same WSN to be shared by multiple applications. Semantic applications are situation-aware and can potentially play a critical role in virtualized WSNs. However, provisioning them in such settings remains a challenge. The key reason is that semantic applications' provisioning mandates data annotation. Unfortunately it is no easy task to annotate data collected in virtualized WSNs. This paper proposes a data annotation architecture for semantic applications in virtualized heterogeneous WSNs. The architecture uses overlays as the cornerstone, and we have built a prototype in the cloud environment using Google App Engine. The early performance measurements are also presented.**

***Keywords*—Wireless Sensor Networks; Semantic Web; Domain Ontologies; WSN Virtualization; Data Annotation; Overlays**

I. Introduction

Wireless Sensor Networks (WSNs) [1] consist of small-scale devices that allow applications to observe various physical phenomenon and then react to the reported events. However, WSN deployments are usually tailored for predefined applications with no possibility for new applications to use them concurrently. To address this, WSN virtualization that uses the concept of concurrent application tasks running on a sensor node and combines such nodes together to work for multiple applications simultaneously has gained considerable attention [2]–[4]. We have recently proposed an early architecture as a solution for WSN virtualization [5].

Typically, virtualized WSNs provide sensor data in raw format. However, classical WSN applications cannot interpret the raw sensor data and understand its context. This makes it almost impossible for their users to get the high-level details of the events and infer additional knowledge to gain situational awareness. For example, a fire monitoring application can only receive a simple fire notification without additional details for its end-user to understand the meanings and context of the fire event, e.g. event status and its location.

Semantic applications, on the other hand, allow their users to make queries such as *what is the current status of the fire?* and *what is the current location of the fire?* to get results like *initial fire* and *in a public library* respectively. Virtualized WSNs typically monitor several real-time events at the same time for different applications. Hence, some end-users of these applications may wish to know the context of specific events. This brings us to the need for a mechanism that annotates the sensor data in a virtualized WSN. Annotating sensor data in virtualized WSNs is quite challenging; since resources are scarce,virtual sensors are created on-demand and may have unpredictable lifetime.

In order to provision semantic applications we need to send additional metadata along with raw sensor data. For example, the raw sensor data for a fire monitoring application can be annotated with concepts such as *observed property* and *location*, which are temperature and, longitude and latitude, respectively, in this case. Semantic annotation has been a popular approach for this purpose. It is defined as a metadata generation and usage schema that can be used to provide new methods, as well as to extend existing ones, to access new information [6]. However, the semantic annotation process requires domain concepts and the relationships that exist between them in order to annotate data. An ontology is used to formally represent a domain, its concepts and the relationships that exist [7]. Within sensor domain, there are several efforts to develop ontologies, e.g., the Semantic Sensor Network (SSN) Ontology developed by the W3C Semantic Sensor Network Incubator Group [8] and SensorML from the Open Geospatial Consortium (OGC) [9]. SSN ontology is more general purpose because it is application domain independent and provides concepts about sensors and their observations.

This paper proposes a data annotation architecture for semantic applications in virtualized WSN environments. We extend our previous WSN virtualization architecture [5] to cater for data annotation. We develop a base ontology by extending the SSN ontology. We also develop a domain ontology for the semantic application we have prototyped. The fire monitoring semantic application receives annotated data and uses the fire domain ontology, along with a reasoner, to infer knowledge. An end-user can query over the annotated data to get the real-time information of the fire event, such

as its status and location. The application is developed and deployed in the cloud using Google App Engine (GAE) and works in a heterogeneous virtualized WSN environment.

The rest of the paper is organized as follows: A motivating scenario is presented in Section II, along with a set of requirements. We discuss our proposed architecture in Section III, followed by our procedures and illustrative scenario in Section IV. The prototype implementation and results are discussed in Section V and an overview of the related work in Section VI. Section VII presents lessons learned along with the future work and Section VIII concludes the paper.

II. Motivating Scenario and Requirements

In this section, we first present a motivating scenario and then derive a set of requirements from it.

A. Motivating Scenario

We extend the motivating scenario presented in [5] for a semantic application that monitors fire events in real time.

Consider a city near an area where brush fires are common and where some houses already have their own sensors to detect fire. The city administration is interested in using WSNs for the early detection of brush fire events as well as to monitor their course. To accelerate the deployment of their new application and to avoid redundancy, the city administration has opted to deploy sensors in areas under its jurisdiction (i.e. streets and parks) and to re-use the WSN nodes already deployed in private homes. These sensors have several sensing capabilities, such as temperature, humidity, CO2 and dust levels. They also execute multiple tasks (thanks to WSN virtualization), some of which may belong to semantic applications. The sensors, executing these tasks, provide annotated data for several semantic applications.

This sensor deployment can be utilized for several semantic applications. For example, the city administration's application can provide detailed information about fire events to its users, rather than simple notifications. Another example is of a weather applications that can use the same annotated data to identify prevailing weather condition such as sunny, haze, partial cloudy and snow. Similarly, a smart parking application could use the same annotated data to determine the current pollution levels and dynamically change the parking fee accordingly. For example, when the pollution level is very high, parking could be offered at a discount or even free.

B. Requirements

Based on the scenario described above, we derive the following six requirements. *First*, the proposed architecture should allow for the real-time annotation of sensor data. This means that the sensor data should be annotated before sending it to the semantic applications. The *second* requirement is that the base ontology should be stored in the WSN in a distributed manner, since it will be used to annotate the sensor data. The *third* requirement is that the annotation should be done in a distributed manner without relying on a central node. This ensures that any node failure does not affect the annotation process. The *fourth* requirement is that it should be possible to enhance or to extend the ontology so that new concepts can be integrated with the existing ones. The *fifth* requirement is that the proposed solution should be applicable to heterogeneous sensor platforms and the data formats that they use, to ensure interoperability. The *sixth and final* requirement is the use of standardized ontologies, so that all semantic application can use standard concepts.

III. Proposed Architecture

In this section, we begin by discussing our previous architecture, since we use it as the basis for this work. Next, we present the architectural principles, followed by the details of layers and functional entities of the proposed architecture. Finally we present the base ontology that we used for sensor data annotation.

A. Our Starting Point

The work in this paper is based on our previous WSN virtualization architecture [5] which is illustrated in Fig. 1. The architecture consists of four layers. The physical layer consists of sensor nodes that can run several application tasks simultaneously. Two types of sensor nodes are considered in the architecture. Type A sensors are resource-constrained sensors that have very limited processing and storage capabilities, e.g. TelosB motes. Type B sensors have better processing and storage capabilities, e.g. Java SunSpots. Since Type A sensors may not be capable enough to work together with other sensors in a group, they rely on more powerful nodes called Gate-to-Overlay (GTO) nodes for this purpose.

The virtual sensor layer abstracts the simultaneous tasks run by the physical sensors as virtual sensors. In this paper we use the terms virtual sensors and sensors interchangeably for consistency. To provide platform independence, the virtual sensor access layer consists of Sensor Agents (SAs). This independence is achieved by using standardized north-bound interfaces and proprietary south-bound interfaces. The final layer consists of application overlays that run simultaneously on top of the physical layer. There are separate interfaces for data and control messages. Overall, the architecture provides the flexibility of using multiple applications concurrently over WSN deployments.

B. Architectural Principles

The *first* architectural principle is that the ontology used to annotate the sensor data is separated as base and domain ontologies. The former consists of concepts related to the deployed sensors and their capabilities, and is stored in the WSN, while the later consists of domain-specific, application-related concepts and is typically stored in the application domain. This basic principle allows the solution to become independent of any application domain.

The *second* architectural principle is that we use two independent overlays: one for data annotation and the other for storing the base ontology. Overlays have several advantages: they are distributed, they do not rely on centralized control and they allow resource sharing [10].

The *third* architectural principle is that every virtual sensor created for semantic application is represented in the annotation overlay by a corresponding entity that annotates its data.

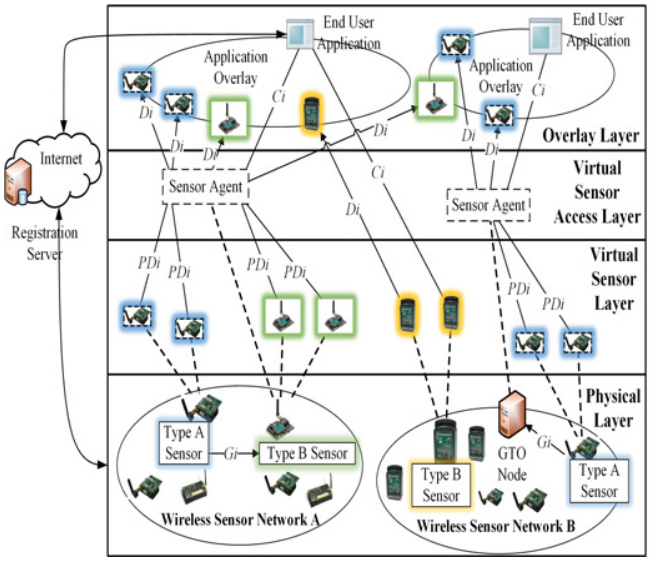


Fig. 1. WSN virtualization architecture

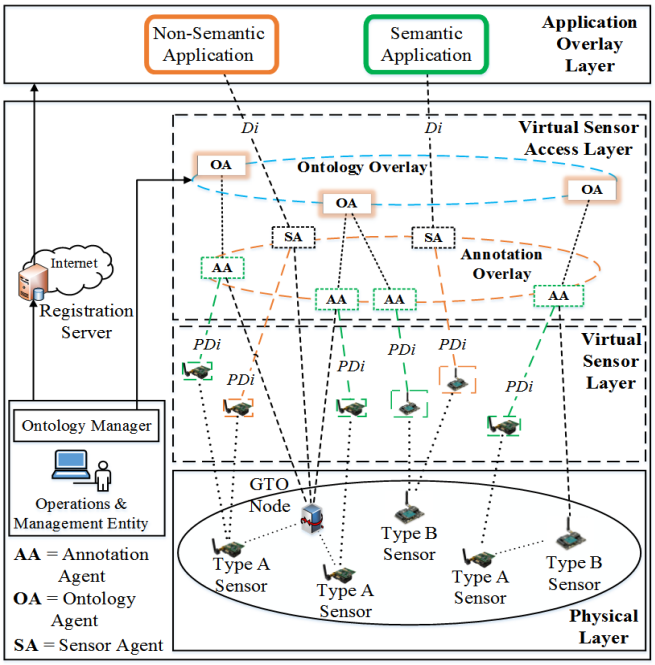


Fig. 2. Proposed data annotation architecture

This means that every sensor sending data to semantic applications will have a dedicated entity for annotation purposes.

The *fourth* principle is that, for resource constrained sensors, the annotations will be performed by capable sensors and other powerful nodes, e.g. gateways. This principle ensures that all kinds of sensors are available for the semantic applications.

C. Layers and Functional Entities

Fig. 2 shows the proposed architecture. It is based on our previous WSN virtualization architecture, presented in Section III-A. The physical layer remains the same while the virtual sensor layer now consists of two types of virtual sensors. The first group are the virtual sensors that are created for semantic applications, referred to hereafter as semantic virtual sensors. They are indicated as green-dashed boxes. The second type of virtual sensors are created for non-semantic applications, referred to hereafter as virtual sensors. These are shown as orange-dashed boxes. The difference between these two types of virtual sensors is that the raw sensor data from the green-dashed ones will be annotated before being sent to end-user semantic applications. The virtual sensor access layer has two new functional entities and two overlays. The functional entities are Annotation Agents (AAs) and Ontology Agents (OAs). We term an agent as an entity that provides a given functionality, therefore several agents are used in our architecture. The Annotation overlay consists of AAs, which annotate sensor data using the base ontology. They communicate with Sensor Agents (SA) in the same overlay to send the annotated data to the semantic applications. The Ontology overlay consists of OAs, which are responsible for storing the base ontology in a distributed manner. The OAs act as super-peers and provide the requested ontology to the AAs. They do not deal with the sensor data.

The architecture supports both semantic and non-semantic applications. The Operations & Management (O&M) entity, which is usually the infrastructure owner, is responsible for providing the base ontology. Since O&M entity is aware of the type of sensors deployed in the WSN, it can easily develop and disseminate the base ontology to the ontology overlay.

The architecture does not deal with the sensor discovery mechanism and storage of sensor data in a repository for data analysis. For the former, existing work such as [11], [12] can be reused. In this work we assume that the sensors have already been discovered and are stored in a registration server. For the latter, we leave it to the applications to decide on the sensor data storage since it is an application specific requirement.

The proposed architecture fulfills the set of requirements mentioned in Section II.B. AAs allow real-time annotation of sensor data in a distributed manner. OAs store the common ontology and are distributed using the concept of overlays. The base ontology can be extended by creating additional OAs. The architecture is platform-independent thanks to the SAs. As we use and extend SSN ontology in our work, the final requirement is also fulfilled.

D. Base Ontology

We have built our base ontology by extending the SSN ontology, since it is quite well-known and widely used to describe sensors and their data. As mentioned before, the goal of having a base ontology is to add metadata to the raw sensor data before it is used by a particular application. We assume that the WSN consists of temperature, humidity, light and carbon sensors and thereby incorporate these type of sensors and their observations in the base ontology. Fig. 3 shows the part of the base ontology, related to temperature sensors.

IV. Procedures and Illustrative Scenario

In our architecture we need different procedures related to the management and operational aspects of the annotation and ontology overlays. The management procedures include the

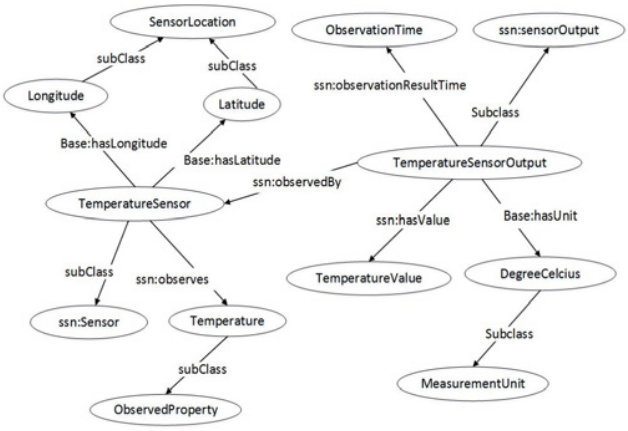


Fig. 3. Temperature sensor part of the base ontology

following. 1) Selection of sensors and GTO nodes that will play the role of *i)* OAs in the ontology overlay, and *ii)* AAs in the annotation overlay. 2) The distribution of the ontology over the OAs. These procedures are motivated by our architectural principles mentioned in Section III.B.

The annotation process requires the ontology, which may not be available with the AAs. This situation calls for an ontology discovery procedure to allow the AAs to annotate the sensor data. The operational procedures include the ontology discovery and the sensor data annotation.

A. Management Procedures

According to the first and second architectural principles, we store the base ontology in the WSN using the concept of overlays, i.e. in the ontology overlay. The ontology overlay consists of OAs that require sufficient storage space and an efficient request/response mechanism. There are two types of nodes that can act as OAs: GTO nodes, which store the complete base ontology, and Type B sensors, which store part of the base ontology.

According to the third architectural principle, each sensor is represented by a corresponding AA in the annotation overlay. However, the role of an AA requires certain capabilities for computational-intensive tasks, such as the mapping sensor data to the base ontology concepts and generating output files. However, not all sensors are capable of performing these tasks, especially the ones that have 16-bit processors and memory on order of KBs, e.g., TelosB. For these sensors either Type B sensors or GTO nodes can act as AAs on their behalf, in accordance with the fourth architectural principle.

According to the second architectural principle, the base ontology needs to be distributed. The following mechanism is used for the distribution. GTO nodes contain the complete base ontology, while Type B sensors only contain the parts of the base ontology, related to phenomena that they sense. For example, a Type B sensor with temperature sensing capability will only contain the temperature portion of the base ontology. In order to accomplish this distribution, the GTO nodes split the base ontology into multiple parts and send it to the relevant Type B sensor. The common ontology concepts are present in each part. It is important to remember that since sensors are prone to failure, it makes sense to have the same parts of the base ontology present in multiple Type B sensors.

Both the GTO nodes and the Type B sensors can be selected for the roles of AAs and OAs. However, the OAs in the GTO nodes contain the complete base ontology, while the OAs in Type B sensors only contain the part of the ontology they require for annotation.

