Fall 2014

Energy efficient scheduling and allocation of tasks in sensor cloud

Rashmi Dalvi

Department:

ENERGY EFFICIENT SCHEDULING AND ALLOCATION OF TASKS IN SENSOR CLOUD

By

RASHMI DALVI

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER SCIENCE

2014

Approved by

Dr. Sanjay Kumar Madria, Advisor
Dr. Sriram Chellappan,
Dr. Maciej Zawodniok

PUBLICATION THESIS OPTION

This thesis consists of the following article that has been published as follows and formatted in University format:

Pages 13-34 published in IEEE journal - Madria Sanjay, Vimal Kumar, and Rashmi Dalvi, “Sensor Cloud: A Cloud of Virtual Sensors.” In Software, IEEE, vol. 31, no. 2, pp. 70-77, IEEE, Mar 2014.

This thesis consists of the following article that will be submitted for publication as follows:

Pages 35-78 to be submitted - Rashmi Dalvi, Sanjay K. Madria “Energy Efficient Scheduling and Allocation of Tasks in Sensor Cloud”

ABSTRACT

Wireless Sensor Network (WSN) is a class of ad hoc networks that has capability of self-organizing, in-network data processing, and unattended environment monitoring. Sensor-cloud is a cloud of heterogeneous WSNs. It is attractive as it can change the computation paradigm of wireless sensor networks. In Sensor-Cloud, to gain profit from underutilized WSNs, multiple WSN owners collaborate to provide a cloud service. Sensor Cloud users can simply rent the sensing services which eliminates the cost of ownership, enabling the usage of large scale sensor networks become affordable. The nature of Sensor-Cloud enables resource sharing and allows virtual sensors to be scaled up or down. It abstracts different platforms hence giving the impression of a homogeneous network. Further in multi-application environment, users of different applications may require data based on different needs. Hence scheduling scheme in WSNs is required which serves maximum users of various applications. We have proposed a scheduling scheme suitable for the multiple applications in Sensor Cloud. Scheduling scheme is based on TDMA which considers fine granularity of tasks. The performance evaluation shows the better response time, throughput and overall energy consumption as compared to the base case we developed. On the other hand, to minimize the energy consumption in WSN, we design an allocation scheme. In Sensor Cloud, we consider sparsely and densely deployed WSNs working together. Also, in a WSN there might be sparsely and densely deployed zones. Based on spatial correlation and with the help of Voronoi diagram, we turn on minimum number of sensors hence increasing WSN lifetime and covering almost 100 percent area. The performance evaluation of allocation scheme shows energy efficiency by selecting fewer nodes in comparison to other work.

ACKNOWLEDGMENTS

At the outset I would like to say that I am highly obliged to my advisor, my thesis committee and my family members. I would like to thank Dr. Sanjay Madria for giving me the opportunity to work on the currently booming technology in the IT world. It was due to his support and guidance that I could complete this work. Further I would like to thank my advisory committee Dr. Chellappan and Dr. Zawodniok for being so considerate and helpful.

I would specially like to thank Dr. Vimal Kumar for his very thoughtful and detailed suggestions and help regarding my work.

I would like to thank Dr. McMillin and Dawn Davis for their guidance in all essential academic procedures.

Finally, I would like to thank my family members for their continued motivation and support and last but not least I thank all my friends who were the best critics and supporters at the same time of my work.

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

1. INTRODUCTION

1.1 CLOUD COMPUTING

Cloud computing refers to the resources offered as a service. There are many definitions on the internet one of which defined in [1] is, “Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction”.

Cloud computing is prevalent than having dedicated resources for applications and provides benefits to cloud user and Cloud Service Provider (CSP). Following are some of the benefits that cloud computing provides to Cloud Users and CSPs.

Advantages to the Cloud User

a) Cost effective than owning the resources
b) No maintenance cost
c) Resources can be provisioned and de-provisioned on demand
d) Provides fault tolerance
e) Resources can be made available quickly
f) Improved flexibility
g) Easy backup and recovery
h) Ubiquitous network access

Advantages to the CSP

a) Idle resources can be utilized and CSP can gain profit out of it
b) Low cost of maintenance

c) Easy to balance load

d) Virtualization and sharing of resources lead to more profit to CSP

1.2 WIRELESS SENSOR NETWORK (WSN)

WSNs are popular because of their capability of building network on its own, they do not require continuous monitoring, and their small size, small memory, 'AAA' battery operated nature makes them cheaper. Moreover, they are programmable which makes them configurable for different applications. Wireless sensors also called as motes can sense physical phenomenon like light, temperature, humidity, radiation, sound, pressure etc. WSNs can be deployed in the area where human reach is difficult for instance, at inhabitable places on the earth, in jungle to monitor behavior of wild animals. They are also used in the medical applications to monitor patient's health such as body temperature, heart beat etc.

Important characteristics of WSNs

a) Scalable network

b) Mobile motes in WSN

c) Provides fault tolerance

d) Heterogeneity of nodes

e) Easy to use

f) Can stand adverse environmental conditions

g) In-network processing

h) Easy deployment

Wireless sensors in WSN can build network with tree or cluster topology. Each WSN will have a Base Station (BS) which is connected to the internet to avail the data to

the application. As compared to all other motes in WSN, BS needs to exhibit more power. Several types of motes are available in market such as micaz, telosb, zigbee having different characteristics. Motes or chosen based on the application they will be used into.

1.3 CHALLENGES PERTAINING TO WSNS

As explained in section 1.2, WSNs have many characteristics which make them suitable for various applications; however they also have some limitations which lead to the research opportunities in this field. Following are some of the challenges.

a) High maintenance cost

b) Limited battery power hence short lifetime

c) Limited memory capacity

d) High ownership cost for large scale network

e) High cost of programming motes

1.4 SENSOR CLOUD

Sensor Cloud Computing is a heterogeneous computing environment, which brings together multiple WSNs, each of which may contain many wireless sensors owned by different entities. In this computing paradigm, the users do not need to own the sensor network before using it. They can simply buy the sensing services from the Sensor Cloud. From a user's point of view, since the amount of investment goes down, usage of large scale sensor networks becomes affordable. Similar to cloud computing, resources in Sensor Clouds can be dynamically provisioned and de-provisioned on demand, providing greater flexibility of operations. Sensor Cloud is a cloud of heterogeneous WSNs providing homogeneous access to the user. User can access real time data just by subscribing to the

Sensor Cloud web service; this feature scales down the cost of owning, programming and maintaining the Wireless Sensor Network. Users can access the data from huge topographical region with minimal cost. At the same time Sensor Cloud provides reliability, scalability, fault tolerance and flexibility to receive data from WSNs without dependency on types of wireless sensors used. Data sharing among multiple users not only benefits the end user but also the service provider.

1.5 MOTIVATION FOR SCHEDULING AND ALLOCATION OF TASKS

In Sensor Cloud scenario, WSNs can have different types of tasks and some may be requested at the same instance of time. In such cases, it is important that WSNs should serve maximum possible tasks gaining profit to the CSP and providing user satisfaction at the same time. Hence for the sensors in WSN, an optimal scheduling scheme has to be developed. On the other hand, due to the limitation of battery power, short life span of wireless sensors, and considering spatial correlation between the adjacent sensors, it is unnecessary to activate all sensors in the region where they are densely deployed. Also if sensors are sparsely deployed in a region, it is necessary that all sensors are activated, hence allocation scheme is needed in Sensor Cloud environment.

1.6 CONTRIBUTION OF THE THESIS

The thesis is presented in a paper based format. Hence the main contributions discussed in the two papers are as follows:

Paper I:

- We have explained the layered architecture of Sensor Cloud

- We defined the concept of virtualization in Sensor Cloud and their relationship between sensors such as one to many, many to one, and many to many.
- We implemented the Sensor Cloud named as “Missouri S&T Sensor Cloud” and presented the detailed block diagram.

Paper II

- We proposed the scheduling and allocation schemes for sensors in Sensor Cloud and explained in detail
- Next we designed the algorithms for both the schemes
- Finally the results of experimental evaluations are presented in the paper

1.7 ORGANIZATION OF THE THESIS

The thesis is mainly presented in paper format. The thesis is organized as follows; Section 1 provides the introduction, and the background. Section 2 presents the related work. Section 3 comprises of Paper I on Sensor Cloud: A Cloud of Virtual Sensors. And Paper II on Energy Efficient Scheduling and Allocation of Tasks in Sensor Cloud. Section 4 contains the Conclusion.