B. Operational Procedures

The first operational procedure is the ontology discovery. There are two possible approaches, pro-active and reactive. In the pro-active approach OAs, as super-peers, periodically advertise the base ontology parts that they have. The AAs then send their ontology requests in response to these advertisements. In the reactive approach, AAs first determine the sensing capabilities of the corresponding sensors, based on which they send discovery request to their super-peers, for the required part of the base ontology.

The second operational procedure is the data annotation, which works as follows. The semantic virtual sensors send their data in a standardized or proprietary format to the AAs. Once an AA receives the raw sensor data, it first checks locally if it has the required ontology to annotate it, if not, a discovery request is sent to the ontology overlay. When it has the required ontology, the AA annotates the raw sensor data, and sends it to the SA. The SA is then responsible for sending the annotated data to the semantic application.

C. Illustrative Scenario

The city administration and home owners deploy fire detecting sensors in public streets and in private homes, respectively. These sensors run multiple application tasks concurrently, using virtual sensors and semantic virtual sensors. The semantic virtual sensors send annotated data to the fire monitoring semantic application. The application receives this data and uses a reasoner to infer knowledge and to get detailed information about fire events.

The annotation process works as follows (a sequence diagram is presented in Fig. 4). Semantic virtual sensors send their raw data in a standardized or proprietary format to the AA. Once an AA receives the raw sensor data, it first checks locally to determine if it has the required ontology to annotate the data, if not it sends request message to an OA for the required ontology. The OA may request another OA for the required ontology if it does not store it. Once the ontology is retrieved, it is sent to the AA, which then annotates the raw sensor data using the received ontology and sends it to the SA. The SA sends the annotated data to the appropriate semantic application. The semantic application applies the domain ontology and a set of rules using a reasoner to infer additional knowledge. If a fire event is detected then a notification is sent to the end-user. The end-user may query for additional information such as fire status and location. In Fig. 4, the end-user queries for the fire status and receives the response, i.e. initial fire.

V. Prototype Implementation and Results

In this section we present our prototype in detail. First we discuss the implementation choices we made, and then we

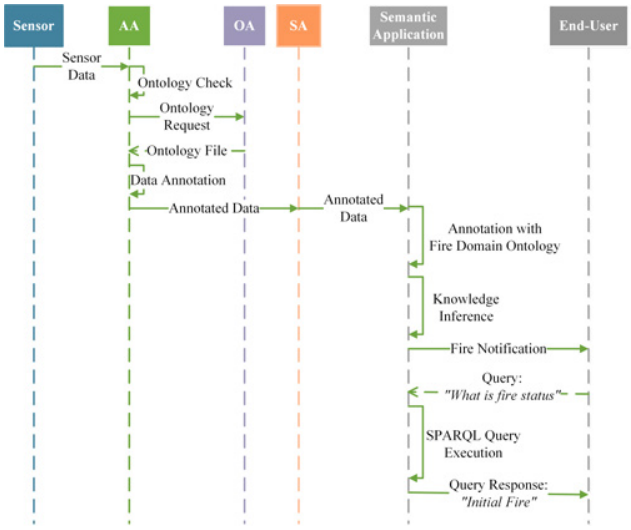


Fig. 4. Sequence diagram of the illustrative Scenario

present our prototype setup and the performance metrics. We end this section with a discussion of the results.

A. Implementation Choices

We developed a fire monitoring semantic application for our prototype based on the scenario presented in Section II.A. The application is offered as Software as a Service (SaaS) to the end-users. It was developed using the Apache Jena Framework, which is an open source Java framework for building semantic web and linked data applications. The application was deployed in a cloud-based Google App Engine (GAE), which is a Platform as a Service (PaaS) that allows the development of SaaS applications without having to maintain a server. We chose GAE because it makes it easy to deploy and maintain applications. The annotation and ontology overlays are implemented using the JXTA [13] protocol, an open source peer-to-peer protocol specification that allows the creation of independent, robust and efficient overlay networks.

The fire monitoring semantic application is a RESTful web service that uses the following components:

1) Fire domain ontology: Contains the concepts of fire, its states, and sensing events along with their states, such as temperature (high, low), relative humidity (high, low) levels, CO2 (high, low) levels and location (city, park, and downtown). Fig. 5 shows some concepts of the fire domain ontology.

2) Jena Inference API: Used to reason over the annotated data and to infer additional knowledge using a set of rules. We developed several rules for our semantic application to provide information to end-user about the fire events. Two examples of rules are given below.

```
[Rule1: (?output ssn:hasValue ?Value)
greaterThan(?Value,80), (?output rdf:type
base:TemperatureOutput),
(?output base:hasUnit base:DegreeCelsius) ->
(?output fda:hasTemperatureType:
fda:HighTemperature) ]

[Rule2: (?output fda:hasTemperatureType
```

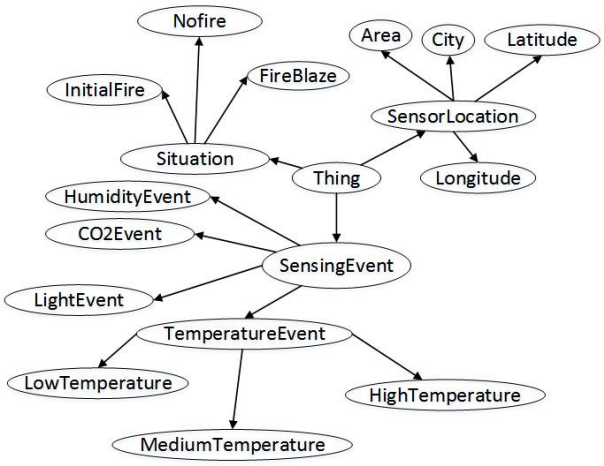


Fig. 5. Some concept of the fire domain ontology

```
fda:HighTemperature)
(?output fda:hasHumidityLevel fda:LowHumidity)
(?output fda:hasCO2Level fda:HighCO2)  ->
(?output fda:hasFireSituation fda: fireBlaze)]
```

3) Query Engine: Used to query annotated data. Below is an example query to get event information like event time, its value, location, and the status (fire event in this case).

```
SELECT  ?Time ?Temperature ?Longitude
?Latitude ?Firesituation
WHERE {
 ?SunSpotOutput base:hasSensingTime ?Time.
 ?SunSpotOutput ssn:hasValue ?Temperature.
 ?Sunspot base:hasLongitude ?Longitude.
 ?Sunspot base:hasLatitude ?Latitude.
 ?SunSpotOutput fda:hasFireSituation
 ?Firesituation.
 FILTER ( regex(str(?Firesituation),
 'http://www.semanticweb.org/WirelessSensor/
  FireApplication#FireBlaze', 'i' )
  }
```

The functional entity AAs are in annotation overlay and have the following components:

1) Web Server: Receives the sensor data;

2) JXTA Edge Peer: Participates in the overlay and request the required parts of the base ontology;

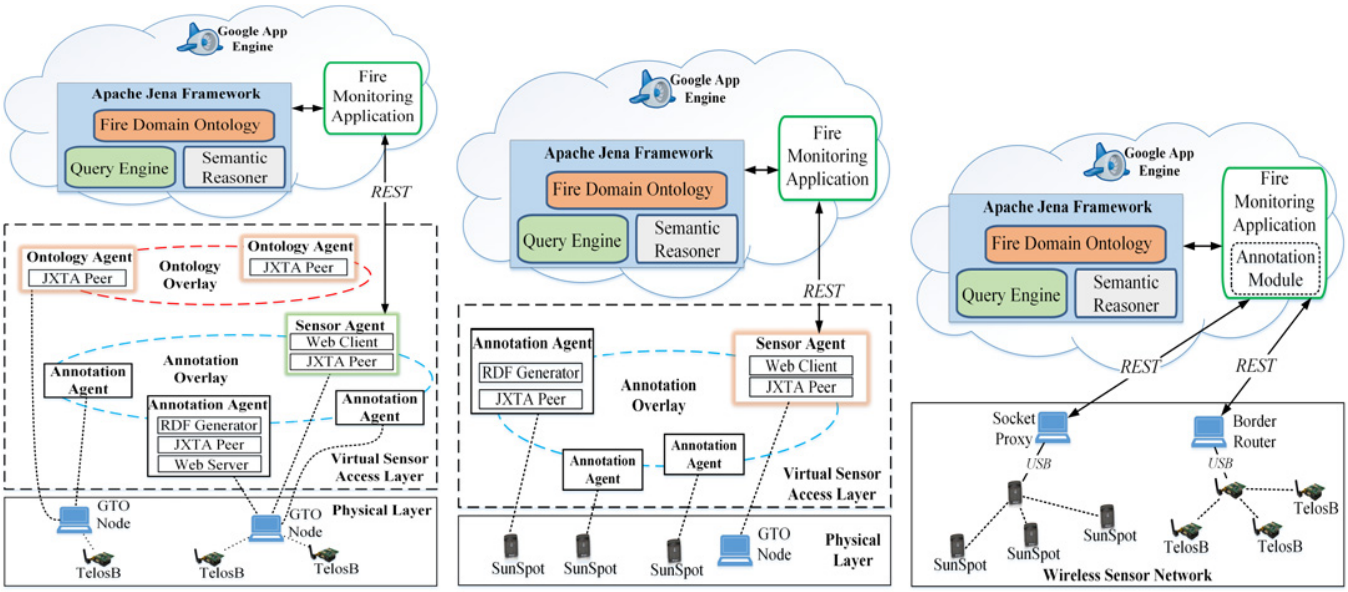
3) RDF Generator: Annotates sensor data using the base ontology; and

4) Web Client: Sends annotated data to semantic application.

The functional entity OAs are in the ontology overlay and have the following component:

1) JXTA Rendevous Peer: To store the base ontology and send it to the requesting AA. We used the JXTA Content Management System (CMS) to advertise the base ontology available in each OA and send it to the requesting AAs.

The proposed architecture is implemented as Infrastructure as a Service (IaaS), which allows us to link our solution to the IaaS, PaaS and SaaS aspects of cloud computing paradigm.



(a) Configuration A (b) Configuration B (c) Configuration C

Fig. 6. Implementation Architecture

B. Prototype Setup

We used two different sensor kits for the prototype, Java SunSpot and TelosB motes from AdvanticSys Kit. In total we used 6 SunSpots (2 as base stations), 4 TelosB motes (1 as border router) running Contiki OS. All these sensors have multiple on-board sensing capabilities but differ in their processing and storage abilities. In our implementation, TelosB motes are Type A sensors and Java SunSpots are type B sensors. All of the sensor were running multiple application tasks. The Java SunSpots had three application tasks running concurrently, periodically measuring temperature, light and blinking LEDs. The TelosB motes had temperature, light, and humidity tasks running concurrently. Type B sensors send their data in SenML [14] format, which is a lightweight standard data model which is suitable for sending sensor data. Type A sensors send their data in simple string format. Fig. 6 shows the three implementation configurations we used for evaluation purposes. The details of these configurations are as follows:

1) Configuration A: We used Type A sensors (TelosB). The semantic virtual sensors sent their raw data to a GTO node. The GTO node (acting as an AA) downloaded the required ontology from an OA and annotated the raw sensor data. Lastly, the annotated data was sent to the fire monitoring semantic application via SA.

2) Configuration B: We used Type B sensors (Java SunSpots). The ontology used to annotate the data was stored locally in the Type B sensors, hence there is no ontology overlay. We implemented the AA in the Type B sensors using μJena library [15]. This way they did not need any GTO node to perform annotation on their behalf. Each semantic virtual sensor generated the raw data, annotated it and sent it to the fire monitoring semantic application via SA.

3) Configuration C: We used both Type A and Type B sensors. All of the sensors sent their raw data over the Internet. For Type A sensors, we used a Contiki border router to allow them to directly communicate with the semantic application. For Type B sensors, we used Java Socket-Proxy which communicated with the semantic application on their behalf. In this configuration, the fire monitoring semantic application performed the annotation itself. This allowed us to measure the extra delay introduced by our approach.

C. Performance Metrics

The prototype's performance was assessed in terms of the following metrics: End-to-End Delay (E2ED), Ontology Download Time (ODT), Impact of the scalability of AAs, Expected Operation Time (EOT) of Java SunSpots, and the Impact of tasks on current draw from Java SunSpots battery.

E2ED is the time difference between when the semantic virtual sensors sent their raw data and when the corresponding success code (200 OK) is received from the fire monitoring semantic application. It includes the time taken by all intermediate steps (i.e. receiving raw data at AA, ontology discovery and download (for configuration A), and annotation process). ODT is the time it takes an AA to request and to receive the required ontology from an OA. Impact of scalability of AAs was studied in terms of discovery of an OA and ODT. To find EOT of Java SunSpots, we executed both semantic and non-semantic tasks continuously until the Spots died. For this purpose no sleep or power saving mechanism was used. Finally we determined the current draw from Java SunSpot battery while in shallow-sleep mode (no task, radio ON), executing semantic, and non-semantic tasks. The experiments were repeated 50 times and their confidence interval is 95%.

D. Results

Fig. 7 shows the individual E2ED of the three configurations. Configuration A has an average E2ED of *3566ms*. The actual annotation delay was negligible (*less than 10ms*), since

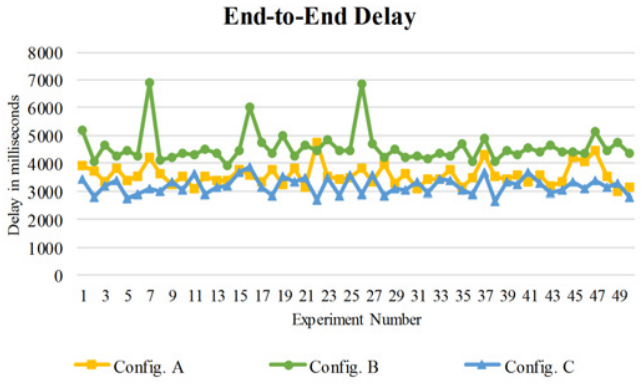


Fig. 7. End-to-End Delay

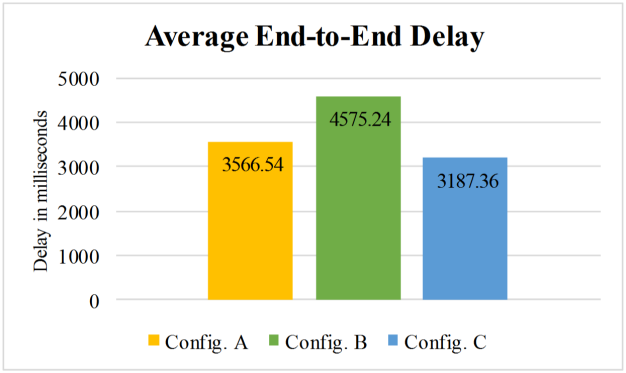


Fig. 8. Average End-to-End Delay

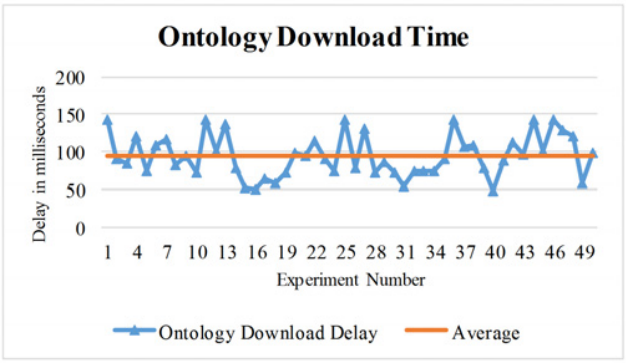


Fig. 9. Ontology Download Time

the AA was implemented on a laptop computer. The E2ED of configuration B is the highest, at *4575ms*. The average annotation delay was *525ms*, since the Java SunSpots were annotating data themselves. We found that this longer time was attributable to the low RAM size, only 1MB. Despite this, SunSpots showed promise and were able to annotate sensor data and run other tasks concurrently without any other issues. The E2ED of configuration C is *3187ms*. As expected, the semantic application was able to annotate the sensor data quickly but at the expense of developing the base ontology and then implementing it in addition to the application logic. Fig. 8 shows their average E2ED of all configurations after 50 repetitions. The average ODT for configuration A is *94ms* as shown in Fig. 9, which is typical in LAN environment using JXTA protocol.

Since JXTA was used for implementation, it had direct impact on the scalability part. The results in Fig. 10 show the increase in OA discovery time when AAs increase. JXTA is known to perform poorly when peers in the network increase and this was demonstrated in this work. However, the increase in AAs did not impact the ODT mainly because OA was already discovered. Here the average ODT was around *100ms*, almost similar to the one shown in Fig. 9.

Fig. 11 shows the EOT of the Java SunSpots while running a semantic and a non-semantic task, without using any sleep mechanism. SunSpots lasted *571* and *603* minutes for semantic and non-semantic tasks respectively in a lab environment. If we consider extreme battery discharge (about 20%) then the operation time reduces to *456* and *482* minutes respectively. We also found that SunSpots draw *38mA* current *(base value)* during the shallow mode (no task, radio ON), *75.6mA* for non-semantic task *(98% increase from base value)* and *79.8mA* for semantic task *(109% increase from base value)*.

For all three configurations, we also experienced delay due to circumstances beyond our control, e.g. from time to time GAE would start a new process for the fire monitoring semantic application and reload it thereby incurring unnecessary delay. We were able to determine this from the log files of our fire monitoring semantic application.

We believe that for future semantic applications, it will be important to use multiple WSN infrastructures that may not be geographically co-located. In such cases, it will be difficult to know beforehand, the capabilities of a WSN, the types of sensors and their observations. Also for WSN infrastructure owners may only want to share the sensor data instead of exposing their infrastructure altogether. In such situations, it makes sense to have an annotation mechanism that provides annotated data to multiple semantic applications.

VI. Related Work

A framework called semantic sensor web [7] annotates sensor data and provides situational awareness. The annotation is done using spatial, temporal and thematic metadata. In [16] the Sensor Observation Service SOS from SWE is extended by incorporating support for a semantic knowledge base. They use spatial, temporal and thematic ontologies to annotate sensor data. Both [7] and [16] rely on SWE, hence they are not suitable for resources-constrained environments. A two-layer architecture to annotate and query the sensor data is presented in [17]. The sensor data is collected in a pattern dictionary, in the back-end layer, to generate patterns along with semantic annotations. The patterns are used to determine the type of a new sensor and to automatically annotate its data. A crawler is used to retrieve the sensor data from multiple WSNs and store it after annotation. The front-end layer provides a GUI that the end-user utilizes to send search requests. The work is more focused on building automation domain.

In [18], the authors use their own SenMESO ontology for annotation which is a combination of various domain ontologies covering the sensor data and features of interest.

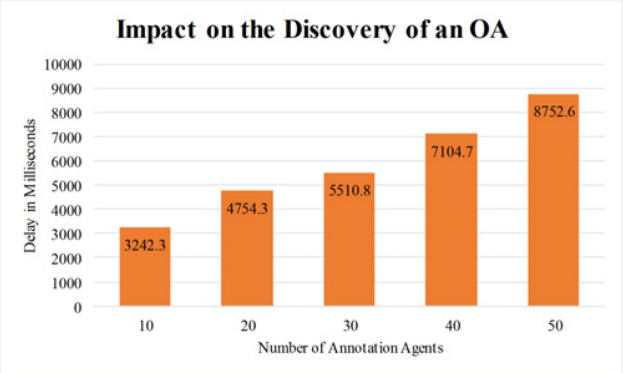


Fig. 10. OA Discovery Time When AAs Increase

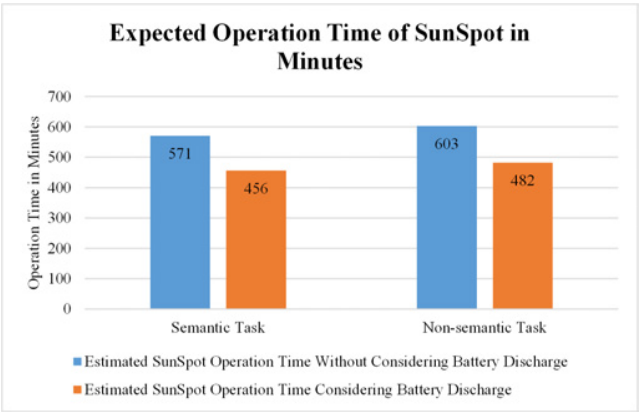


Fig. 11. Expected Operation Time of Java SunSpots (always on)

The sensors send the observed data in SenML format to the gateways. The gateway nodes generate an XML file and send it to the aggregation gateways which use the stored ontologies to annotate the sensor data and thereby allow different applications to use it. As an extension of this work, the authors present a mechanism to annotate M2M data in [19]. The work focuses on developing semantic-based M2M applications. The authors designed an M3 ontology to integrate cross-domain M2M data. There are no details regarding network architecture, but a web-based prototype is available. Two cross-domain semantic-based applications are also discussed.

Overall, the existing studies have several limitations, such as domain-specific solutions, and their use of protocols such as Sensor Web Enablement (SWE) [9] that are difficult to setup and definitely not suitable for resource-constrained environments. Another limitation is that they are focused on interoperability between sensors rather than their data.

VII. Lessons Learned and Future Work

We have learned several lessons. The first lesson is that WSN node-level virtualization is still a potent research area with very few solutions readily available. More efforts are required from designing a capable WSN operating system like [20] to unconventional energy harvesting mechanisms for sensor nodes like [21]. The second lesson is that current overlay middleware solutions are not suitable for WSNs because none has been designed to work with these resource-constraint devices. JXTA is too heavy for sensor nodes and its future is also uncertain. The third lesson is that there are not many libraries for semantic annotation that can be used by resource-constrained devices. We found an old J2ME-based μJena library and after several modifications managed to use it with Java SunSpots. However it only annotates data in N-TRIPLE format, whereas standard Apache Jena Framework supports multiple formats. Extensions to μJena library to annotate sensor data similar to Apache Jena Framework can be a useful contribution.

We have identified several key research issues that need to be addressed. First is the optimal selection of sensor nodes for the roles of AAs and OAs using energy-aware algorithms. These algorithms also need to take into account the characteristics of WSNs. Second issue is regarding the management of base ontology, since new types of sensors with new sensing capabilities may be deployed along with the existing WSN infrastructure. There is a need to have an easy to use mechanism to create and manage the ontology and later distribute it in the WSN infrastructure in an efficient manner. Third issue is that there is a need for lightweight P2P middleware for capable sensor nodes. This would make it possible for geographically-distributed sensors to share their data efficiently.

The final but very important issue is the possible integration of our proposed architecture with Platform-as-a-Service (PaaS) for the rapid provisioning of WSN application that can be offered as SaaS. In our current implementation we (partly) bypass Google Infrastructure for the interactions with our virtualized WSN infrastructure. As future work, we plan to integrate WSN infrastructure with a PaaS and allow its management at a higher level of abstraction through dynamic resource provisioning.

VIII. Conclusion

Semantic applications are being used in many application areas such as life sciences, media, and information systems. Annotating sensor data allows the end-users to get high-level information about the real-world situations instead of raw measurements of individual sensors. This could potentially open doors to many new applications. In this paper we have proposed an architecture for annotating sensor data in virtualized WSNs where sensors run multiple application tasks concurrently. Our architecture is applicable to both resource-constrained and resource-full sensors. We have also demonstrated the feasibility of the proposed architecture by realizing a representative use case using heterogeneous sensors. Several research issues have also been identified as future work.

Acknowledgment

This work is partially supported by CISCO systems through grant CG-576719, and by the Canadian Natural Science and Engineering Research Council (NSERC) through the Discovery Grant program.

References

[1] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292-2330, Aug. 2008.

[2] I. Khan, F. Belqasmi, R. Glitho, and N. Crespi, "A multi-layer architecture for wireless sensor network virtualization," in *Wireless and Mobile Networking Conference (WMNC), 2013 6th Joint IFIP*, Dubai, UAE, pp. 14.

[3] A. Merentitis, et al., "WSN Trends: Sensor Infrastructure Virtualization as a Driver Towards the Evolution of the Internet of Things," presented at the *UBICOMM 2013, The Seventh International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, Porto, Portugal, 2013, pp. 113-118.

[4] S. Abdelwahab, B. Hamdaoui, M. Guizani, and A. Rayes, "Enabling Smart Cloud Services Through Remote Sensing: An Internet of Everything Enabler," *IEEE Internet of Things Journal*, vol. 1, no. 3, pp. 276-288, Jun. 2014.

[5] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless Sensor Network Virtualization: Early Architecture and Research Perspectives," *IEEE Network Magazine*. (accepted for publication), in-press.

[6] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff, "Semantic annotation, indexing, and retrieval," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 2, no. 1, pp. 49-79, Dec. 2004.

[7] A. Sheth, C. Henson, and S. S. Sahoo, "Semantic Sensor Web," *IEEE Internet Computing*, vol. 12, no. 4, pp. 78-83, Jul. 2008.

[8] M. Compton, et al., "The SSN ontology of the W3C semantic sensor network incubator group," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 17, pp. 25-32, Dec. 2012.

[9] M. Botts, G. Percivall, C. Reed, and J. Davidson, "OGC®Sensor Web Enablement: Overview and High Level Architecture," in GeoSensor Networks, S. Nittel, A. Labrinidis, and A. Stefanidis, Eds. Springer Berlin Heidelberg, 2008, pp. 175-190.

[10] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys Tutorials*, vol. 7, no. 2, pp. 72-93, Second 2005.

[11] M. Liu, T. Leppanen, E. Harjula, Z. Ou, A. Ramalingam, M. Ylianttila, and T. Ojala, "Distributed resource directory architecture in Machine-to-Machine communications," in *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Lyon, France, 2013, pp. 319324.

[12] J. Menp, J. J. Bolonio, and S. Loreto, "Using RELOAD and CoAP for wide area sensor and actuator networking," *J Wireless Com Network*, vol. 2012, no. 1, pp. 122, Dec. 2012.

[13] L. Gong, "JXTA: a network programming environment," *IEEE Internet Computing*, vol. 5, no. 3, pp. 88-95, May 2001.

[14] C. Jennings, J. Arkko, and Z. Shelby, "Media Types for Sensor Markup Language (SENML)." *work-in-progress* [Online]. Available: https://tools.ietf.org/html/draft-jennings-senml-10. [Accessed: 29-Sep-2014].

[15] F. Crivellaro, "μJena: Gestione di ontologie sui dispositivi mobile," Thesis, M.Sc., *Politecnico di Milano*, Milan, Italy, 2007.

[16] C. . Henson, J. K. Pschorr, A. Sheth, and K. Thirunarayan, "SemSOS: Semantic sensor Observation Service," in *International Symposium on Collaborative Technologies and Systems, 2009. CTS 09*, Baltimore, USA, 2009, pp. 44-53.

[17] D. Pfisterer, et al., "SPITFIRE: toward a semantic web of things," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 40-48, Nov. 2011.

[18] A. Gyrard, C. Bonnet, and K. Boudaoud, "A machine-to-machine architecture to merge semantic sensor measurements," in *WWW 2013, 22nd International World Wide Web Conference, Doctoral Consortium*, Rio de Janeiro, BRAZIL, 2013.

[19] A. Gyrard, C. Bonnet, and K. Boudaoud, "Enrich machine-to-machine data with semantic web technologies for cross-domain applications," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, Seoul, Korea, 2014, pp. 559-564.

[20] O. Hahm, E. Baccelli, H. Petersen, M. Whlisch, and T. C. Schmidt, "Demonstration Abstract: Simply RIOT: Teaching and Experimental Research in the Internet of Things," in *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, Piscataway, NJ, USA, 2014, pp. 329-330.

[21] E. Gelenbe, D. Gesbert, D. Gunduz, H. Kulah, and E. Uysal-Biyikoglu, "Energy harvesting communication networks: Optimization and demonstration (the E-CROPS project)," in *2013 24th Tyrrhenian International Workshop on Digital Communications - Green ICT (TIWDC)*, Sept. 23 - 25, 2013 Genoa, Italy, pp. 16.

Annex G

Paper VII

This paper is work-in-progress

Towards Provisioning of Semantic Applications over Virtualized Wireless Sensor Network IaaS

Imran Khan[×*+], Rifat Jafrin[+], Jagruti Saho[+] Roch Glitho[+], Noel Crespi[*]
[*]Institut Mines-Télécom, Télécom SudParis, Evry, France
[+]Concordia University, Montreal, Canada
[×]imran@ieee.org

Abstract

Sharing a deployed Wireless Sensor Network (WSN) infrastructure among multiple, concurrent applications can help realize the true potential of Internet-of-Things (IoT) and ubiquitous computing. Virtualized WSNs can help to achieve such sharing where multiple applications and services use a deployed WSN infrastructure at the same time. These applications and services may include semantic applications which are very much pertinent to provide situational awareness to the end-users who are then able to understand the context of the events and make informed decisions. However, provisioning of semantic applications in virtualized WSNs is not trivial. Another challenge, typically faced by WSN infrastructure owners, is how to create and manage domain independent ontology, used for sensor data annotation, which corresponds to their deployed infrastructure. In this paper we extend our previous work to *i)* allow WSN infrastructure owners to create and manage the ontology, and *ii)* to facilitate the efficient sensor data annotation in a distributed manner. For the former we present an ontology development and management application. For the later we present a heuristic-based genetic algorithm to select capable nodes for storing the base ontology in a deployed WSN. The ontology management extension is implemented and evaluated using a proof-of-concept prototype using Java SunSpot kit while the simulation results of the algorithm are presented.

Keywords

Wireless Sensor Networks; Semantic Web; Domain Ontologies; Data Annotation; Overlay networks

1. Introduction

Recently the concept of Wireless Sensor Networks (WSNs) virtualization [1], [2], [3], [4] has gained attention that uses the concept of multiple concurrent application tasks running on a sensor node. With this concept it has become possible to offer a deployed WSN infrastructure for multiple applications and services. This is in contrast to the traditional task-oriented, domain-specific deployments of WSNs [5] where applications came usually bundled with them and it is prohibitively expensive and time-consuming to deploy new applications over these deployments. WSNs are considered as one of the corner stones of the Internet-of-Things (IoT) paradigm [6] hence it is pertinent to offer them as Infrastructure-as-a-Service (IaaS) using the same concepts as in Cloud Computing. The amalgamation of WSNs and Cloud Computing results in virtualized WSN (vWSN) IaaS [7] that can decouple infrastructure from the applications using it. This way newer applications and services could be provisioned as and when required.

Traditional WSN applications are built around the concept of receiving sensor data in raw format without any ability to understand its context and meaning. Additionally, this raw sensor data fails to provide high-level details to gain situational awareness because an end-user cannot make queries to better understand a situation. For example, a traditional fire monitoring application can get only a notification about the fire eruption event but will not allow its user to query for the details like *where is the source of fire?* Semantic applications, on the other hand, easily allow end-users to make such queries to get results like '*in a public library*'. This allows for provisioning more rich and interactive applications to the end-users. Another benefit of incorporating semantic concept to the WSNs is that we can have standard way to share sensor data across different application domains. This can be particularly useful to achieve interoperability among various vertical solutions which are typically found these days. With the increase in number of

This paper is work-in-progress

vWSN IaaS deployments by the third-party actors (public, community and research deployments) future WSN deployments will have to support both traditional as well as semantic applications.

With this background, it is clear that we need efficient solutions to annotate sensor data using ontology concepts. Recently we proposed an in-network, distributed sensor data annotation architecture [8] to provision traditional as well as semantic applications over a vWSN IaaS. Capable nodes in the architecture store the ontology concepts, which are later used for annotation purposes. This in-network annotation approach has many benefits than existing centralized solutions that first store sensor data and later annotate it. For example, each capable sensor is able to annotate its data directly in real-time.

However, for a general solution a standard ontology from the sensor domain is required. It will be helpful to have the ontology which is independent of any application domain. The reason being that it is almost impossible to know all future application that will possibly use a WSN infrastructure. This standard ontology should be developed, managed and distributed over the network in an autonomous distributed manner by the WSN infrastructure owner. This mandates an efficient ontology development and management solution. The solution should ease the development of the standard ontology, update it (due to deployment of new sensors or removal of old ones), and send it to the capable nodes to allow for distributed sensor data annotation.

This paper extends our previously proposed architecture by making the following contributions: An ontology development and management application is presented that allows a WSN infrastructure owner to easily create and manage the standard ontology. This web-based GUI application allows intuitive ontology creation and management even for a novice user. Second contribution consist of several enhancements to the original architecture to disseminate and store the developed ontology in the WSN infrastructure in a distributed manner. The concept of overlays is used for ontology dissemination after it is developed, and also for the sensor data annotation. Finally we propose a heuristic-based Genetic Algorithm (GA) to select capable nodes for storing the developed ontology. We use multi-objective criteria to select best possible candidates (including capable sensors) for ontology storage. The GA tries to minimize the energy and memory consumption by selecting candidate nodes in a near-optimal way.