2. RELATED WORK

Sensor Cloud is the cloud of virtual sensors built on top of physical WSNs. Many researchers have come up with various architectures for Sensor Cloud. Moreover in WSNs, scheduling and allocation of sensors is an important issue because of limited resources such as battery power, computation capacity etc. To address these issues many schemes have been proposed. This section briefly discusses the existing work in the field of Sensor Cloud and scheduling and allocation of tasks in Sensor Cloud.

2.1 A FRAMEWORK OF SENSOR – CLOUD INTEGRATION OPPORTUNITIES AND CHALLENGES

Paper [2] defines the framework and architecture of Sensor cloud. However, their architecture is completely different than the architecture we have proposed in our work, in that they have used pub/sub based model. Sensor data are coming through gateways to a pub/sub broker. Pub sub broker is required in their system to deliver information to the consumers of SaaS applications as the entire network is very dynamic. Also, they considered the inter networking between Cloud Providers (CLP) and then they proposed to have Service Level Agreements (SLAs) in case of violation.

2.2 SENSOR-CLOUD INFRASTRUCTURE

In paper [3], they discussed various components that constitute a Sensor Cloud system, its management, and the control flow of various components. Figure 2.1 shows the control flow they have proposed for Sensor-Cloud.

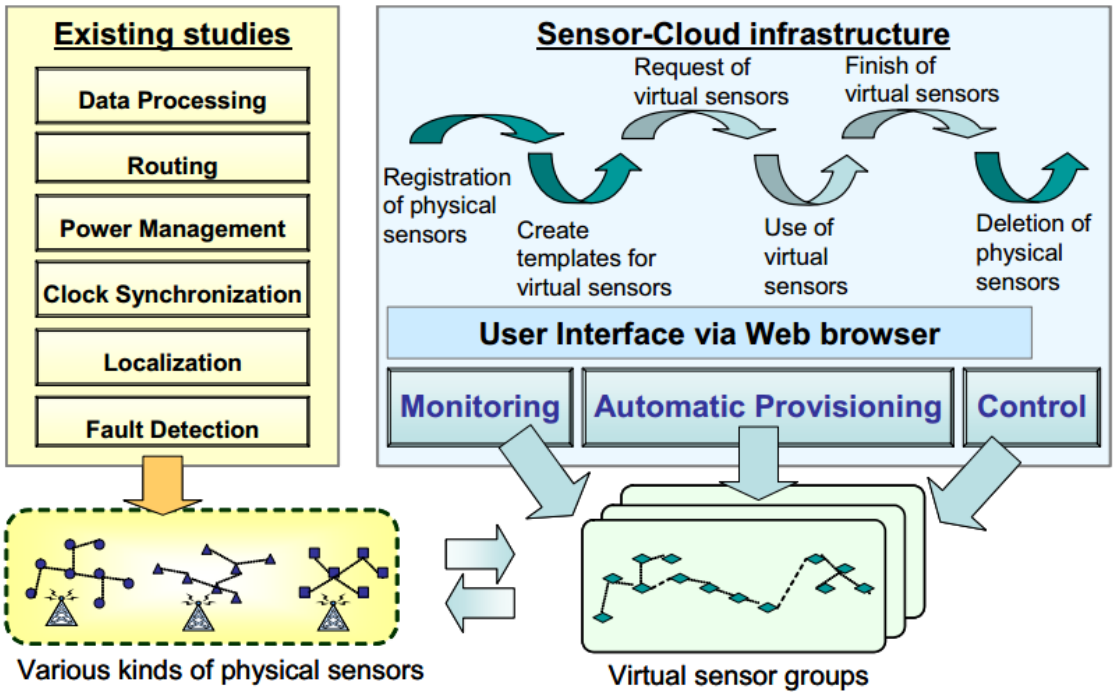


Figure 2.1 Overview of Sensor Cloud Infrastructure [3]

In the proposed infrastructure, they considered three types of actors Sensor Owner, Sensor Admin and End User. Sensor admin builds virtual sensors on top of physical sensors owned by sensor owners. In their concept of virtualization, on the basis of demand from end user, virtual sensors are provisioned and de-provisioned by the sensor admin. They believed if multiple end users control physical sensors, it may result into inconsistent commands. Hence, between physical and virtual sensors, they considered many to one relationship, but they did not consider one to many and many to many relationships. Moreover, they did not implement the Sensor Cloud but defined only the architecture.

2.3 VIRTUAL SENSORS: ABSTRACTING DATA FROM PHYSICAL SENSORS

The paper [4] describes the abstraction of virtual sensors from physical sensors. Instead of simply aggregating the data from sensors, in their approach, they have focused on aggregating and processing the abstract data. The virtual sensor can include multiple heterogeneous physical sensors from which information is abstracted.

First, the developer defines the data requests in the form of classes from the respective applications with the help of virtual sensors. Then a program dynamically interacts with the virtual sensors, extracting required data which eventually serves the application.

2.4 A MIDDLEWARE FOR FAST AND FLEXIBLE SENSOR NETWORK DEPLOYMENT

In paper [5] authors have proposed Global sensor networks (GSN) which is a middleware designed to rapidly deploy heterogeneous wireless sensors. GSN also relies on virtual sensors; however, unlike other approaches where virtual sensors are defined using classes, the virtual sensors in GSN are defined using XML. GSN, the closest approach to our Sensor Cloud model, offers a ready-to-use system, which can integrate large number of wireless sensors networks. However, it can’t be classified as a Sensor Cloud because GSN’s purpose is to provide efficient and flexible deployment of multiple stand-alone WSNs.

2.5 OPTIMIZING PUSH/PULL ENVELOPES FOR ENERGY-EFFICIENT CLOUD-SENSOR SYSTEMS

This [6] work presents the technique to bring down sensing latency and energy consumption in WSNs. They have presented a performance model for sensor-based system

(SBS) on top of three tier architecture consisting of Cloud, Edge and Beneath (CEB) layers. Hence, in their scheme they have proposed a concept of Optimal Push/Pull envelop (PPE) which optimally adjusts the rate of push/ pull actions of each sensor. Figure 2.2 shows the model of three layered architecture.

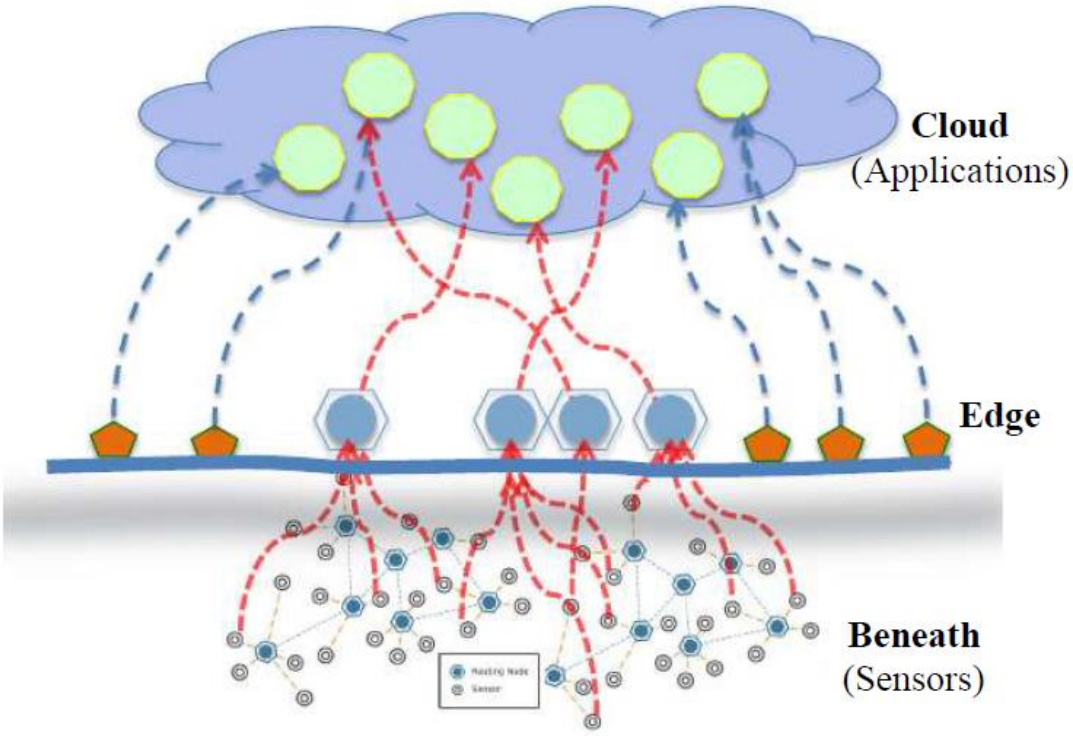


Figure 2.2 Three-layer model of sensor computing with cloud, edge and beneath [6]

In the scheme [6], Sensor Management is performed by Edges so locality is preserved. Edges can abstract unnecessary information about sensors and support staging and optimization by caching historical data. In addition, they have proposed OPT1 and OPT2 algorithms to achieve optimization. In our work, we have extended their concept of push and pull tasks; task T1 in our work is similar to push requests however task T2 is to pull request. We have also considered task T3 event based requests, which are different from push and pull.

2.6 MULTIPLE TASK SCHEDULING FOR LOW-DUTY-CYCLED WIRELESS SENSOR NETWORKS

A task scheduling technique is proposed in [7], which uses scheduling of tasks to solve problem of load balancing in low duty cycled WSNs. They proved that load balancing problem is NP-Complete and proposed greedy algorithm and heuristic algorithm to balance the load and schedule the sensors optimally. However, they assumed that the sensors can serve multiple tasks at the same time. Moreover, the scheme is much more dynamic, but they did not account for the finer granularity of the tasks, and how would scheme will work if length of the tasks are unequal, is not clear.

2.7 SCHEDULING ON WIRELESS SENSOR NETWORKS HOSTING MULTIPLE APPLICATIONS

Paper [8], addresses the problem of allocation and scheduling of sensors in a multi-application environment which eventually leads to energy conservation in WSNs. Assumption behind the proposed system is that each sensor can be utilized by a single application at a time; which brings forth the need of allocation and scheduling algorithms in a Sensor Cloud. While designing algorithms, various parameters such as hop distance, energy requirement, potential energy are considered. Algorithms are classified as Knowledge free and Knowledge based. Furthermore, the paper provides statistical information to prove that knowledge based algorithms outperform knowledge free algorithms. Allocation algorithm is to reduce the energy consumption, whereas scheduling algorithm is to reduce the response time of sensors. However, the proposed approach is not suitable in a Sensor Cloud where many application users might be interested into data from the same geographical region.

2.8 SCHEDULING NODES IN WIRELESS SENSOR NETWORKS: A VORONOI APPROACH

In [9], authors have proposed the allocation scheme in which, using Voronoi diagram some wireless sensors close to each other turn off to increase the overall lifetime of WSNs. Their scheme is designed only for densely deployed WSNs. However, in our scheme, WSNs can have sparsely and/ or densely deployed regions at the same time. More number of sensors are selected from sparsely deployed regions and less from densely deployed regions. Thus, our scheme will work in a Sensor Cloud scenario irrespective of the type of sensor deployment in individual WSN.

Figure 2.3 shows an example of the algorithm where Node 4 is eliminated by algorithm from the sensing activity because the area covered by Node 4 is completely covered by the area covered by Nodes 1, 2, 3 and 5.

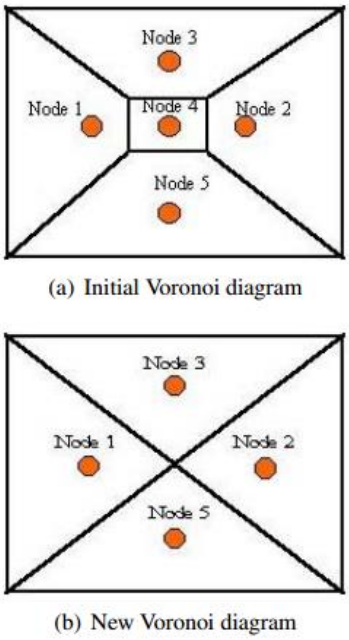


Figure 2.3- Example of the algorithm that uses Voronoi diagram to decide if a node should be turned off or on [9]

2.9 PARTICLE SWARM OPTIMIZATION AND VORONOI DIAGRAM FOR WIRELESS SENSOR NETWORKS COVERAGE OPTIMIZATION

In the allocation scheme proposed in [10], authors are using PSO (Particle Swarm Optimization) along with Voronoi diagram to select the nodes from ROI (Region of Interest). PSO is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. In particular, their solution is not suitable in a Sensor Cloud scenario where we expect WSNs to have both sparsely and densely deployed regions.

PAPER

I. SENSOR CLOUD: A CLOUD OF VIRTUAL SENSORS

Sanjay K. Madria, Vimal Kumar, Rashmi Dalvi

Department of Computer Science, Missouri S & T, Rolla, MO

ABSTRACT

While the use of wireless sensors is growing, their full potential remains to be utilized. One reason for this is the traditional model of computing which is used to interact with wireless sensors. The traditional model of computing imposes restrictions on how efficiently wireless sensors can be used. Newer models for interacting with wireless sensors such as Internet of Things and Sensor Cloud aim to overcome these restrictions. In this paper we present the Missouri S&T Sensor Cloud that we have developed. The Missouri S&T Sensor Cloud enables networks spread in a huge geographical area to connect together to be used by multiple users at the same time. We also present the concept of virtual sensors which is at the core of our Sensor Cloud. We further discuss how virtual sensors assist in creating a multi user environment on top of resource constrained physical wireless sensors.

1. INTRODUCTION

The Industrial WIRELESS sensor network market is expected to grow at a yearly rate of 43.1 percent and reach US$3.795 billion by the year 2017 [8]. Although the usage of wireless sensors continues to grow, their full potential is bounded by the model of computation used to handle them. In traditional wireless sensor network (WSN) applications, a user needs to own a WSN, program the wireless sensors, deploy them and spend time and resources to maintain the network. The user is also restricted to one application per sensor network.

In this article, we describe a new paradigm of computation for wireless sensor networks, the Sensor Cloud, which decouples the network owner and the user and allows multiple WSNs to interoperate at the same time for a single or multiple applications that are transparent to users. We define a Sensor Cloud as a heterogeneous computing environment spread in a wide geographical area that brings together multiple WSNs consisting of different sensors. Each WSN can have a different owner. The Sensor Cloud then virtualizes the wireless sensors and provides sensing as a service to users. Because users buy sensing services on demand from the Sensor Cloud, use of large-scale sensor networks becomes affordable with ease of use.

A Sensor Cloud is composed of virtual sensors built on top of physical wireless sensors, which users automatically and dynamically can provision or de-provision on the basis of applications' demands. This approach has a number of advantages. First, it enables better sensor management capability. The users can use and control their view of WSNs with standard functions for a variety of parameters such as region of interest, sampling frequency, latency, and security. Second, data captured by WSNs can be shared among

multiple users, which reduces the overall cost of data collection for both the system and user. Because data reusability in WSNs is transparent to the Sensor Cloud users, redundant data capture is reduced, thus increasing efficiency. Third, the system is transparent regarding the types of sensors used. The user doesn't need to worry about low-level details such as which types of motes and sensors are used and how to configure them; the Sensor Cloud automatically handles these details.

As a running example for the rest of this article, we'll consider the following scenario. Traffic flow sensors are widely deployed in large numbers in places, including Washington, D.C., and Ohio. These sensors are mounted on traffic lights and provide real-time traffic flow data. Drivers can use this data to better plan their trips. In addition, if the traffic flow sensors are augmented with low-cost humidity and temperature sensors, they can provide a customized and local view of temperature and heat index data on demand. The National Weather Service, on the other hand, uses a single weather station to collect environmental data for a large area, which might not accurately represent an entire region.

2. RELATED WORK IN SENSOR CLOUD

We must note that our definition of a Sensor Cloud is different from others, [1] and is an extension of the concept of the Internet of Things. [2] The Internet of Things integrates the services provided by sensing devices with cloud computing over the Internet. Our Sensor Cloud, on the other hand, is a cloud of virtual sensors built on top of physical sensors, and it provisions virtual sensors to the user for providing sensing as a service. Gerd Kortuem and his colleagues discussed various components that constitute a Sensor Cloud system, its management, and the control flow of various components. [3] Nayot Poolsappasit and his colleagues provided a layered architecture of a Sensor Cloud and outlined the security challenges of such a system. [4] Sanem Kabadayi and her colleagues first described the abstraction of virtual sensors. [5] Navdeep Kaur Kapoor and her colleagues designed the allocation algorithm to reduce the energy consumption, and scheduling algorithms are designed to reduce the response time of sensors in multi-application scenarios. [6] Global sensor networks (GSN) is a middleware designed to rapidly deploy heterogeneous wireless sensors. [7] GSN also relies on virtual sensors; however, unlike approaches where virtual sensors are defined using classes, [5] the virtual sensors in GSN are defined using XML. GSN, the closest approach to our Sensor Cloud, offers a ready-to-use system, which can integrate large number of wireless sensors networks. However, it can't be classified as a Sensor Cloud because GSN's purpose is to provide efficient and flexible deployment of multiple stand-alone WSNs.