The rest of the paper is organized as follow: In Section 2 a motivating scenario is presented along with a set of requirements. In Section 3 we discuss the related work and evaluate them using the set of requirements. In Section 4 we discuss our proposed architecture in detail. Section 5 presents our heuristic algorithm. Details regarding prototype implementation and results are discussed in Section 6. Section 7 identifies the future work and concludes the paper.

2. Motivating Scenario & Requirements

In this section, we first present a general motivating scenario and then derive a set of requirements from it. More specific motivational examples can be found in [1], [2] and [8].

A. Motivating Scenario

Let us assume that a WSN Infrastructure owner deploys its heterogeneous sensors having different capabilities on a large geographic area to detect different physical phenomena. In the context of IoT such WSN Infrastructure owner can be a city administration interested to provide smart city services to its citizens or a large scale R&D research project such as SmartSantander [r9]. In these situations, the infrastructure owner is interested to offer the deployed WSN as IaaS to users to provision multiple applications and services over it. Some of these could be semantic-based allowing their users to infer additional knowledge about the detected physical phenomenon. Hence efficient sensor data annotation mechanism is required in which it should be possible for WSN infrastructure owner to develop and manage the ontology that will be used for the annotation purposes.

B. Requirements

Based on the scenario above, we derive the following five requirements.

First, the proposed solution should be domain/application independent meaning that sensor data annotation should not be domain/application specific. This can be achieved by using or extending standardized ontologies used for WSNs.

The *second* requirement is that the proposed solution should be able to deal with the infrastructure heterogeneity to ensure interoperability. Any large scale WSN infrastructure will contain different sensors nodes with having different sensing capabilities, data formats and other properties.

The *third* requirement is that it should easy for the WSN infrastructure owner to create, extend and manage the standard ontology without knowing technical/protocol details.

The *fourth* requirement is the distributed storage of the ontology in the WSN. This guarantees fault-tolerance and allows remote sensors to find required ontology nearby instead of communicating with a central node that may be multiple hops away.

The *fifth* requirement is that the real time annotation of sensor data should be supported. This can be particularly useful for emergency applications that rely on sensors to get detect real-world events.

The *sixth and the final* requirement is that the sensor data annotation should be performed in a distributed manner to ensure that node failures do not affect the annotation process.

3. Related Work

In this section we present existing solutions, similar to our research area, and evaluate them critically.

One of the earliest efforts to annotate sensor data with semantic metadata, to provide situational knowledge, is proposed in Semantic Sensor Web (SSW) framework [r10]. This work is based on Sensor Web Enablement (SWE) from OGC Semantic Web effort by W3C. SWE annotates the sensor data using temporal, spatial and thematic concepts. OGC SWE languages are used for temporal and spatial annotation of sensor data. However thematic annotations are applied using sensor data analysis or tags. The authors use the RDFa for the semantic annotation of the sensor data and domain ontologies for providing concepts and relationships. Semantic Web Rule Language (SWRL) is used to reason over the annotated data and to infer knowledge. Authors develop two proof-of-concept prototype applications using their proposed architecture.

In [r11] a three layer architecture based on OGC SWE and semantic web is presented that facilitates the gathering, processing and exploiting sensor data in real-time. They use Observations and Measurements (O&M) and SensorML specifications for semantic specification of the sensors, their properties and their raw data. The first layer (data) involves in collecting raw data from heterogeneous senor. Then the second layer (processing) aggregate those raw measurement, transform them into XML format and forward it to the next layer. The third layer (semantic) process those aggregated data by mapping them into ontology model contained in a database .The annotation are created at third layer and stored in a knowledge base. An external reasoning tool is used to respond to the queries that the end user submits.

The work in [r12] discusses the outcomes of a large-scale European project SPITFIRE allowing transition from semantic sensor web to semantic web-of-things. The work provides three main contributions: *i)* new sensor description mechanism that easily integrates with Linked Open Data cloud (LOD). The data from the LOD can be used by different applications/services. *ii)* Semi-automatic creation of semantic sensor descriptions. The sensor data is collected in a pattern dictionary to generate patterns along with semantic annotations. The patterns help to determine the type of a new sensor and automated annotation of its data. *iii)* Efficient search mechanism to find sensors and things based on their current state. A crawler is used to retrieve the sensor data/metadata from multiple sources (sensors and web pages). The gathered data is stored in an RDF triple store and later queried using SPARQL query engine. A reference implementation architecture is presented as a proof-of-concept but performance measurements for validation are not presented.

This paper is work-in-progress

The work done in [r13] propose a new ontology SenMESO for sensor data annotation which is a combination of various domain ontologies covering sensor data and features of interest. The gateway nodes receive sensor data in different formats and convert it into XML format in order to support interoperability. The aggregation gateways incorporate semantics to XML sensor data using RDF, RDFS, OWL and domain ontologies. In this work, the sensor measurements, after annotation, are linked with the LOD (Link Open Data Cloud) where additional information can be inferred using additional concepts. An online prototype is available to show viable of the proposed architecture.

The work [r14] focuses on developing semantic-based M2M applications by combining, enriching cross-domain ontologies that exist in isolation but use similar concepts. The idea is to link and reuse the existing ontologies that have been developed by the domain experts without using semantic web guidelines. The authors designed an M3 ontology to integrate cross-domain M2M data providing a unified way to describe events, measurements and sensors. M3 ontology is an extension of SSN ontology, contains detailed description of more than 30 sensors and various domains. New OWL properties are used to inter-link the same concepts present in different domain ontologies. Additionally Linked Open Rules (LOR) concept is introduced to share and reuse semantic rules from different domains. Domain rules are specified using semantic web rule language. Two cross-domain semantic-based applications are also discussed but without any results.

In [r15] provide an approach to convert the sensor data in SenML format to RDF. A SenML reading from a sensor is first transformed to RDF elements, then an array of RDF triples is generated and finally these RDF triples are serialized to different formats. A prototype implementation is presented which is an application for monitoring water quality of fish farms. The implementation results show the performance gains while using SenML as compared to other data formats such as RDF, N-triple, and N3. Only the data from IoT devices (sensors) that send data in SenML can be transformed to RDF, which is the main limitation of this work.

Table I shows the evaluation of the existing works against identified requirements.

Table I: Comparison of related work

Related-Work	Use of standard ontologies	Applicable to heterogeneous sensors	Ontology management mechanism	Distributed storage of the ontology	Real-time annotation of sensor data	Annotation mechanism
Ref [r10]	Yes	Yes	No	No	No	Centralized
Ref [r11]	Yes	Yes	No	No	Yes	Centralized
Ref [r12]	Yes	Yes	Semi-automatic	No	No	Centralized
Ref [r13]	Yes	Yes	No	Yes	Yes	Decentralized
Ref [r14]	Yes	Yes	No	Yes	Yes	Decentralized
Ref [r15]	No	Only those supporting SenML	No	No	Yes	Centralized
This work	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Decentralized**

This paper is work-in-progress

4. Proposed Architecture

We begin by discussing the architecture we have proposed in our previous work concerning in-network, distributed sensor data annotation in vWSNs since we use it as the basis for this paper. Later we present architectural principles used to propose extended architecture and then the layers and functional entities.

A. Our Starting Point

The work in this paper is based on our previous WSN virtualization architecture [r8] which is illustrated in Fig. 1. The architecture consists of four layers. The physical layer consists of sensor nodes that support node-level virtualization. Both resource-constrained (e.g. TelosB, called Type A) as well as capable (e.g. Java SunSpots, called Type B) sensor nodes are considered. Capable sensors as well as high-end machines (e.g. base stations and sink nodes) act as Gates-to-Overlays (GTO) nodes to facilitate resource-constrained sensors to support node-level virtualization. The second layer is Virtual Sensor layer that abstracts as virtual sensors, the simultaneous tasks run by the physical sensors. There can be two types of virtual sensors: those who run semantic application tasks (and require data annotation), called semantic virtual sensors and those who run non-semantic (traditional) application tasks, called virtual sensors.

The Virtual Sensor Access layer has three functional entities (Annotation Agents (AAs), Ontology Agents (OAs) and Sensor Agents (SAs)) and two overlays (Annotation and Ontology overlays). The Annotation overlay consists of AAs, which annotate sensor data using the standard ontology. Each semantic virtual sensor is represented by a corresponding AA in the Annotation overlay. Also in the same overlay are the SAs, which receive annotated as well as non-annotated data from the virtual sensors and forward it to the end applications. The Ontology overlay consists of OAs that store the standard ontology. These OAs act as super-peers and provide the ontology to the requesting AAs. Final layer is Application Overlay layer which consists of multiple applications (semantic and non-semantic) over the deployed vWSN IaaS.

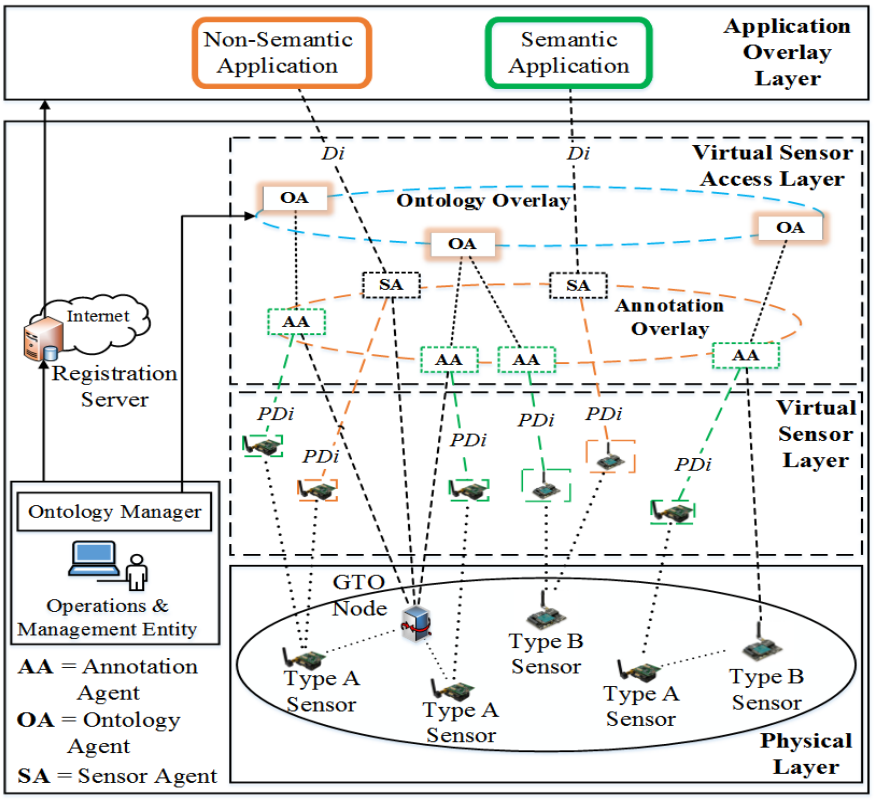


Figure 1: Sensor data annotation architecture for vWSNs

This paper is work-in-progress

The architecture is based on the following assumptions: first it is assumed that the sensors have already been discovered and are stored in a registration server. Applications and services that wish to utilize the sensors send queries to the registration server. There are several existing works such as [r16], [r17] to accomplish this. Second assumption is that the architecture does not store the sensor data (either raw or annotated). While it is perfectly possible to store sensor data and use it for data analytics and visualization when required, but currently this feature is not provided.

The key challenge that we addressed in our previous architecture, was to provide in-network sensor data annotation in a distributed manner in real-time unlike existing centralized solutions that first store the sensor data and later annotate it. Another contribution was to make our proposed solution domain/application independent since it is difficult to determine the type applications using WSN IaaS. This was achieved by using the concept of base ontology (related to the deployed infrastructure) for sensor data annotation.

B. Architectural Principles

The *first* architectural principle is that the WSN infrastructure owner should have an easy-to-use mechanism to develop and maintain base ontology.

The *second* architectural principle is that the base ontology, used to annotate the sensor data, will be stored in the network using overlays. Overlays have advantages like, resource sharing and lack of central control for more distributed solutions [r18]. They also make is easy to publish, search and receive the content in the overlay.

The *third* architectural principle is that the base ontology will be stored only in few selected capable nodes at a time to in order to not to over burden the nodes. This will also keep the control traffic (concerning base ontology to minimum). Whenever, network dynamics change or node failure occurs, the algorithm is executed again to select a new set of capable nodes.

The *fourth* architectural principle is that in order to keep the execution of the node selection algorithm to minimum, the base ontology should be replicated in the WSN infrastructure. This means that there will be multiple nodes storing the same copy of the base ontology, thereby increasing robustness and fault-tolerance.

C. Proposed Architecture

Fig. 2 shows the proposed architecture. It is based on our previous WSN virtualization architecture presented in Section 4-A. At physical layer there is a new node called WSN IaaS Manager that has a global view of the deployed WSN infrastructure. In this architecture, WSN IaaS Manager is responsible to first select capable nodes for storing the base ontology and then disseminate the ontology files over the *ODi* interface to the selected nodes. For this purpose a multi-objective genetic algorithm is used. The virtual sensor access layer remains the same as in the previous architecture.

There is a new functional entity in the ontology overlay in the virtual sensor access layer called Ontology Manager (OM). The role of OM is to hold the base ontology and provide it to the Ontology Agents (OA) when requested. The OM can be a centralized entity but in the proposed architecture, it is distributed over many capable nodes in the network (i.e. GTO nodes and Type B sensors). The rest of the functional entities and the overlays in virtual sensor access layer remain same as in previous architecture, i.e. the Annotation Agents (AA) request for the desired ontology from the Ontology Agents (OA) and use it for annotating the raw sensor data. Complete base ontology is replicated and stored in multiple OMs, while OAs store only portions of the base ontology. Finally, Sensor Agents (SA) in the Annotation overlay send the annotated data to the semantic applications.

It is important to mention that a single node can play the role of multiple functional entities. For example, a GTO node can act as OM, OA, SA and AA at the same time since it is much more resourceful than sensor nodes. On the other hand, a Type B Sensor may fulfil only some of these roles at a time, e.g. as AA and/or OM only since it is not resourceful as a GTO node.

This paper is work-in-progress

In order to develop and manage the base ontology, the WSN infrastructure owner uses the Ontology Development and Management Application. WSN infrastructure owner can hire a domain expert or even out-source the ontology development to a third-party. Once the base ontology is developed, it is provided to the WSN IaaS Manager who then makes decision on where to send the base ontology in the network. The only entity, that can receive the base ontology from the WSN IaaS Manager is the OM, hence it is important that OM have the most update version of the base ontology at all times.

The architecture is based on the architectural principles, mentioned in the previous section and fulfils the requirements mentioned in Section 3-B. According to *first* architectural principle, a web-based GUI application is developed that allows an interactive and easy way to create base ontology. As per *second* principle, a dedicated overlay (Ontology Overlay) is used for the storage of base ontology. Two functional entities, OM and OA are used to store complete and partial base ontology files respectively. As per *third* principle, the WSN Infrastructure Manager selects capable node using a multi-objective genetic algorithm whose details are presented in next section. Finally as per *fourth* principle, multiple capable nodes are selected to act as OM in the deployed infrastructure.

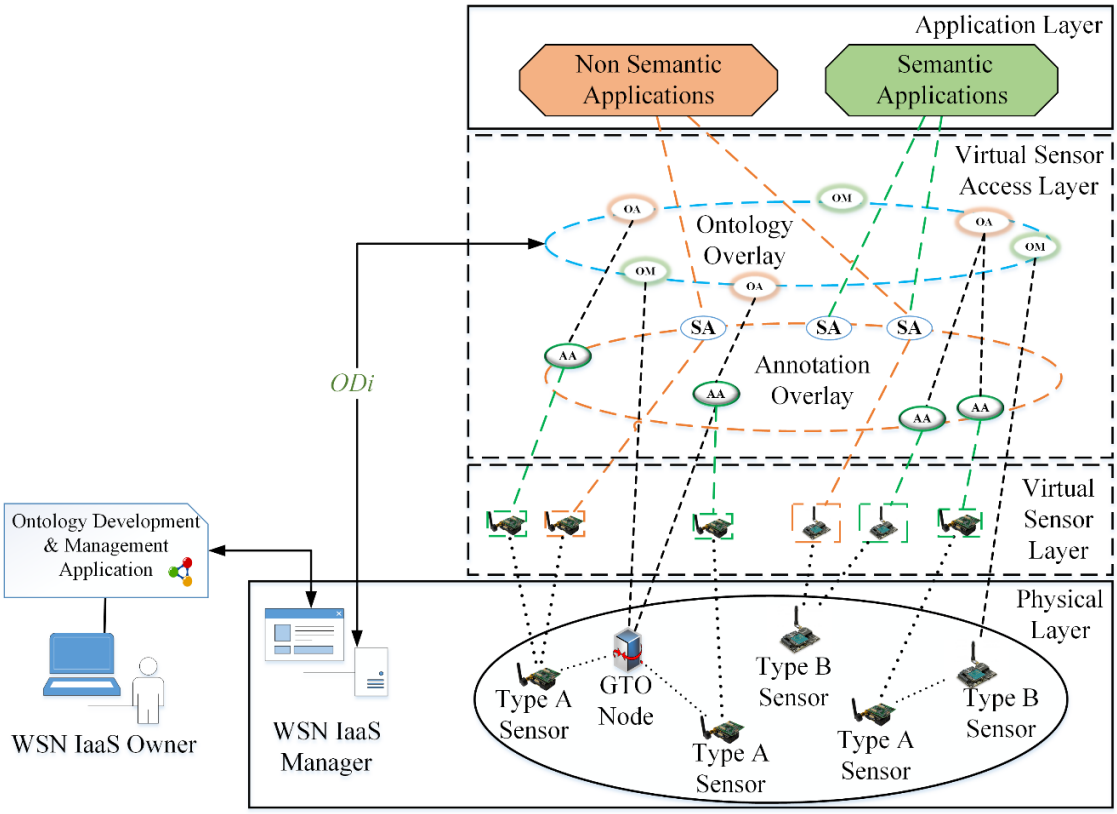


Figure 2: Proposed Architecture

D. Ontology Development and Management Application

In order to facilitate the easy creation and maintenance of base ontology as web-based GUI application is developed. Fig. 3 shows the screenshot of the application. It has the following functionality.

Fig. 4 illustrates the process of creating base ontology using the application. *First* step is that the ontology developer first adds the concepts related to sensor domain. For each concept sensor type, output type, output unit and observed property can be specified. This way different type of concepts can be included in the system, e.g., temperature, light, carbon, each linked to the sensors deployed in the network. Whenever a new type of sensor is deployed in the network, its information can be easily added in the ontology. For example, its name, attached sensor(s), sensor type, number of

attached sensors, its dimensions and range of values it supports. This *second* step is optional. The *third* step is that once the new concepts are included in the system, default ontology is used to incorporate these new concepts. For example, we use the standard SSN ontology (as default ontology) and extend it with new concepts. In this step default ontology is loaded from the local ontology database automatically. In the *fourth* step, the new concepts (mentioned as child concepts) are included with the existing concepts automatically. In the *fifth* step, the values of property and domain range are updated to reflect the new additions/modifications. Finally, the base ontology is created which can be used by the sensor to annotate their data. Fig. 5 shows the newly created base ontology.

The application also provides option to modify/update existing concepts. All steps, except first and second, are performed automatically, i.e. without any user input.