3. VIRTUAL SENSORS

A virtual sensor is an emulation of a physical sensor that obtains its data from underlying physical sensors. Virtual sensors provide a customized view to users using distribution and location transparency. In wireless sensors, the hardware is barely able to run multiple tasks at a time and the question of being able to run multiple VMs, such as in traditional cloud computing, is out of the question. To overcome this problem, in our Sensor Cloud we implement virtual sensors as an image in the software of the corresponding physical sensors. The virtual sensors contain metadata about the physical sensors and the user currently holding that virtual sensor. Additionally, the virtual sensor can have a data processing code, which can be used to process data in response to complex queries from the user. We have implemented virtual sensors in four different configurations: one-to-many, many-to-one, many-to-many, and derived configurations.

3.1 ONE-TO-MANY CONFIGURATIONS

In this configuration, one physical sensor corresponds to many virtual sensors. Although individual users own the virtual image, the underlying physical sensor is shared among all the virtual sensors accessing it. The middleware computes the physical sensor's sampling duration and frequency by taking into account all the users; it reevaluates the duration and frequency when new users join or existing users leave the system. Hence, this system is dynamic. Figure 3.1(a) shows this configuration.

3.2 MANY-TO-ONE CONFIGURATIONS

In this configuration, the geographical area is divided into regions and each region can have one or more physical sensors and sensor networks. When a user requires aggregated data of specific phenomena from a region, all underlying WSNs switch on with the respective phenomena enabled, and the user has access to the aggregated data from these WSNs. The sampling time interval at which all underlying sensors sense is equal to the sampling time interval requested by the user. This configuration can be used to provide fault tolerance if the underlying physical sensors fail. A virtual sensor communicates with a number of underlying physical sensors and it shows the aggregate view of the data to the user. When physical sensors fail, the WSN-facing layer of the Sensor Cloud captures the failure and the virtual sensor communicates it. A working sensor, which provides data within the quality-of-service (QoS) limits, can gather the required data. Thus, the virtual sensor adapts to a change in topology and the WSN-facing layer and is transparent to the user.

3.3 MANY-TO-MANY CONFIGURATIONS

This configuration is a combination of the one-to-many and many-to-one configurations. A physical sensor can correspond to many virtual sensors, and it can also be a part of a network that provides aggregate data for a single virtual sensor (see Figure 3.1(a)).

3.4 DERIVED

A derived configuration refers to a versatile configuration of virtual sensors derived from a combination of multiple physical sensors. This configuration can be seen as a

generalization of the other three configurations, though, the difference lies in the types of physical sensors with which a virtual sensor communicates. While in the derived configuration, the virtual sensor communicates with multiple sensor types; in the other three configurations, the virtual sensor communicates with the same type of physical sensors.

Derived sensors can be used in two ways: first, to virtually sense complex phenomenon and second, to substitute for sensors that aren't physically deployed. Two examples can help us to understand this better.

- Example 1 - Many different kinds of physical sensors can help us to answer complex queries such as, "Are the overall environmental conditions safe in a wildlife habitat?" The virtual sensor can use readings of a number of environmental conditions from the physical sensors to compute a safety level value and answer the query.
- Example 2 - If we want to have a proximity sensor in a certain area where we don't have one mounted on a physical wireless node, the virtual sensor could use data from light sensors and interpolate the readings and the variance in the light intensity to use as a proximity sensor. Figure 3.1(b) shows examples of derived sensors.

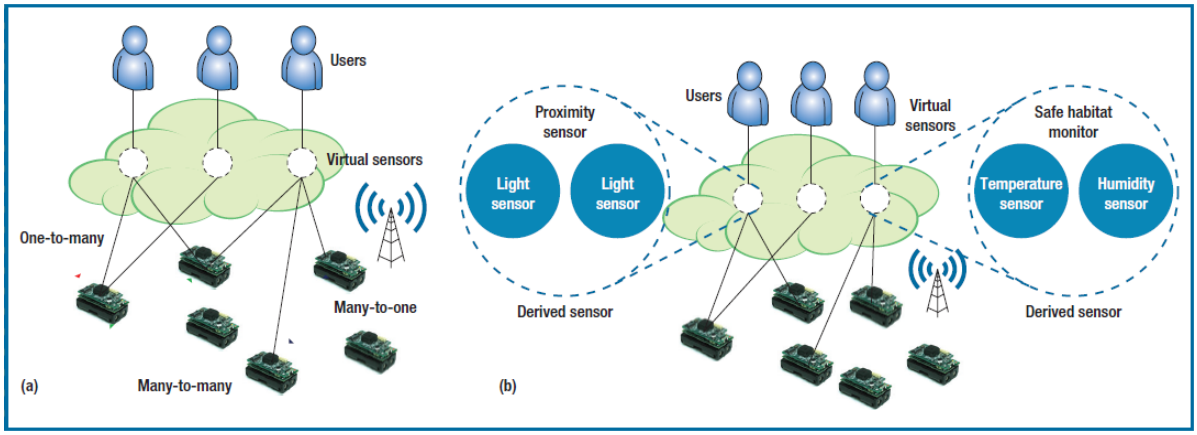


Figure-3.1 Various virtual sensor configurations: (a) one-to-many, many-to-one, and many-to-many, and (b) derived

3.5 APPLICATION

In our running example, the virtual sensor can fall under any of the four configurations. A user might interact with one particular traffic flow sensor to assess traffic condition. Multiple users might also use the same sensor. A user might configure a virtual sensor to provide the average temperature of a region, which may involve multiple sensors. A user might also configure derived virtual sensors to calculate a heat index from temperature and humidity data.

4. MISSOURI S&T SENSOR CLOUD

The Sensor Cloud infrastructure at the Missouri University of Science and Technology (S&T) campus is divided into three prominent layers: client-centric, middleware, and sensor-centric (see Figure 4.1). The client-centric layer connects the users to the Sensor Cloud, whereas the middleware layer performs service negotiation, provisioning and maintenance of virtual sensors, and communication of data from the sensor-centric layer to the client- centric layer. The sensor-centric layer deals with the physical wireless sensors and their maintenance as well as routing of data and commands. From an implementation point of view, we can condense the layered architecture of Figure 4.1 to the block diagram of Figure 4.1. Each block in Figure 4.1 comprises one or more related functionalities in the layered architecture. The block diagram representation in Figure 4.2 is a more implementation-friendly illustration of these functionalities.

4.1 THE CLIENT CENTRIC LAYER

The client-centric layer acts as the gateway between the Sensor Cloud and the user. It's a collection of components that facilitate and manage the interactions between the user and the core of the Sensor Cloud that is, the virtual sensors. The client-centric layer comprises the user interface, session management, membership management, and the user repository components.

The Missouri S&T Sensor Cloud user interface lets users specify parameters such as regions of interest, sensing phenomena, sampling frequency, sensing duration, and mode

(secure or not secure). The Web application parses the request and the parameters and communicates them to the back-end application server.

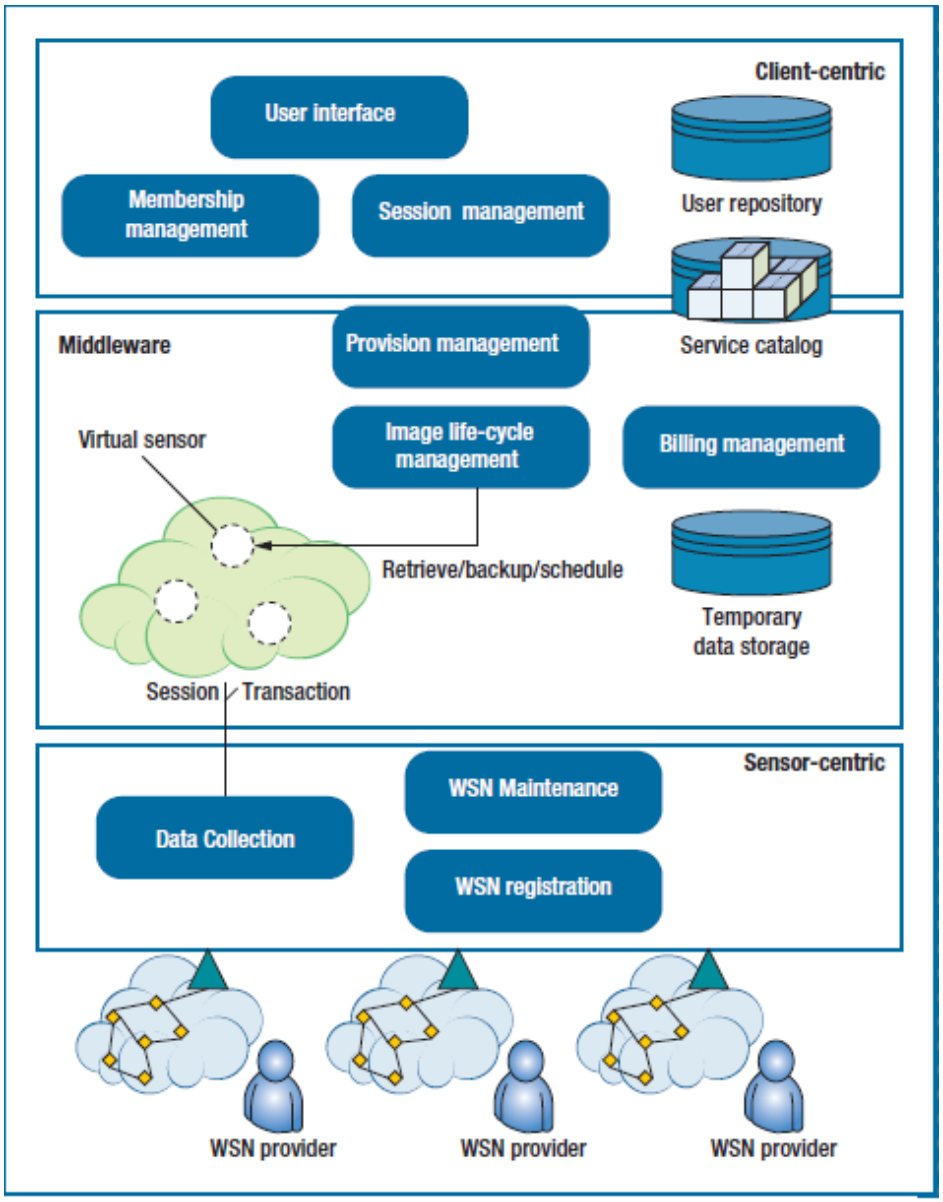


Figure-4.1 A layered Sensor Cloud architecture from the Missouri University of Science & Technology Sensor Cloud. It's divided into three prominent layers: client-centric, middleware, and sensor-centric.

The session management component handles the secure creation, management, and termination of sessions between the middleware and the user. The membership management

component of the client-centric layer takes care of the authorization of users and provision of access to the services for which they are authorized.

The user repository component stores detailed user information in the system such as account credentials, payment history, billing information, and so on. It also keeps track of data sent by WSNs and accessed by the end users.

In our running example, the client- centric layer will expose a GUI showing the locations of the available sensors to the user. The user can create virtual sensors based on regions. The information about the selected region, sampling frequency duration, and QoS agreement would be sent to the middleware.

4.2 THE MIDDLEWARE LAYER

The middleware layer acts as the intermediary between the client-centric and sensor-centric layers and connects the client requests with the data collected from the sensors. The middleware performs a number of functions such as provision management, image life-cycle management, and billing management.

Provision management facilitates the service negotiation between the user and the Sensor Cloud and provisions virtual sensors for each incoming request. This component of the middleware resides in the Web application server block of Figure 4.2. It receives requests from UIs of multiple users with their parameters and modes. If a request can be fulfilled, control is passed onto the virtual sensor server block which triggers the WSNs associated with the selected region and specified parameters via the back-end application server. Once an

agreement between the user's requirement and the Sensor Cloud's capabilities has been reached, the virtual sensors are created.

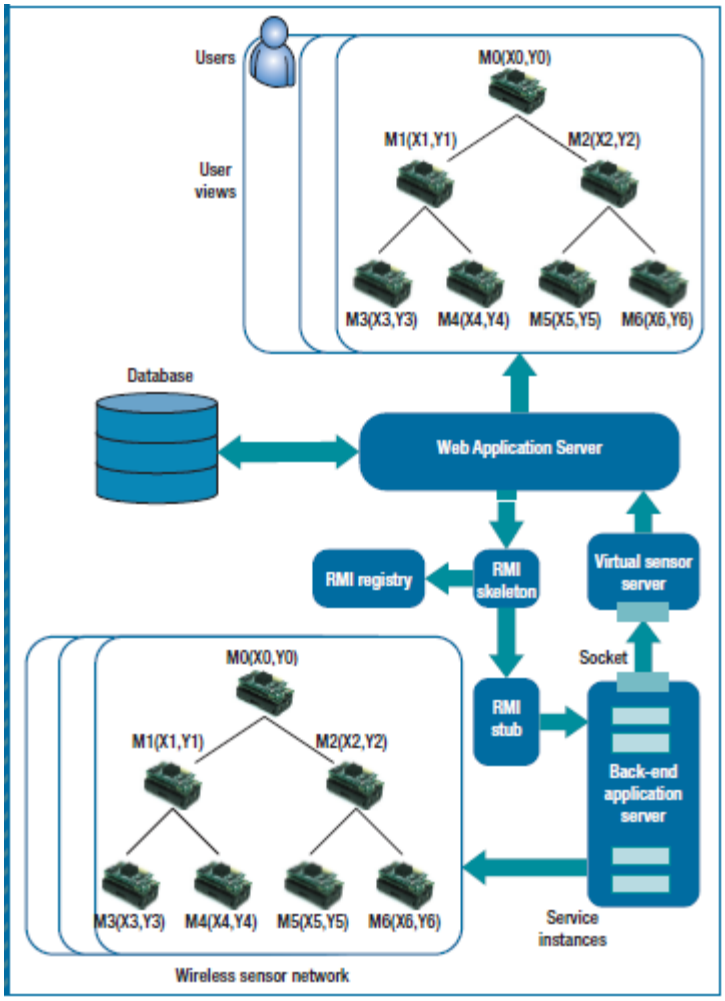


Figure-4.2 A block diagram representation of the architecture from Figure 4.1. Each block comprises one or more related functionalities from the layered architecture.

The image life-cycle management component is implemented in the virtual sensor server block of Figure 4.2. The virtual sensor server receives requests for instances from the provision management component and takes care of creating instances for the virtual sensors provisioned for the users. After creating the instances, it communicates with the back-end application server to request a corresponding service instance. Each service instance of the

back-end application is registered in the remote method invocation (RMI) registry before any request for that region can be satisfied. As shown in the block diagram in Figure 4.2, with respect to the user's selection of region, the associated service instance of the backend applications and their registered names are identified by querying the database. Later, these RMI-registered names are looked up from the registry to connect a particular user's virtual sensors to the respective service instance. The virtual sensor server mediates the communication between the Web application and the back-end application and ultimately displays a different set of sensor data to each end user connected to the Sensor Cloud. The billing management component of the Sensor Cloud is responsible for keeping track of each user session with respect to the types of sensors used, number of sensor used, and mode of sensor usage to generate the invoice based on previous service agreements.

In our example, the middleware layer receives information about the user session from the user-centric layer and creates the virtual sensor. The virtual sensor configuration is decided on the basis of the type of data the user wants, the user's region of interest, and the agreed upon QoS. If multiple users request information from the same sensors (for example, traffic information from the same location), the middleware consolidates the requests.

4.3 THE SENSOR CENTRIC LAYER

The sensor-centric layer directly communicates with the physical sensors using the WSN registration, WSN maintenance, and data collection components. When network owners want to provide service through the Sensor Cloud interface, they need to register their WSNs. The Sensor Clouds verify the physical sensors and their capabilities, which should also provide location information. At this point in time, we aren't considering mobile sensors,

therefore pre-deployed location information in the form of longitude, latitude, region IDs, and cluster IDs should suffice. The information collected in the WSN registration phase is used in cataloging the information about physical and later virtual sensors.

Once a WSN is registered, the network owner is in charge of keeping the physical sensors in good health. The registration binds the WSN owner and the Sensor Cloud in a trust relationship where the WSN owner is expected to provide accurate, untampered sensor readings, and the Sensor Cloud is expected to correctly provide compensation for the received sensor readings. The trust between the two parties can be enhanced by using secure and trusted data collection and aggregation techniques on the WSN side and by performing data anonymization and doing computations on encrypted data on the cloud side. (A more detailed discussion of the security, privacy, and trust issues in Sensor Cloud can be found elsewhere [5].)