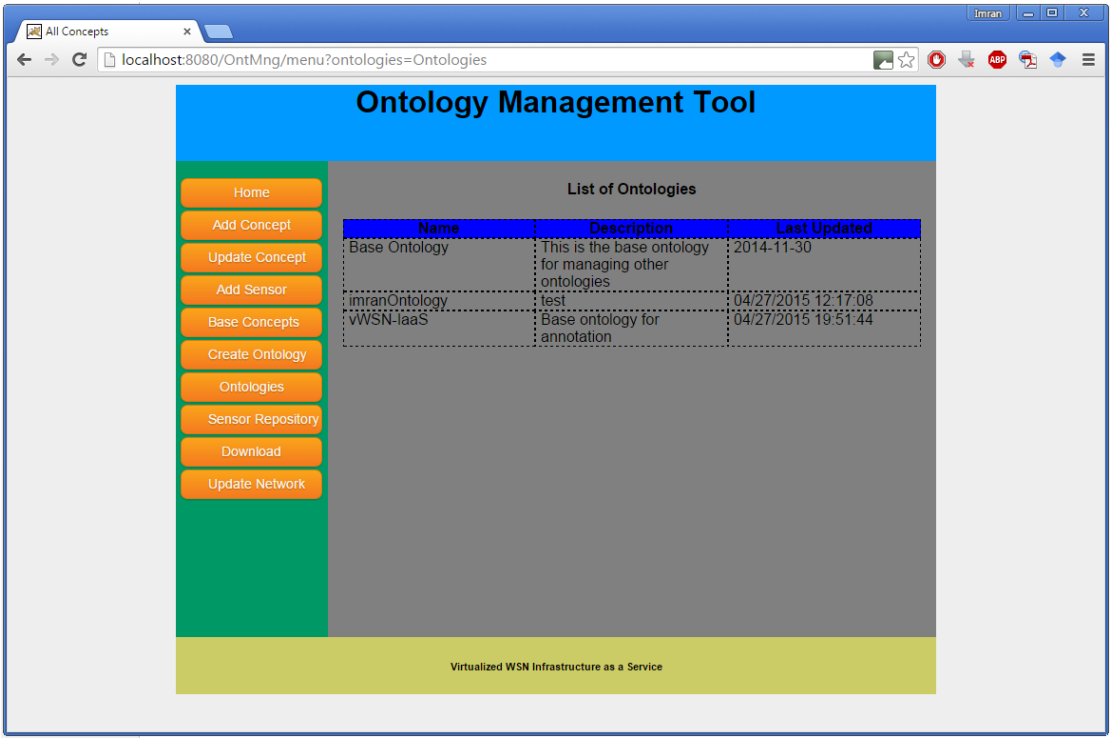


Figure 3: Ontology Development and Management Application

This paper is work-in-progress

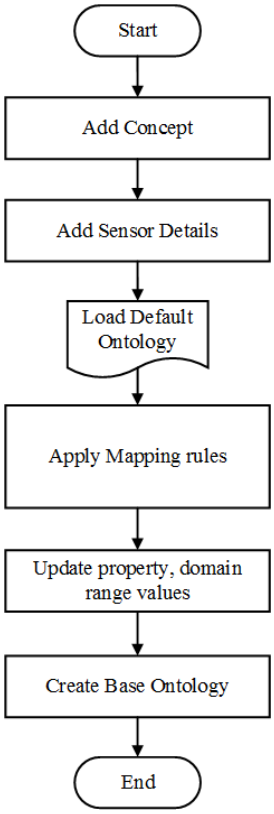


Figure 4: Base Ontology Creation Process

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://localhost/BaseOntology.owl#"
    xmlns:base="http://BaseOntology.owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:ssn="http://purl.oclc.org/NET/ssnx/ssn#">
    <owl:Ontology rdf:about="http://BaseOntology.owl"/>
    <owl:Class rdf:about="http://purl.oclc.org/NET/ssnx/ssn#Sensor"/>
    <owl:Class rdf:about="http://BaseOntology.owl#Longitude">
        <rdfs:subClassOf>
            <owl:Class rdf:about="http://BaseOntology.owl#SensorLocation"/>
        </rdfs:subClassOf>
    </owl:Class>
    <owl:Class rdf:about="http://purl.oclc.org/NET/ssnx/ssn#SensorOutput"/>
    <owl:Class rdf:about="http://BaseOntology.owl#ObservedProperty"/>
    <owl:Class rdf:about="http://BaseOntology.owl#MeasurementUnit"/>
    <owl:Class rdf:about="http://BaseOntology.owl#Latitude">
        <rdfs:subClassOf rdf:resource="http://BaseOntology.owl#SensorLocation"/>
    </owl:Class>
    <owl:ObjectProperty rdf:about="http://purl.oclc.org/NET/ssnx/ssn#observes"/>
    <owl:ObjectProperty rdf:about="http://BaseOntology.owl#hasUnit"/>
    <owl:ObjectProperty rdf:about="http://purl.oclc.org/NET/ssnx/ssn#observedBy"/>
    <owl:DatatypeProperty rdf:about="http://BaseOntology.owl#hasLongitude"/>
    <owl:DatatypeProperty rdf:about="http://BaseOntology.owl#hasLatitude"/>
    <owl:DatatypeProperty rdf:about="http://BaseOntology.owl#hasSensingTime"/>
    <owl:DatatypeProperty rdf:about="http://BaseOntology.owl#hasSensorName"/>
    <owl:DatatypeProperty rdf:about="http://purl.oclc.org/NET/ssnx/ssn#hasValue"/>
</rdf:RDF>
<!-- Created with Protege (with OWL Plugin 3.4.8, Build 629)  http://protege.stanford.edu -->
```

Figure 5: Sample Base Ontology Created by the Ontology Developer

This paper is work-in-progress

E. Procedures

The proposed architecture needs certain procedures to operate properly. The procedures are classified as *i)* management operations and *ii)* operational procedures.

The management procedures include the following. *1)* Selection of sensors and GTO nodes that will play the role of, *i)* OMs and OAs in the ontology overlay, and *ii)* AAs in the annotation overlay. *2)* Distribution of base ontology over OMs, *3)* distribution of base ontology over OAs, and *4)* Recovery from OA failures.

The operational procedures include the *1)* ontology discovery by OAs, and *2)* sensor data annotation. The first operational procedure is needed in case AA does not have the ontology while the final operational procedure is the actual annotation process.

1) Management Procedures

The management procedures pertinent to the selection of OAs and AAs are discussed in [r8] whereas the selection of OM is as follows. The WSN IaaS Manager has most recent information of the network i.e. the GTO and sensor nodes, and their current status regarding energy and storage space. Based on this information, WSN IaaS Manager selects, at random, capable nodes to act as OMs along with a set of OAs. The set of OAs is determined after executing the genetic algorithm whose details are presented in next section. Each OM receives complete base ontology, while the OAs receive parts of the base ontology. The reason is that, in the architecture only GTO nodes will act as OMs while Type B sensors will act as OAs. Therefore, storing parts of the base ontology in OAs makes sense since they are more resource-constrained nodes. Another benefit is that when a concept is modified or extended, only OAs storing that particular part will be updated instead of all OAs.

The ontology distribution over OMs is as follows: After the developed base ontology is received by WSN IaaS Manager, it randomly selects a set of GTO nodes to act as OMs and provides them with the complete base ontology. The base ontology is stored in multiple OMs to ensure redundancy.

The ontology distribution over OAs is as follows: The developed base ontology consists of multiple concepts such as temperature, humidity, light, carbon, acceleration, pressure and so on.

$$\text{Base ontology} = \text{concept}_{\_\text{temperature}} + \text{concept}_{\_\text{humidity}} + \ldots + \text{concept}_{\_k}$$

Since OAs are resource-constrained devices and may not need all concepts, OMs divide the base ontology into multiple parts in such as a way that each part contains one complete concept. Then each of these parts are randomly sent to the selected OAs. This strategy helps in situations where a WSN deployment contains heterogeneous sensors with different capabilities but currently few applications are using only some of these capabilities. When a different concept is required, OAs can easily request it from an OM in the ontology overlay.

The final management procedure is recovery from failures, i.e. when an OA fails due to any issue. When such failure occurs it is important to select a new candidate to act as OA and provide it with the same part of the base ontology. Since WSN IaaS Manager will be able to detect the OA (node) failure in the network, it will re-execute the same node selection algorithm to find a suitable replacement of the failed sensor to act as OA. After joining the ontology overlay, the new OA will be able to request for part of base ontology when required.

2) Operational Procedures

The first operational procedure is the ontology discovery and it can be proactive or reactive. In proactive approach, OMs send parts of base ontology to the chosen OAs who then send these received parts to AAs even if there is no semantic application using the vWSN infrastructure yet. In reactive approach, the virtual sensor executing task for semantic application send data to AAs for annotation. If AA does not have the required concept, it will send ontology request to OA. If the OA has the required concept, it will send it, otherwise the request will be sent to the OM.

This paper is work-in-progress

The second operational procedure is the data annotation, works as follows. The semantic virtual sensors send raw sensor data to the AAs. Once an AA receives the sensor data, it first checks locally if it has the required ontology to annotate it, if not a discovery request is sent to the ontology overlay. Finally AA annotates the sensor data, using the received ontology, and sends it to SA. SA is then responsible for sending the annotated data to the semantic application.

5. Implementation and Results

In this section we present the implementation details using a prototype. The implementation shows the usage of Ontology Development and Management Application and covers the ontology distribution using the architecture.

A. Implementation Choices

To develop Ontology Development and Management Application, the following software and technologies are used: MySQL Database to store the base ontology. MySQL Database is a popular and easy to use database solution and can be used to efficiently store the base ontology concepts along with details. Since the designed application is web-based, it is hosted using open-source and popular web server Apache Tomcat. Tomcat implements many Java EE specifications and provides a complete Java-based HTTP web server environment for Java code to run in. The Ontology Development and Management Application itself is developed in Java using JAX-WS web services API. In order to generate base ontology, using the stored concepts, as an RDF file we use the Protégé OWL API, open-source Java-based library for OWL and RDF(S). The API makes it easy to load, save and modify OWL data models.

For creating annotation and ontology overlay, we used JXTA protocol which is an open source P2P protocol specification to create independent, roust and efficient overlay networks. For this work we used java-based JXSE implementation of JXTA. WSN IaaS Manager, OM and OA implement JXTA Rendezvous Peer functionality to store the base ontology and its parts respectively. We used the JXTA Content Management System (CMS) to send the base ontology from WSN IaaS Manager to OM, then from OM to OA and finally from OA to AA. JXTA CMS also makes it easy to advertise and distribute the contents in the overlay. The functionality of AA is split into two parts: one is implemented in the laptop as JXTA Edge Peer functionality to request for and receive the part of base ontology. The second part is the actual annotation process implemented in Java SunSpot using the J2ME-based μJena library [r19].

B. Prototype Setup

Fig. 6 shows the prototype setup. We used three laptops connected to a private LAN. One laptop is used to host the Ontology Development and Management Application and to act as WSN IaaS Manager. The second laptop acts as OM while third laptop acted as OA and implements the partial functionality of AA, as mentioned before. The respective functionalities of these entities were implemented as Java applications in all three laptops. One Java SunSpot kits was used consisting of 1 base station node and 2 SunSpots with onboard sensors. Each SunSpot executed two application tasks at the same time. The annotation functionality of AA was implemented in the SunSpots as mentioned before.

This paper is work-in-progress

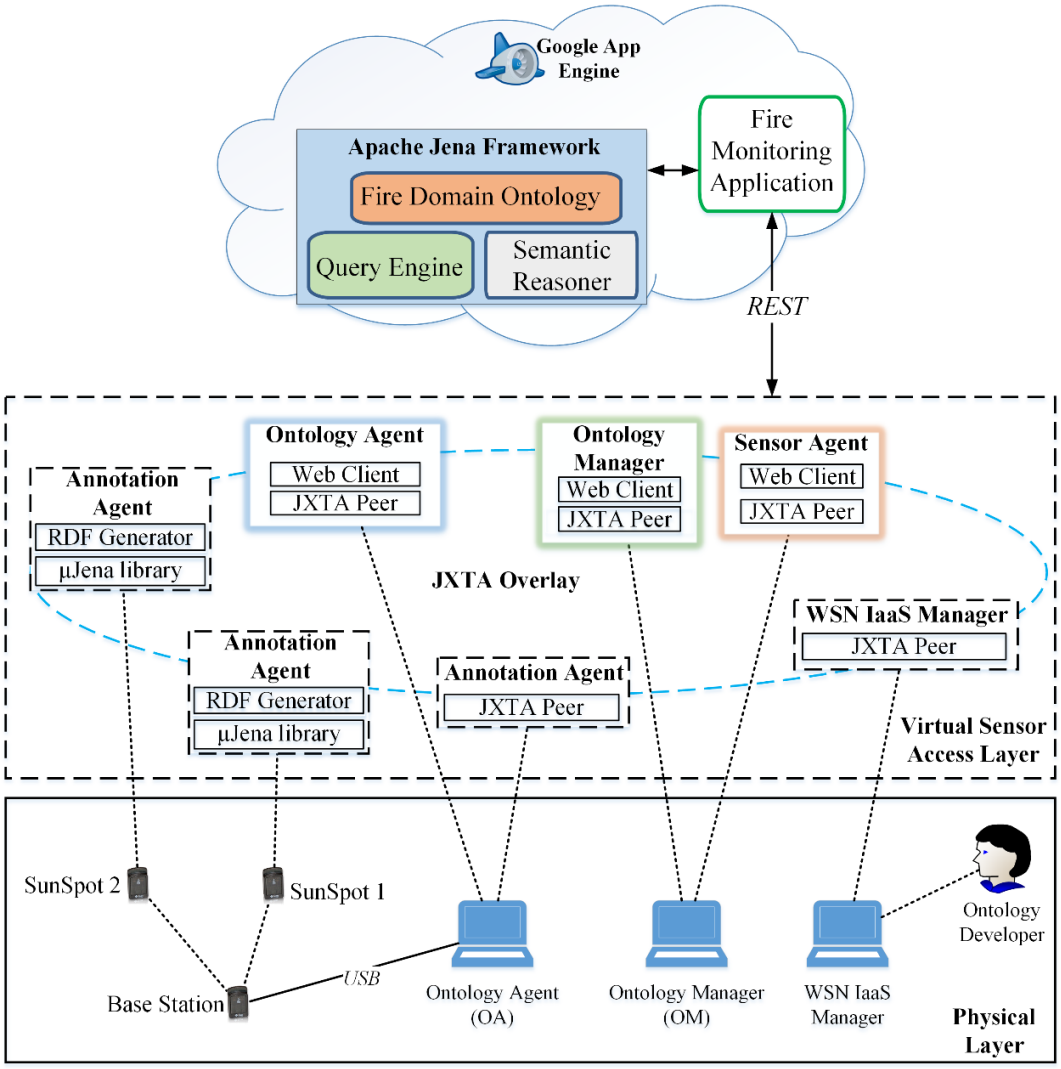


Figure 6: Prototype Setup

C. Performance Metrics

The performance of the prototype was assessed in terms of the following metrics: Overlay Creation Delay (OCD), Ontology Dissemination Time (ODisT) and Ontology Download Time (ODT). OCD is the time to create JXTA overlay from a non-existent state to a ready state, when it is ready to accept join requests. We measured this delay inside the Java code to ensure that the OCD does not include the JVM start-up delay. ODisT is the combination of the following delays: i) Delay from ontology application to WSN IaaS Manager, ii) Delay from WSN IaaS Manager to OM, and iii) Delay from OM to OAs. For ODT we measured the delay when an AA requests and receives the missing part of base ontology from OA.

All these experiments were repeated 50 times with 95% confidence interval.

D. Results

Fig. 7 shows the OCD of 50 experiments as well as its average value. The average OCD is found to be 1906ms from 50 experiments. It is important to remember that the OCD pretty much depends on the configurations of the machines that act as JXTA peers and is unavoidable. However, it is experienced only during the overlay initiation phase so does not necessarily make much impact during the sensor data annotation process.

This paper is work-in-progress

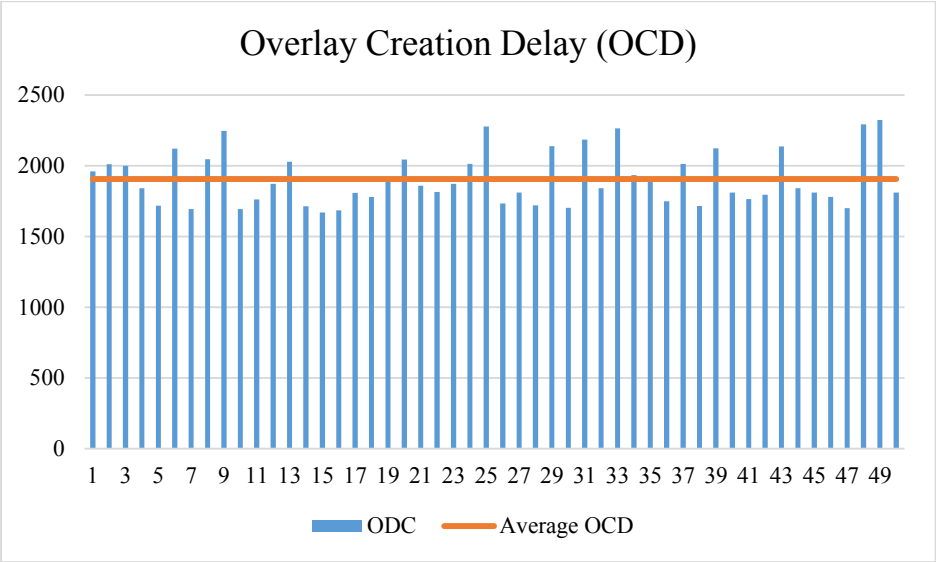


Figure 7: Overlay Creation Time

Figure 8 shows the ODisT which includes *i)* Delay from ontology application to WSN IaaS Manager, *ii)* Delay from WSN IaaS Manager to OM, and *iii)* Delay from OM to OAs. During our experiments we found that the delay from ontology application to WSN IaaS Manager is negligible since both entities were on the same laptop. Therefore this delay is not included in the given results. The delay from WSN IaaS Manager is shown in vertical lines in Fig. 8. The average delay is ~56ms. The delay from OM to OA is shown as dots in Fig. 8 and on the average it is about 54ms. In total the average ODisT from 50 experiments is around 109ms.

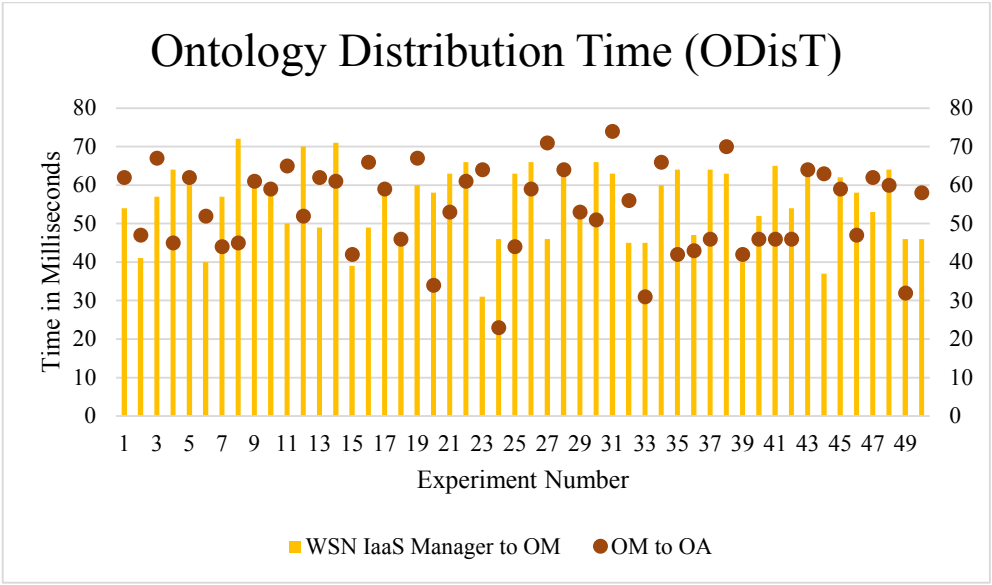


Figure 8: Overlay Creation Time

Figure 9 shows the ODT when AA requested for the required part of the base ontology and received the corresponding owl file. The average ODT from 50 experiments is *~137ms*. The reason for higher ODT as compared to the delay from WSN IaaS Manager to OM and delay from OM to OA is because ODT includes the request and reply delay as AA first sent a request for the ontology file and later received it where as for the other delays there was no

request message, WSN IaaS Manager and OM simply sent the ontology file to the destination without receiving any request message.

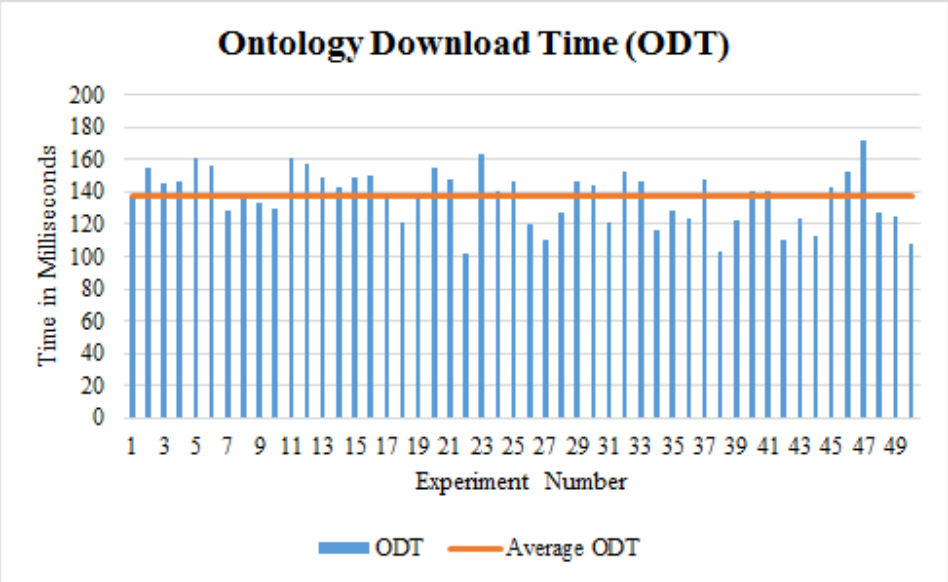


Figure 9: Ontology Download Time

6. Future Work and Conclusion

We have identified couple of work items as future work. First is to explore the possibility of using semantic web for the efficient publication and discovery of the deployed WSN IaaS. So far we have only focused on the annotation of sensor data in real-time but it would be interesting to find whether semantic web can help in publishing and discovering sensors and their services in a virtualized WSN IaaS. This will provide a standard way of advertising the capabilities and services of a deployment and make it easier for interested users to easily discover sensors according to their requirements.

Utilization of a deployed WSN by multiple applications and services potentially opens avenues to new business models and innovation. Virtualized WSNs make such utilization reality by allowing multiple application and services to use deployed sensors for their tasks concurrently. Semantic-based WSN applications are more useful for their end-users who instead of getting simple event notifications are able to get event details at a higher level of abstraction and understand the context as well. In this paper our previous work is extended with new architectural enhancements to allow a WSN infrastructure owner to easily create and manage ontologies related to the deployed infrastructure. These developed ontologies are then used by sensors to annotate their data independent of any application domain. Furthermore, we used a simple heuristic-based genetic algorithm to select capable nodes in the WSN to store the different ontology files and provide them for annotation whenever required. A proof-of-concept prototype is developed to show the feasibility of the proposed architecture.

This paper is work-in-progress

References

[1] – I. Khan, et al., "Wireless Sensor Network Virtualization: A Survey," *Communications Surveys & Tutorials, IEEE* , vol.PP, no.99, pp.1,1, doi: 10.1109/COMST.2015.2412971, March 2015.

[2] – I. Khan, et al., "Wireless Sensor Network Virtualization: Early Architecture and Research Perspectives", *IEEE Network Magazine, (accepted for publication), in-press*. May/June 2015

[3] – I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, "SenShare: Transforming Sensor Networks into Multi-application Sensing Infrastructures," in Wireless Sensor Networks, G. P. Picco and W. Heinzelman, Eds. Springer Berlin Heidelberg, 2012, pp. 65–81.

[4] – M. Navarro, M. Antonucci, L. Sarakis, and T. Zahariadis, "VITRO Architecture: Bringing Virtualization to WSN World," in 2011 IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS), 2011, pp. 831–836.

[5] – J. Yick, B. Mukherjee, and D. Ghosal. "Wireless sensor network survey." *Computer networks* 52.12 (2008): 2292-2330.

[6] – M. A. Feki, F. Kawsar, M. Boussard, and L. Trappeniers, "The Internet of Things: The Next Technological Revolution," Computer, vol. 46, no. 2, pp. 24–25, 2013.

[7] – I. Khan, et al., "Getting Virtualized Wireless Sensor Networks' IaaS Ready for PaaS", accepted for publication in IoTIP-15 Workshop in 11th IEEE International Conference on Distributed Computing in Sensor Systems (IEEE DCOSS 2015) conference, 2015, June 10-12, Fortaleza, Brazil

[8] – I. Khan, et al., "A Data Annotation Architecture for Semantic Applications in Virtualized Wireless Sensor Networks", in proceedings of 14th IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2015) – Technical Session, 2015, May 11-15, Ottawa, Canada.

[9] – L. Sanchez, et al., "SmartSantander: IoT experimentation over a smart city testbed," Computer Networks, vol. 61, pp. 217–238, Mar. 2014.

[10] – A. Sheth, C. Henson, and S. S. Sahoo, "Semantic Sensor Web," IEEE Internet Computing, vol. 12, no. 4, pp. 78–83, Jul. 2008.

[11] – A. Zafeiropoulos, et al., "A Semantic-Based Architecture for Sensor Data Fusion," in The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBICOMM '08, 2008, pp. 116–121.

[12] – D. Pfisterer, et al., "SPITFIRE: toward a semantic web of things," IEEE Communications Magazine, vol. 49, no. 11, pp. 40–48, Nov. 2011.

[13] – A. Gyrard, C. Bonnet, and K. Boudaoud, "A machine-to-machine architecture to merge semantic sensor measurements," in WWW 2013, 22nd International World Wide Web Conference, Doctoral Consortium, May 13-17, 2013, Rio de Janeiro, Brazil, 2013.

[14] – A. Gyrard, C. Bonnet, and K. Boudaoud, "Enrich machine-to-machine data with semantic web technologies for cross-domain applications," in 2014 IEEE World Forum on Internet of Things (WF-IoT), 2014, pp. 559–564.

[15] – X. Su, H. Zhang, J. Riekki, A. Keränen, J. K. Nurminen, and L. Du, "Connecting IoT Sensors to Knowledge-based Systems by Transforming SenML to RDF," Procedia Comput. Sci., vol. 32, pp. 215–222, 2014.

This paper is work-in-progress

[16] – M. Liu, T. Leppanen, E. Harjula, Z. Ou, A. Ramalingam, M. Ylianttila, and T. Ojala, "Distributed resource directory architecture in Machine-to-Machine communications," in 2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2013, pp. 319–324.

[17] – J. Mäenpää, J. J. Bolonio, and S. Loreto, "Using RELOAD and CoAP for wide area sensor and actuator networking," J Wireless Com Network, vol. 2012, no. 1, pp. 1–22, Dec. 2012.

[18] – E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," IEEE Communications Surveys Tutorials, vol. 7, no. 2, pp. 72–93, Second 2005.

[19] – F. Crivellaro, "μJena: Gestione di ontologie sui dispositivi mobile," M.Sc., Politecnico di Milano, Milan, Italy, 2007.

Annex H

Paper VIII

A Genetic Algorithm-based Solution for Efficient In-network Sensor Data Annotation in Virtualized Wireless Sensor Networks

Imran Khan*, Jagruti Sahoo†, Son Han*, Roch Glitho† and Noël Crespi*
*Institut Minés-Télécom, Télécom SudParis, 91011 Evry Cedex, France
Email: imran@ieee.org, {son.han, noel.crespi} @it-sudparis.eu
†Dept. CIISE, Concordia University, H3G 2W1, Montreal, Canada
Email: jagrutiss@gmail.com, glitho@ciise.concordia.ca

***Abstract*—Sharing a deployed Wireless Sensor Network (WSN) infrastructure among multiple, concurrent applications can help realize the true potential of Internet-of-Things (IoT). Virtualized WSNs can help to achieve such sharing where multiple applications and services use a deployed WSN infrastructure at the same time. These applications and services may include semantic applications which are very much pertinent to provide situational awareness to the end-users who are then able to understand the context of the events and make informed decisions. However there is a fundamental issue of performing sensor data annotation in an efficient manner in such networks. In this paper we propose a heuristic-based genetic algorithm to select capable nodes to perform in-network sensor data annotation in virtualized WSN in a way that maximizes energy and storage efficiency. Simulation results are also presented.**

***Keywords*—Wireless Sensor Networks; Internet of Things; Semantic Web; WSN Virtualization; Genetic Algorithm**

I. INTRODUCTION

Recently the concept of Wireless Sensor Networks (WSNs) virtualization [1], [2], [3], [4] has gained attention that uses the concept of multiple concurrent application tasks running on a sensor node. With this concept it has become possible to offer a deployed WSN infrastructure to multiple applications and services. This is in contrast to the traditional task-oriented, domain-specific deployments of WSNs [5] where applications came usually bundled with them. Normally it is prohibitively expensive and time-consuming to deploy new applications over the traditional deployments. WSNs are considered as one of the building blocks of the Internet-of-Things (IoT) paradigm [6] hence it is pertinent to explore the possibility of sharing them among multiple applications and services.

Traditional WSN applications are built around the concept of receiving sensor data in raw format without any ability to understand its context and meaning. Additionally, this raw sensor data fails to provide high-level details to gain situational awareness because an end-user cannot make queries to better understand a situation. For example, a traditional fire monitoring application can get only a notification about the fire eruption event but will not allow its user to query for the details like *where is the source of fire?* Semantic applications, on the other hand, easily allow end-users to make such queries to get results like *in a public library*. This allows for provisioning more rich and interactive applications to the end-users. Another benefit of incorporating semantic concept to the WSNs is that we can have standard way to share sensor data across different application domains. This can be particularly useful to achieve interoperability among various vertical solutions which are typically found these days. With the increase in number of WSN deployments by the third-party actors (public, community and research deployments), future WSN deployments will have to support both traditional as well as semantic applications.

With this background, it is clear that we need efficient solutions to annotate sensor data using ontology concepts. Recently we proposed an in-network, distributed sensor data annotation architecture [7] to provision traditional as well as semantic applications over a vWSN IaaS. Capable nodes in the architecture store the ontology concepts, which are later used for annotation purposes. This in-network annotation approach has many benefits than existing centralized solutions that first store sensor data and later annotate it. For example, each capable sensor is able to annotate its data directly in real-time. However, being resource constrained networks, it makes sense to use only limited number of sensors to perform intensive tasks such as storing the ontology concepts and sharing them for the annotation purposes. There needs to be a simple yet efficient mechanism to select a set of capable nodes that store ontology concepts.

In this paper we propose a heuristic-based Genetic Algorithm (GA) to select capable nodes for storing the developed ontology. We use multi-objective criteria to select best possible candidates (including capable sensors) for ontology storage. The GA is designed to select sensors with maximum energy and storage space available at that time. We use two level encoding scheme to model the problem. Simulations results of the implementation are also presented.

The rest of the paper is organized as follow; a motivating scenario is presented in Section II, along with a set of requirements and brief overview of our previous work that we use as basis for this work. In Section III proposed algorithm is described in detail. Section IV presents the simulation results while related work is discussed in Section V. Finally Section VI concludes the paper along with discussion on future work.

II. Background, Motivating Scenario and Requirements

In this section we first present overview of our architecture that has been used as basis for this work. Later we present a simple motivating scenario to show the problem addressed in this paper. Finally a set of requirements is drawn from the scenario which should be fulfilled by a solution.

A. Our Starting Point

The work in this paper is based on our previous WSN virtualization architecture [7] which is illustrated in Fig. 1. The architecture consists of four layers. The physical layer consists of sensor nodes that support node-level virtualization. Both resource-constrained (e.g. TelosB, called Type A) as well as capable (e.g. Java SunSpots, called Type B) sensor nodes are considered. Capable sensors as well as high-end machines (e.g. base stations and sink nodes) act as Gates-to-Overlays (GTO) nodes to facilitate resource-constrained sensors to support node-level virtualization. The second layer is Virtual Sensor layer that abstracts as virtual sensors, the simultaneous tasks run by the physical sensors. There can be two types of virtual sensors: ones running semantic application tasks (and require data annotation), called semantic virtual sensors and ones running non-semantic application tasks, called virtual sensors.

The Virtual Sensor Access layer has three functional entities (Annotation Agents (AAs), Ontology Agents (OAs) and Sensor Agents (SAs)) and two overlays (Annotation and Ontology overlays). The Annotation overlay consists of AAs, which annotate sensor data using the standard ontology. Each semantic virtual sensor is represented by a corresponding AA in the Annotation overlay. Also in the same overlay are the SAs, which receive annotated as well as non-annotated data from the virtual sensors and forward it to the end applications. The Ontology overlay consists of OAs that store the standard ontology. These OAs act as super-peers and provide the ontology to the requesting AAs. Final layer is Application Overlay layer which consists of multiple applications (semantic and non-semantic) over the deployed vWSN IaaS.

The architecture is based on the following assumptions: first it is assumed that the sensors have already been discovered and are stored in a registration server. Applications and services that wish to utilize the sensors send queries to the registration server. There are several existing works such as [8], [9] to accomplish this. Second assumption is that the architecture does not store the sensor data (either raw or annotated). While it is perfectly possible to store sensor data and use it for data analytics and visualization when required, but currently this feature is not provided.

The key challenge that we addressed in our previous architecture, was to provide in-network sensor data annotation in a distributed manner in real-time unlike existing centralized solutions that first store the sensor data and later annotate it. Another contribution was to make our proposed solution domain/application independent since it is difficult to determine the type applications using WSN IaaS. This was achieved by using the concept of base ontology (related to the deployed infrastructure) for sensor data annotation.

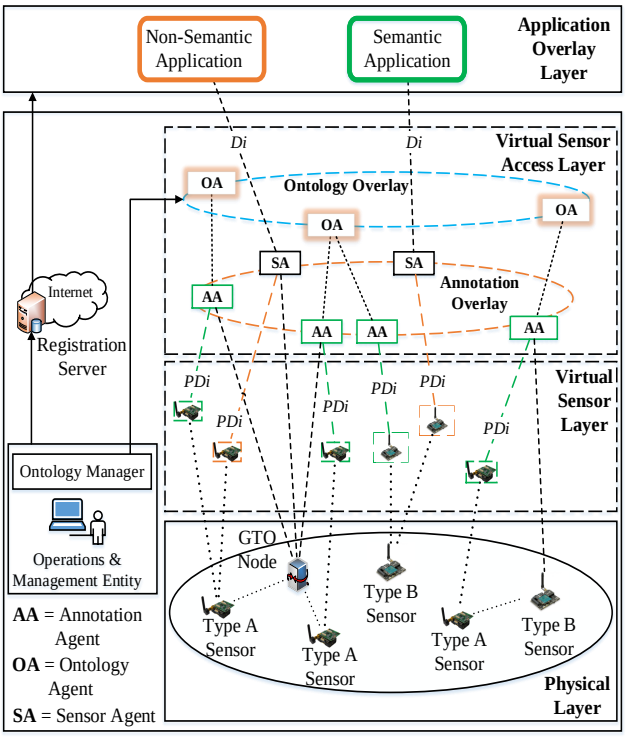


Fig. 1. Data annotation architecture

B. Motivating Scenario

Let us assume that a WSN Infrastructure owner deploys its heterogeneous sensors having different capabilities on a large geographic area to detect different physical phenomena. In the context of IoT such WSN Infrastructure owner can be a city administration interested to provide smart city services to its citizens or a large scale R&D research project such as SmartSantander [10]. In these situations, the infrastructure owner is interested to offer the deployed WSN as IaaS to users to provision multiple applications and services over it. Some of these could be semantic-based allowing their users to infer additional knowledge about the detected physical phenomenon.

Now according to the architecture presented in previous section, base ontology concepts need to be stored in the WSN in a distributed manner. The failure prone and energy deficient nature of WSN mandates that only few of the sensor nodes, that have required energy and storage space available, be used for storing the base ontology.

C. Requirements

Based on the scenario described above, we derive the following three requirements. *First* requirement is that any proposed solution should be applicable to large scale deployment of sensors.

The *second* requirement is that the proposed solution should try to achieve multiple objectives at the same time. For example, energy level of sensors, available storage space among others.

The *third* requirement is that the proposed solution should be able to provide a set of sensors that will store the ontology along with associated sensors that will request and receive the ontology from them whenever required.

III. Multi-Objective Genetic Algorithm for Selecting Capable Nodes

In this section we discuss the multi-objective GA for the selection of capable nodes that will act as OAs.

A. Genetic Algorithm

Genetic Algorithm (GA) [11], [12] follow the process of natural evolution by applying the principle of survival of the fittest. GA works in an iterative manner that mimics the natural selection of the fittest and elimination of the week solutions. In each iteration the solutions are evaluated against a fitness function, the ones that fulfil the criteria of the fitness function are retained while others are screened out. Genetic operations such as crossover, mutation are performed on the fittest solutions to produce new generation of solutions. This whole process is repeated until a certain condition is met. During its execution, GA does not need any other input except the fitness value to select most suitable and fittest solutions. In literature, GA have been used for solving many optimization and selection problems where the focus is to find many near-optimal non-dominated solutions.

B. Capable Node Selection Problem

In our proposed architecture, ontology concepts are stored in multiple capable sensor nodes in a distributed way. In order to have a dynamic and lightweight solution, we propose to select a set of capable nodes to store the ontology concepts (i.e. act as OAs). In this work, we assume that all AAs are able to act as OAs (in terms of capabilities). However, due to limited resources of AAs (energy and memory), it is necessary to select an optimal number of them that will act as OAs. This selection needs to ensure that only nodes that fulfil the energy and storage requirements, at that particular time, are selected. Once a set of nodes is identified, ontology concepts are provided to them. Thus in this work we have energy-related requirements and memory related requirements. The optimal selection of OAs can be modelled as a multi-objective optimization problem where the objectives include maximizing residual energy and maximizing residual storage. The solution to this problem provides the OAs and their respective AAs. We use the GA to solve the optimization problem. This genetic algorithm is executed by a central node, i.e. Ontology Manager (OM) shown in Fig. 1.

C. Problem Representation

We propose a two-level encoding scheme to encode a chromosome for GA, illustrated in Fig. 2. The level-1 encoding is used to represent the OAs; whereas the level-2 encoding is used to represent the members (i.e. AAs) of each OA. The two-level encoding is based on binary encoding. The level-1 encoding consists of n binary bits, where n is number of active sensors in the network. In the chromosome each gene represents a sensor. A value of 0 in level-1 encoding means

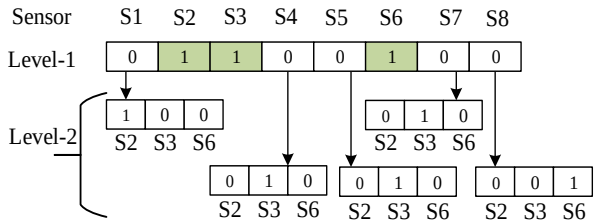


Fig. 2. Two-level encoding example

that the sensor is not selected to act as OA whereas a value of 1 means it is selected to act as OA.

The level-2 encoding is described as follows. Each gene in level-1 encoding that has 0 bit has a level-2 binary string. Each such string consists of m bits where m is equal to the number of 1 bits in level-1 encoding. To encode this, a position is randomly chosen between 1 and m and is filled with 1. Rest m-1 positions are filled with 0s. Note that, numeric encoding could have been used for level-2 encoding. However, binary encoding has an advantage of being flexible in performing mutation operation as described later. The steps of the encoding procedure are as follows.

`/* Level-1 Encoding */`

Generate a random binary strings ;
Randomly fill the bits with $1s$ *and* $0s$ *;*

`/* Level-2 Encoding */`

For each bit with value 0 *in level-1 ;*
Create a binary string of length equal to number of $1's$ *in level-1 ;*
Choose a bit at random and put 1 *;*
Fill all other bits in that string with 0 *;*

IV. GA Operations

Crossover and mutation are the two genetic operations performed on the chromosomes in a population. In a crossover operation, genes from different chromosomes (parents) are recombined to produce new chromosomes (children). The crossover operation ensures that after many generations, best features of the parents are carried to the next generation.

In the genetic algorithm, 2-point crossover is used. In particular, the 2-point crossover operation is applied to level-1 encoding. When a gene is extracted from a parent chromosome, the corresponding level-2 encoding is also extracted. Since the proposed encoding results in variable length level-2 string for each chromosome, the crossover operation must preserve the number of 1 bits in each chromosome. Thus, the basic 2-point crossover operation cannot be applied directly on the chromosomes. One way of resolving this issue is to consider fixed length level-2 strings which also requires pre-specifying the number 1 bits for the level-1 encoding for each chromosome in the initial population. The basic 2-point crossover operation on fixed length level-2 strings is illustrated in Fig 3. The other way is to adopt a variant of 2 point

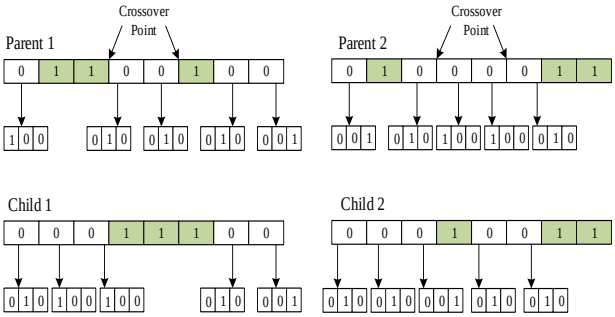


Fig. 3. Example 2-point crossover operation (fixed length level-2 string)

crossover operation [13] which preserves the number of 1 bits in each chromosome that will be produced after crossover.

After the crossover operation, mutation operation is applied to each child. During this operation two genes, selected at random, are interchanged in a chromosome. In our case, the mutation operator is applied in 2 steps to the level-2 strings. In the first step, two genes that have 0 bits in level-1 encoding are selected at random. Then, their level-2 strings are interchanged.

In the second step, a gene that has 0 bit in level-1 encoding is selected at random. Then, in its level-2 binary string, a random position corresponding to a 0 bit is selected. Then, this 0 bit is interchanged with the 1 bit. These steps are illustrated in Fig. 4. Note that, with numeric encoding for level-2 strings, only step-1 can be performed. Thus, using binary encoding for level-2 strings results in flexibility in mutation operation.

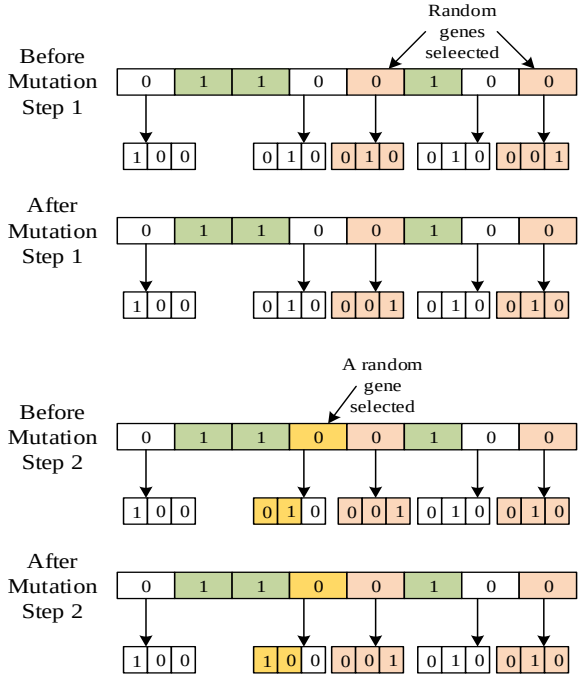


Fig. 4. Example mutation operation

A. Objective Functions

In order to select most promising and fittest individual to produce new generation in each iteration, we use two objectives function that allows us to compare the individuals. Our main objectives are:

f1: Maximizing the residual energy of sensors
f2: Maximizing the residual storage of sensors

These objectives ensure that the total residual energy and total residual memory of sensor nodes are maximized. For computing f1, we adopted the following energy model.

$$E_{Tx} = \begin{cases} l * E_{elect} + l * \varepsilon_{fs} * d^2 & (d < d_0) \\ l * E_{elect} + l * \varepsilon_{mp} * d^4 & (d \geq d_0) \end{cases}$$

Where, E_{Tx} denotes energy consumed in sending 1 bit of data to a node at distance d. E_{elect} is the amount of energy consumption per bit to run the transmitter and receiver circuitry. The details of other parameters can be found in [14].

The objective f1 is expressed as:

$$f1 = \sum_{j=1}^{m} \left[E_{max,j} - \left[\left(\sum_{i=1}^{n_o(j)} E_r \right) + E_j(o) \right] \right]$$

Where,
m = number of sensors selected as OAs,
$E_{max,j}$ *= Current residual of energy of sensor j,*
$no(j)$ *= Number of AAs for which sensor j act as OAs,*
E_r *= Energy spent in communication between two sensors,*
$E_j(O)$ *= Energy spent by sensor j in receiving ontology file*

$E_{max,j}$ is known from the status received from sensors before executing GA. E_r and $E_j(O)$ are computed using the above energy model.

The objective f2 is expressed as:

$$f2 = \sum_{j=1}^{m} \left[M_{max,j} - \left[\left(\sum_{i=1}^{NA} M_j \right) + M_j(o) \right] \right]$$

Where,
m = number of sensors selected as OAs,
$M_{max,j}$ *= Current value of storage of sensor j,*
N_A *= Total no. of application tasks running in sensor j,*
M_j *= Storage needed for applications in sensor j,*
$M_j(O)$ *= Storage needed for ontology file in sensor j*

Since the OA selection problem is a multi-objective optimization problem, there is not one optimal solutions rather a set of solutions called pareto-optimal solutions. However, finding the best or a good trade-off solution is often difficult as it requires a proper analysis of the pareto-front. Therefore, multi-objective optimization problems are often solved using scalarization or weight sum approach which transforms multi-objective optimization to single-objective optimization.

$$Z = W1 * f1 + W2 * f2$$
$$W_1 + W_2 = 1,\ 0 \le W_1, W_2 \le 1$$

Using this approach, the new objective function is expressed as:

Where, W1 and W2 are weights and indicate the relative importance of the objective functions. These weights can be adjusted based on the need.

B. Implementation and Results

In this section we first discuss the simulation setup and then discuss the results.

C. Simulation Setup

We implemented our algorithm using Apache Commons Math library [15]. We execute the algorithm for large number of sensors to get a near-optimal solution containing the list of sensors that can act as OM in the deployed WSN IaaS. The pseudo-code of the algorithm is shown below. The parameters used for the implementation are shown in Table I.

Result: Set of capable nodes that can act as OAs
Input: Population size α
total number of sensors n
crossover probability β
mutation probability γ
number of iterations σ
Output: solution X