The WSN maintenance component provides interoperability of the heterogeneous mote platforms, periodically checks the health of each mote in the Sensor Cloud, provides synchronization between sensors, and collects metadata information about the motes and the networks. To handle non-interoperability issues, we installed a GumStix computer on module (www.gumstix.com) at the junction of two or more incompatible WSNs. The GumStix is hooked up with a number of different motes at different ports. The GumStix collects data from one port and transmits it to other ports as needed.

To check the health of the motes and collect metadata information, the WSN maintenance component periodically pings each network. The motes in the network reply with information such as their battery level, the mote to which they were last connected, their location, their region, and so on. Although we haven't yet tackled the issue of synchronization

between the networks and sensors, in the future, global time information can be piggybacked on the ping packets. This would provide time synchronization between the networks and the sensors.

To provide a finer-grained synchronization, we can also use the powerful GumStix nodes. The data collection component of the sensor-centric layer connects the system directly to the wireless sensor networks. Each WSN is connected to the data collection component through a base station. The data collection component, which resides in the back-end application server, runs one service instance for each base station. The service instance opens two dedicated ports, one to communicate with the base station and one to communicate with the associated virtual sensor on the virtual sensor server.

The sensor-centric layer in our example receives a query packet from the middleware and replies with the requested data. This layer handles the routing protocol, fault tolerance, and other network-related issues.

5. SOFTWARE DESIGN

The Sensor Cloud is multi-tiered client server software architecture with each layer logically separated from the other. The sensor-centric layer is the data tier. The layer consists of physical wireless sensors that generate real-time data. The middleware is the application or the logic tier, which controls the data collection. The client-centric layer represents the presentation tier.

From a software developer's point of view, there are two different facets to developing a Sensor Cloud: the system side and the sensor side. The system side consists of the client-centric and middleware layer (the presentation and application tiers) and is basically used to manage physical resources. The end user's view will vary depending on the application. The heart of any Sensor Cloud application, is going to be the middleware layer. The middleware layer is expected to be flexible enough to handle issues when physical and virtual sensors are scaled up and down. It's also expected to aggregate the user requirements and redirect data according to these requirements from the sensors to the users, in addition to handling its regular tasks.

The second facet of a Sensor Cloud is the physical sensors side, which consists of the sensor-centric layer. WSNs are distributed networks, where a number of physically separate entities work together toward a goal (in this case, generating data according to the user's requirements). WSNs face the same general issues as distributed systems such as synchronization, fault tolerance, and security. While developing for WSNs, developers need to keep these issues in mind, in addition to the wireless sensor-specific constraints such as low bandwidth, low processing power, and finite energy sources.

6. QOS IN SENSOR CLOUD

A Sensor Cloud manages QoS at two levels: at the sensor-centric layer and at the virtual sensors. The sensor- centric layer handles network-related issues such as responding to node failures, network partitioning, and packet losses. The virtual sensor layer then works on top of the sensor-centric layer's services to manage QoS parameters such as reliability, data accuracy, and coverage on top of the network layer. As we explained earlier, the many-to-one virtual sensor configuration can be used to provide data reliability via aggregation. The accuracy of data in such cases depends on the spatial and temporal correlation between the nearby sensors. The virtual sensor layer makes use of the correlations to retrieve data within the QoS limits from nearby sensors. The virtual sensor layer can also switch between configurations, for example, from one-to-many to many-to-many to provide data to multiple users within the specified QoS limits.

7. IMPLEMENTATION

We used the Linux platform for multiple back-end servers, which cater to multiple regions. We programmed the Web application and back-end server application in Java. We used RMI along with socket communication for communication among the various back-end servers. We used TelosB motes as wireless sensors, each equipped with a humidity, temperature, light intensity, and infrared sensor. We programmed the sensors using TinyOS 2.2 (http://tinyos.stanford.edu/tinyos-wiki/index.php/Installing_TinyOS).

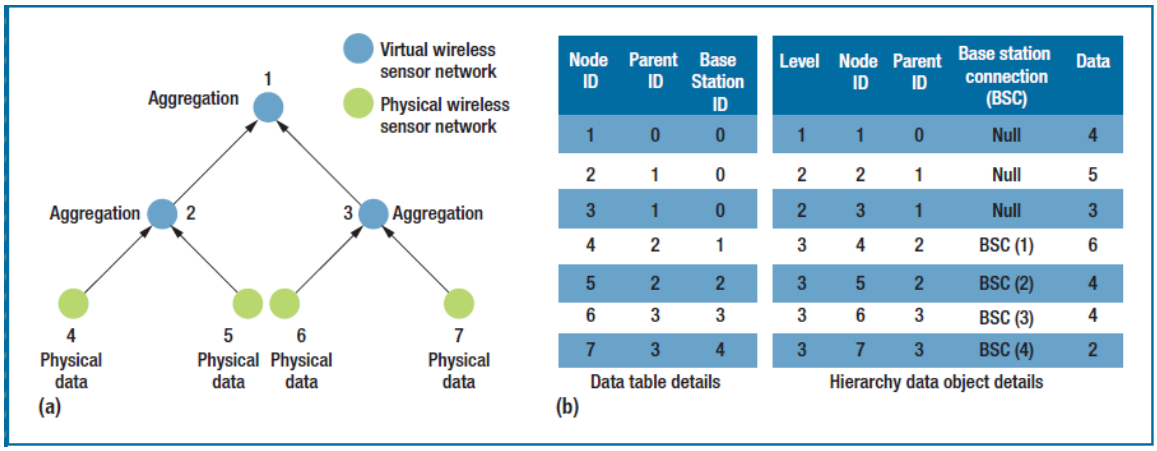
7.1 DATA STREAMING FOR MULTI-USER ENVIRONMENT

The virtual sensor model can effectively support a multiuser environment. A single wireless sensor provides data for multiple users, where the users can request data at varying frequencies and of different phenomena. When the Web application server in Figure 4.2 receives user requests, they are transferred to the virtual sensor server. The virtual sensor server performs the mapping between the user's virtual and physical sensors. If multiple virtual sensors correspond to one physical sensor, the virtual sensor server combines the request by combining the sampling duration, sampling frequency, and sensing phenomena. The combined request is then forwarded to the appropriate service instance of the back-end server. The service instance communicates with the WSN and collects the data. Data from each WSN is sampled at the minimum frequency of all requests. This data is time stamped with the local time and stored in the database and displayed to the user at the requested frequency. Users can also select data from multiple base stations at the same time, either by selecting locations that includes multiple WSNs or by selecting multiple locations. The data

will then be aggregated periodically at the requested frequency on the basis of the selected locations' hierarchy.

7.2 VIRTUAL SENSOR IMPLEMENTATION

In this section, we describe the usage of many-to-one virtual sensors; however, one-to-many, many-to-many, and derived sensors are also implemented in a similar fashion. The geographical area that the Sensor Cloud covers is divided into regions that are arranged hierarchically. Aggregated data from various networks in a region are based on the user's selection of the region and the selected region's hierarchy. The scheme can be understood well by visualizing a network of WSNs (see Figure 7.1(a)). The topology in Figure 7.1(a) shows the hierarchy of WSNs, where each intermediate node and the root node is a network in itself and can have any number of children. In this example, we assume that an end user needs data from region 1, which is a virtual node. Similarly, all intermediate nodes (2 and 3) and the root node in the hierarchy are virtual nodes. On the other hand, all leaf-level nodes (4, 5, 6, and 7) represent physical WSNs. Thus, we can call this topology a network of VSNs (virtual sensor networks) and WSNs. The hierarchy information stored in data table (see Figure 7.1(b)) is static and is required while aggregating the information at intermediate and root-level nodes. On the other hand, a hierarchy data object is created for each user request. Once a many-to-one mapping of the virtual sensor is obtained, the virtual sensor server sends a request to the back-end application server, which then forwards the request to the concerned WSNs. Once all WSNs providing data to selected VSNs in the hierarchy are switched on, the backend application server starts relaying data, which is parsed according to the VSN hierarchy.



Node ID	Parent ID	Base Station ID
1	0	0
2	1	0
3	1	0
4	2	1
5	2	2
6	3	3
7	3	4

Data table details

Level	Node ID	Parent ID	Base station connection (BSC)	Data
1	1	0	Null	4
2	2	1	Null	5
2	3	1	Null	3
3	4	2	BSC (1)	6
3	5	2	BSC (2)	4
3	6	3	BSC (3)	4
3	7	3	BSC (4)	2

Hierarchy data object details

(b)

Figure-7.1 Virtual sensor examples: (a) hierarchy of a user's region of interest and (b) hierarchy data object tables for virtual sensors.

8. CONCLUSION AND FUTURE WORK

A major hindrance to the widespread adoption of wireless sensor networks is the difficulty in deploying and managing the networks. These tedious tasks take the focus away from the usage of WSNs to managing the WSNs. Sensor clouds aim to take the burden of deploying and managing the network away from the user by acting as a mediator between the user and the sensor networks and providing sensing as a service. In this paper we have described the Missouri S&T Sensor Cloud. We discussed the layered architecture of the Sensor Cloud and our implementation of it in Java and TinyOS. The Missouri S&T Sensor Cloud allows sensor networks spread in wide geographical areas to be connected and used as a single entity. It further allows the usage of sensor networks by multiple users at the same time by virtualizing the resources in software. Virtualization also helps in creating what we term as derived sensors from heterogeneous data streams. While we have presented a working model of the Sensor Cloud, our future work would include working towards a complete plug and play model of application deployment and energy efficient scheduling of wireless sensors.

REFERENCES

[1] M.M. Hassan, B. Song, and E.-N. Huh, "A Framework of Sensor-Cloud Integration Opportunities and Challenges," Proc. 3rd Int'l Conf. Ubiquitous Information Management and Communication (ICUIMC 09), ACM, 2009, pp. 618–626.

[2] Kortuem, G., Kawsar, F., Fitton, D., & Sundramoorthy, V., "Smart Objects as Building Blocks for the Internet of Things," IEEE Internet Computing, vol. 14, no. 1, 2010, pp. 44–51.

[3] M. Yuriyama and T. Kushida, "Sensor-Cloud Infrastructure—Physical Sensor Management with Virtualized Sensors on Cloud Computing," Proc. 13th Int'l Conf. Network-Based Information Systems (NBIS 10), IEEE CS, 2010; doi:10.1109/NBiS.2010.32.

[4] Poolsappasit, N., Kumar, V., Madria, S., & Chellappan, S., "Challenges in Secure Sensor-Cloud Computing," Secure Data Management, LNCS 6933, Springer, 2011, pp. 70–84.

[5] S. Kabadayi, A. Pridgen, and C. Julien, "Virtual Sensors: Abstracting Data from Physical Sensors," Proc. Int'l Symp. World of Wireless, Mobile and Multimedia Networks (WoWMoM 06), IEEE CS, 2006; doi:10.1109/WOWMOM.2006.115.

[6] N.K. Kapoor, S. Majumdar, and B. Nandy, "Scheduling on Wireless Sensor Networks Hosting Multiple Applications," Proc. IEEE Int'l Conf. Communications (ICC 11), IEEE, 2011; doi:10.1109/icc.2011.5963195.

[7] K. Aberer, M. Hauswirth, and A. Salehi, "A Middleware for Fast and Flexible Sensor Network Deployment," Proc. 32nd Int'l Conf. Very Large Databases, VLDB Endowment, 2006, pp. 1199–1202.

[8] Industrial Wireless Sensor Networks (IWSN) Market: Global Forecast & Analysis (2012–2017), tech. report SE 1662, marketsandmarkets.com, June 2012; www.marketsandmarkets.com/Market -Reports/wireless-sensor-networks -market-445.html.

II. ENERGY EFFICIENT SCHEDULING AND ALLOCATION OF TASKS IN SENSOR CLOUD

Rashmi Dalvi and Sanjay Kumar Madria

Department of Computer Science, Missouri University of Science and Technology,

Rolla, MO

ABSTRACT

Wireless Sensor Networks (WSNs) are frequently used for a number of reasons like unattended environmental monitoring. WSNs also have low battery power hence various schemes have been proposed to reduce energy consumption during the processing. Consider a Sensor Cloud in which owners of heterogeneous WSNs come together to offer sensing as a service to users of multiple applications. In Sensor Cloud environment, it is important to manage the requests from multiple applications efficiently. In present work, we have proposed a scheduling and allocation schemes suitable for the multiple applications in a Sensor Cloud. This scheduling scheme proposed is based on TDMA which considers the fine granularity of tasks. The allocation scheme utilizes Voronoi diagram to reduce the number of sensors involved to increase the WSN's short life. It does so by using spatial correlation between the sensing ranges of sensors in WSNs. In our performance evaluation, we show that our proposed scheme saves energy of sensors and provides substantial coverage of the sensing region for much longer duration in comparison to other works.

1. INTRODUCTION

Wireless sensor networks (WSNs) are quite popular because they can be used in a wide range of applications, easy to deploy, withstand adverse conditions, work in unmonitored networks, and provide dynamic data access. Unfortunately, WSNs have limited amount of battery power and thus, they have shorter lifespan. The sensors must be efficiently allocated and scheduled for all tasks so that their lifespan increases, thus improving the battery's longevity. We consider a Sensor Cloud of heterogeneous WSNs as described in our work [12]. The WSN owners within a Sensor Cloud collaborate with one another to provide sensing as a service and thereby gain profit from underutilized WSNs. The users of the applications employ sensing service, to select either single or multiple WSNs. Multiple users and applications are served by the Sensor Cloud at the same time. Hence we need scheduling and allocation scheme which is suitable for Sensor Cloud.

Pantazis et al. [2] proposed TDMA based scheduling scheme that balances power saving and end-to-end delay in WSNs. Scheme schedules the wakeup intervals such that data packets are delayed by only one sleep interval from sensor to gateway. While scheduling the sensors however they did not consider a multi-application environment. In a Sensor Cloud, users of multiple applications will want to consume the data differently which brings forth the need of categorizing user tasks. The scheme proposed in [2] fails to account for various types of tasks and thus, it cannot achieve the goal of energy conservation for different types of requests. Similar to push requests in [1], a number of users may like to consume data from WSNs at a specific frequency. Other users prefer to do so once and in ad-hoc fashion like push requests in [1]. Some users may like to be notified when a specific event occurs. In all

such needs, in-spite of being energy efficient [2] will not conserve enough energy because they did not consider fine granularity of the tasks received from applications and application users.

Viera et al. [4] proposed the allocation scheme for WSNs using Voronoi diagram. When many wireless sensors are close to each other, their algorithm turns off some to increase the overall lifetime of WSNs. They used distributed strategy to turn off the sensors, therefore sensors have to gossip with other sensors that causes communication overhead. Energy spent in communicating with other nodes leads to wastage. Additionally, their approach is suitable for densely deployed WSNs only. In a Sensor Cloud, WSN owners employing different deployment strategies (dense deployment, sparse deployment, combination of dense and sparse deployment) may want to participate. So approach proposed in [4] will not be applicable in Sensor Cloud.

In this paper, we have extended the scheduling scheme proposed in [2] by considering fine granularity of the tasks. In multi-application environment, tasks are treated differently depending on their type. This increases the user satisfaction and by reducing the response time of the tasks and improving throughput greatly. It also provides the optimal energy conservation. In allocation scheme presented here, in order to save battery power, centralized strategy is used to take the decision of turning on required nodes. Moreover, the WSNs within the scheme in this work can have densely and/ or sparsely deployed regions at the same time. In dense zones, less number of the sensors need not be set at an on state because sensors in the vicinity tend to sense the same data. In sparsely deployed zones, the most number of sensors are expected to be allocated to the task. It is important to turn only the required sensors to an on state in a WSN that contains both densely and sparsely deployed regions. Thus

allocation method in this work takes into account all these types of deployments, for better results.

The major contributions of this work also include, experimentally showing that our scheduling scheme provides better throughput and response time than the base case we developed for the Sensor Cloud. It treats each task differently which leads to optimal energy conservation. In allocation scheme, we prove that the centralized approach proposed in the paper selects lesser number of nodes than in [4], and eliminates gossiping between the motes. With experiment we show that, it provides 95% coverage of the sensing area. We also show that, in comparison with [4], percentage improvement in number of nodes compared to, the percentage compromise of uncovered area is very less.