```
/* Initialization                                   */
/* Level 1 Encoding                                 */
Generate α random solutions of size n
for i ← 0 to n do
    individual[i] = randomInt[0, 1];
    /* if individual[i] == 1 then it is
       an OA                                        */
    /* else it is not an OA                         */
end
/* Level 2 Encoding                                 */
for each individual k not Selected As OA in Level 1 do
    Create a binary string of length m bits ;
    /* where m is the number of 1s at
       level-1                                      */
    Fill one random bit of the string with 1 ;
    Fill rest of the bits with 0 ;
end
repeat
    crossOver(with β probability);
    mutation(with γ probability);
    fitnessEvalution();
    replaceWithNewGeneration();
    iterations++;
until iterations <= σ;
return Solution
```

Algorithm 1: Capable Node Selection Algorithm

TABLE I. PARAMETERS FOR GENETIC ALGORITHM

Population Size α	1000, 2000, 3000
Crossover Probability β	0.2, 0.5, 0.8
Mutation Probability γ	0.05
Number of Iterations σ	50
Elitism Rate	0.2

Since our work targets capable and advanced sensor platforms, we considered Java SunSpots [16] for our simulation. Java SunSpots have built-in rechargeable Li-ion battery with a total energy of about 9590 joules. The current residual energy of the sensors is fixed at random from 50% to 100% of this value. A uniform value of 50j is assumed for communication between OAs and AAs. Similarly a uniform value of 80j is assumed for OAs to receive ontology file from the central OM node. In our previous work [7] we developed the base ontology with multiple concepts (e.g. temperature, light, carbon, humidity). The maximum ontology file size we had was around 8Kb for a single concept. In this work, we assume a storage space of 10Kb for storing a single ontology file in an OA. In addition to this we consider the scenario where OAs may be executing applications tasks themselves. Here we assume that each OA executes three application tasks. It is important to mention that rev 8 of Java SunSpot provides about 7200Kb of storage space of application tasks. Hence considering 10Kb for ontology storage makes sense. Each experiment was repeated 10 times and the results presented here show the average values of these experiments.

D. Results

The total fitness value of the optimal set of OAs is shown in Fig. 5. It is important to mention that the values are higher because fitness value of all OAs is combined. We observe that the higher population size (i.e. more sensors) do not necessarily lead to maximum fitness value. However, as iterations passed, the fitness became larger. For this result we kept crossover rate at 0.8. For each population size the best solution was found in the last few iterations (e.g. 47th, 48th and 50th).

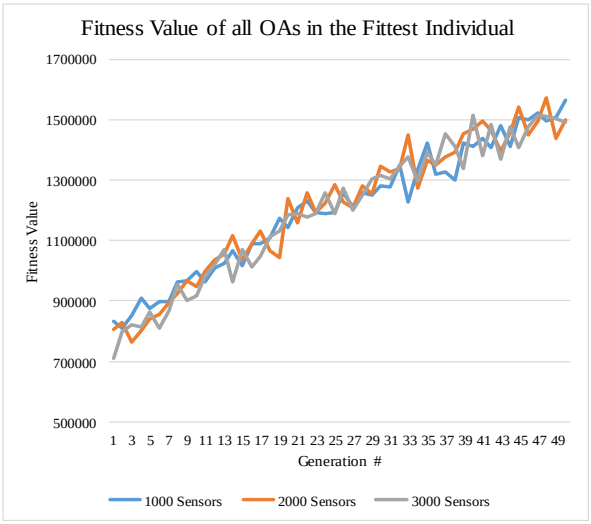


Fig. 5. Fitness value of all OAs in the fittest individual

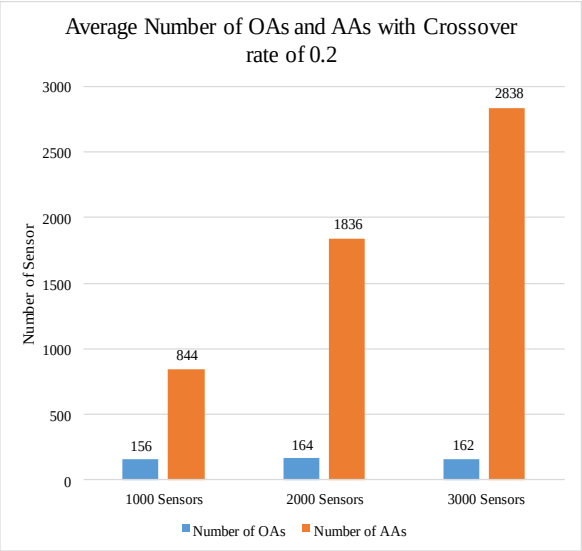


Fig. 6. Average no. of OAs and AAs obtained with crossover rate of 0.2

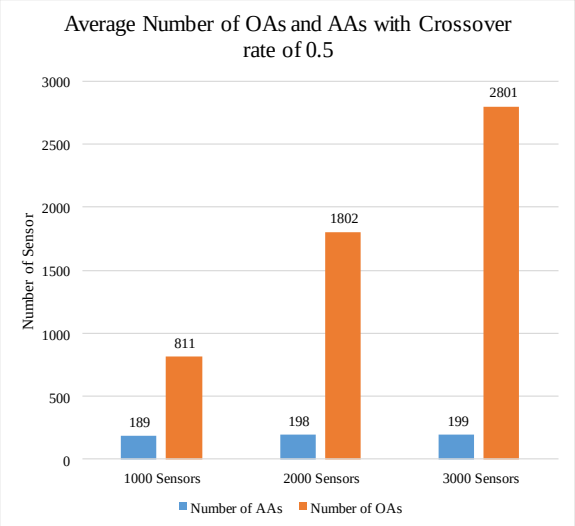


Fig. 7. Average no. of OAs and AAs obtained with crossover rate of 0.5

The next results shows the number of OAs and AAs obtained by using different crossover rates. Fig. 8, 9 and 10 show the average number of OAs and AA obtained using crossover probability of 0.2, 0.5 and 0.8 respectively. It is interesting to note that the low crossover rate leads to less number of OAs. This means that there will be more AAs associated with one OA. As we increased the crossover probability, number of OAs increased as well irrespective of the population size. Another interesting observation is that there is not major increase in number of OAs when population size increases. In fact the number of OAs remain pretty much consistent irrespective of the population size.

We also performed simulation by varying the mutation rate but found that it did not have any major impact.

Overall these results provide us interesting insights and motivate us to further improve the algorithm and solve other research problems in this area.

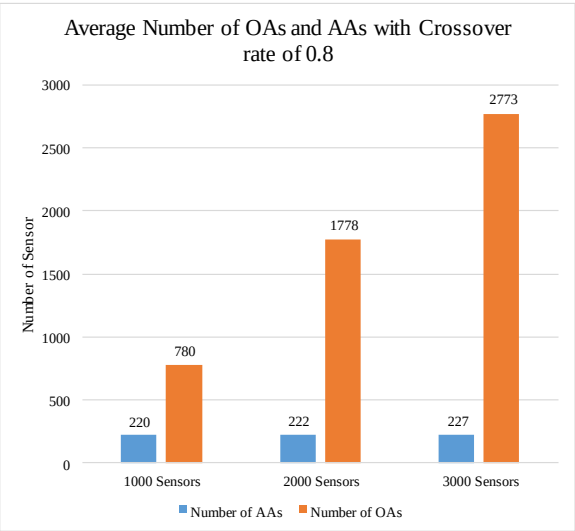


Fig. 8. Average no. of OAs and AAs obtained with crossover rate of 0.8

V. Related Work

The problem addressed in this paper is comparable to some previous work that have used GA in WSN domain. In fact our work is similar to the typical cluster head selection albeit for different purposes. However, majority of the existing works do not associate sensors, having different roles, to each other which is a key requirement in our work.

The work in [17] presents a design optimization solution in WSNs using GA. The authors use multiple objectives to find the optimal operation mode of sensors and assigns them specific roles so that the overall energy consumption is minimized and application-specific requirements are satisfied. The WSN is modelled as a square grid deployment with around 990 sensors. Sensor can either act as cluster-heads or as normal sensors when they are active. However, this work does not provide the association between sensors having different roles as per our third requirement.

In [18] the optimal coverage problem is solved using the multi-objective GA in WSNs. In order to provide maximum possible coverage a subset of sensors need to be active. There is a trade-off, more coverage means more active sensors hence more energy consumption. Using GA the authors tried to find minimum number of sensors required to provide full coverage. However, this work also does not fulfil our third requirement.

In [19] GA is used to create clusters in WSN in an efficient way. The work uses multi-objectives to select the fittest chromosome. As part of the result, suitable cluster heads are identified. This part is similar to our work, however the member nodes for the cluster heads are not identified by the GA, instead a minimum distance strategy is used by a base station node. Therefore this work also does not fulfil out third requirement.

The authors in [20] proposed a GA based approach to perform load balancing for clustering in WSN. The work tries to assign sensors to different gateways in a similar manner to our work (AAs to OAs). However, their solution does not achieve multi-objectives and only considers overlaod on gateway nodes. Hence this work does not satisfy our second requirement.

VI. Conclusion

In this paper we proposed a heuristic-based GA for optimal selection of capable nodes (called OAs) to hold ontology files in a WSN. The algorithm also associates sensors (called AAs) to these capable sensor to get ontology files whenever required. The proposed solution is centralized and is executed by main node in the network.

There are also many avenues to extend this work. For example, distance between OAs and AAs should be considered when associating them with each other. Similarly there needs to be an update mechanism to mitigate the effects of OA failures in the WSN, i.e. whenever OAs fail new ones are selected automatically. Similarly, a network status update mechanism is required to gather the current status of the sensors, their available energy levels, and available storage. A periodic heartbeat mechanism or an enhanced routing mechanism could be used for this purpose. Another possibility is to find approaches which could support localized execution of the GA. This would make it possible to deal with OA failures locally thereby avoiding executing GA on the whole network. Finally, in this work it is assumed that after finding the pairs of OAs and AAs, the OM node disseminates the ontology files to the selected OAs and notifies AAs about their potential OA. However, how this is achieved needs further investigation.

References

[1] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, Wireless Sensor Network Virtualization: A Survey, IEEE Communications Surveys & Tutorials, vol. PP, no. 99, pp. 1-1, 2015.

[2] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, Wireless sensor network virtualization: early architecture and research perspectives, IEEE Network, vol. 29, no. 3, pp. 104-112, May 2015.

[3] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, SenShare: Transforming Sensor Networks into Multi-application Sensing Infrastructures, in Wireless Sensor Networks, G. P. Picco and W. Heinzelman, Eds. Springer Berlin Heidelberg, 2012, pp. 65-81.

[4] M. Navarro, M. Antonucci, L. Sarakis, and T. Zahariadis, VITRO Architecture: Bringing Virtualization to WSN World, in 2011 IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS), 2011, pp. 831-836.

[5] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," Computer Networks, vol. 52, no. 12, pp. 2292-2330, Aug. 2008.

[6] M. A. Feki, F. Kawsar, M. Boussard, and L. Trappeniers, The Internet of Things: The Next Technological Revolution, Computer, vol. 46, no. 2, pp. 24-25, 2013.

[7] I. Khan, et al., "A data annotation architecture for semantic applications in virtualized wireless sensor networks," Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on , vol., no., pp.27,35, 11-15 May 2015.

[8] M. Liu, T. Leppanen, E. Harjula, Z. Ou, A. Ramalingam, M. Ylianttila, and T. Ojala, "Distributed resource directory architecture in Machine-to-Machine communications," in 2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Lyon, France, 2013, pp. 319-324.

[9] J. Menp, J. J. Bolonio, and S. Loreto, "Using RELOAD and CoAP for wide area sensor and actuator networking," J Wireless Com Network, vol. 2012, no. 1, pp. 1-22, Dec. 2012.

[10] L. Sanchez, et al., SmartSantander: IoT experimentation over a smart city testbed, Computer Networks, vol. 61, pp. 217-238, Mar. 2014.

[11] J. H. Holland. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press, 1992.

[12] D. Whitley, "A genetic algorithm tutorial." Statistics and Computing 4.2 (1994): 65-85.

[13] Y.C. Hou, Y.H. Chang, A new efficient encoding mode of genetic algorithms for the generalized plant allocation problem, Journal of Information Science and Engineering 20 (2004) 1019-1034.

[14] J.-M. Kim, S.-H. Park, Y.-J. Han, and T.-M. Chung, CHEF: Cluster Head Election mechanism using Fuzzy logic in Wireless Sensor Networks, in 10th International Conference on Advanced Communication Technology, 2008. ICACT 2008, vol. 1, pp. 654-659.

[15] Apache Software Foundation, Apache commons mathematics library, 2015, http://commons.apache.org/math/

[16] R. B. Smith, "SPOTWorld and the Sun SPOT," in Proceedings of the 6th International Conference on Information Processing in Sensor Networks, New York, NY, USA, 2007, pp. 565-566.

[17] K. P. Ferentinos and T. A. Tsiligiridis, Adaptive design optimization of wireless sensor networks using genetic algorithms, Computer Networks, vol. 51, no. 4, pp. 1031-1051, Mar. 2007.

[18] J. Jia, J. Chen, G. Chang, and Z. Tan, Energy efficient coverage control in wireless sensor networks based on multi-objective genetic algorithm, Computers & Mathematics with Applications, vol. 57, no. 1112, pp. 1756-1766, Jun. 2009.

[19] S. Hussain, A. W. Matin, and O. Islam, "Genetic Algorithm for Energy Efficient Clusters in Wireless Sensor Networks", ITNG, 2007, Information Technology: New Generations, Third International Conference on, Information Technology: New Generations, Third International Conference on 2007, pp. 147-154.

[20] P. Kuila, S. K. Gupta, and P. K. Jana, A novel evolutionary approach for load balanced clustering problem for wireless sensor networks, Swarm and Evolutionary Computation, vol. 12, pp. 48-56, Oct. 2013.

Acronym

AA	Annotation Agents
API	Application Programming Interface
Ci	Control interface
CoAP	Constrained Application Protocol
Di	Data interface
DNS-SD	Domain Name System-based Service Discovery
E2ED	End-to-End Delay
EOT	Expected Operation Time
FCA	Fire Contour Algorithm
FND	Fire Notification Delay
GA	Genetic Algorithm
Gi	Gateway interface
GTO	Gates-to-Overlays
GUI	Graphical User Interface
HPD	HTTP Post Delay
IaaS	Infrastructure-as-a-Service
IETF	Internet Engineering Task Force
IoT	Internet-of-Things
IP	Internet Protocol
IT	Information Technology
JVM	Java Virtual Machine
Kbps	Kilobits per second
LAN	Local Area Network
Mbps	Megabits per second
O&M	Operations & Management
OA	Ontology Agents
OCD	Overlay Creation Delay
ODisT	Ontology Dissemination Time
ODT	Ontology Download Time
OM	Ontology Manager
OS	Operating System
OTA	Over-the-Air
OWL	Web Ontology Language
P2P	Peer-to-Peer
PaaS	Platform-as-a-Service
PAC	Peer Aware Communication
PCi	Proprietary interface
PDi	Proprietary Data interface
RDF	Resource Description Framework
REST	Representational State Transfer
SA	Sensor Agent
SaaS	Software-as-a-Service
SSN	Semantic Sensor Network
TCP	Transmission Control Protocol
VM	Virtual Machine
VS	Virtual Sensor
VSCD	Virtual Sensor Creation Delay
VSN	Virtual Sensor Network
VSST	Virtual Sensor Start Time
vWSN	virtualized Wireless Sensor Network
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network