2. RELATED WORK

Kapoor et al. [7], addressed the issue of allocating and scheduling sensors in a multi-application environment. This work eventually led to energy conservation in WSNs. The assumption behind the proposed scheme is that each sensor can be utilized by a single application. This brings forth the need of allocation and scheduling algorithms in Sensor Cloud. Various parameters (e.g. hop distance, energy requirements, and potential energy) were considered when the algorithms were being designed. These algorithms were classified as either Knowledge-Free or Knowledge-Based. An allocation algorithm was used to reduce energy consumption. A scheduling algorithm was used to reduce the sensor's response time. This [7], proposed approach is not suitable in a Sensor Cloud in which many users of different applications might be interested in obtaining data from the same geographical region.

In another work regarding scheduling of WSNs, Xiong et al. [3] proposed, a multi task scheduling technique for low-duty-cycled WSNs. They concentrated on load balancing problem for multiple tasks among sensor nodes in both spatial and temporal dimensions. They, however, did not consider the granularity of the tasks running in a Sensor Cloud.

Pizzocaro et al. [5] proposed the use of a WSN allocation scheme in a multi WSN environment. Their scheme is based on bidding (conducted by sensors within the network for pre-empting sensors). Bidding creates extensive intra-communication within a WSN, consuming extra energy. In order to conserve the energy, we present an allocation scheme executed completely by Base Station (BS), bringing the intra-communication down effectively. Moreover, [5] pre-empts the sensors completely for a specific task. As a result,

when a many users are interested in the same set of sensors at the same time, they cannot be served by their scheme.

Vuran et al. [6] used a Voronoi diagram to eliminate nodes that share a common sensing space that is greater than the threshold. Their algorithm works to turn the nodes off according to the distortion that exists between the transmission and reception of packets. This scheme is not suitable to a Sensor Cloud scenario because the selection of nodes does not change according to requests received from multiple applications. For example, only sensors selected to be in an active state serve a specific region until their energy is completely depleted. Aziz et al. [10], proposed an allocation scheme that uses PSO (Particle Swarm Optimization), with a Voronoi diagram, to select nodes from an ROI (Region of Interest). The PSO is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Their solution is not suitable in a Sensor Cloud scenario in which WSNs are expected to have both densely and sparsely deployed regions. Alsalih et al. [11], used a Voronoi diagram to explain an allocation scheme. Sensors in the network maintain the adjacency list which is used while communicating with neighboring sensors about their position. It finds the partitions in the WSN and selects the minimum number of sensors in the region. This method, however involves a higher communication overhead. It also requires additional storage space at each sensor to maintain an adjacency list.

Andrei et al. [8] proposed the approach of scheduling tasks. They suggested that the Minimum cut theorem be used to partition WSN into zones, reducing the number of transmission and receptions. Thus, each convergent transmission (from any node to the BS) will have a minimum number of hops. Such a strategy may, however, lead to communication

overhead in the process of partitioning and gossiping between the sensors in WSNs. Additionally, the minimum cut algorithm runs only once at the network initialization. Therefore, this scheme is not suitable for dynamic networks. Xiong et al. [3], proposed a scheduling scheme that uses a load balancing approach to solve problems in low duty cycled WSNs. Although this scheme is dynamic, did not account for fine granularity of the tasks. They also failed to address how the scheme would work if the length of each task was unequal to the next. Cao et al. [9] also proposed scheduling in low duty cycled WSNs. Here, multiple paths are used to transmit the data from the sensor to the BS when data transmission fails to increase the reliability. Although this scheme provides a greater fault tolerance, it requires an extensive amount of energy.

3. SENSOR CLOUD

3.1 ARCHITECTURE

The Sensor Cloud's architecture [12] is illustrated in Figure 3.1. This architecture is broadly divided into three layers: Client centric, Middleware, and Sensor centric.

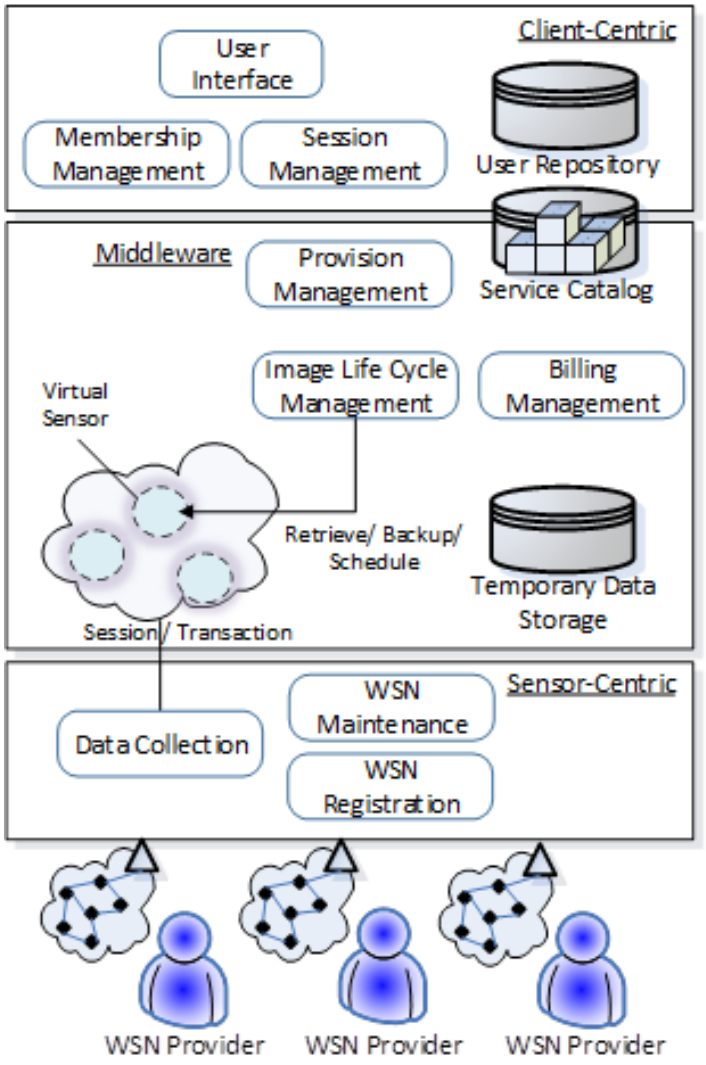


Figure-3.1 Sensor Cloud Architecture

As the name suggests, a Client centric layer connects end users to the Sensor Cloud; it manages a user's membership, session, and GUI. Middleware is the heart of the Sensor Cloud. It uses virtualization, with help from various components (eg. provision management, image life-cycle management, and billing management) to connect all participating WSNs to the end user. Sensor centric layer connects the Middleware to the physical WSNs. It also registers participating WSNs, maintains participating WSNs, and collects data.

3.2 CHALLENGES WITHIN A MULTI-APPLICATION ENVIRONMENT

A Sensor Cloud is comprised of heterogeneous WSNs. Thus various types of WSNs may participate in the cloud. For example, WSNs in a Sensor Cloud can be deployed sparsely and/or densely. Deployment transparency provided to the Middleware will lead to selection of minimum number of sensors. This will improve life of the network. Hence, to select minimum number of sensors, an efficient allocation scheme is needed.

Conventional schemes that are typically used to schedule and allocate WSNs will not work efficiently in multi-application environment, because those schemes are designed for single application environment. The scheme proposed here focuses on scheduling and allocating sensors in multi-application environment to increase user satisfaction.

3.3 ASSUMPTIONS

The assumption is that, in a Sensor centric layer, the base station will know where the wireless sensors are located. Knowledge of sensor's physical locations, will aid base station

to allocate sensors to a given task. This will eliminate the communication overhead required by the sensors in allocation process.

In WSNs, tree topology and cluster topology are widely used over other network topologies. Because of its structure, tree topology provides wider area coverage than cluster topology. Thus, in this work a tree topology is assumed for all WSNs that participate in a Sensor Cloud.

4. SCHEDULING TASKS

The sensing requests were broadly classified into three categories. In multi-application environment, Applications that consume Sensor Cloud services are expected to receive requests from one or more of the categories specified in section 4.5.1. These requests are also known as Tasks.

4.1 TYPES OF TASKS

The types of tasks are classified as Task T1, Task T2, and Task T3.

4.1.1 Task T1. Task Type 1 (referred to as T1) is a task that is requested by users when they need sensor data at a specific frequency. These tasks may include a request for information on a specific topic (eg. weather broadcast). This information is sampled periodically and then broadcast back to the user. These requests have also been referred to as Push requests [1]. When this type of request is made, the BS sends data to the sensors only once at the beginning of the request. The sensors sense and then send the data to the BS at a given sampling frequency and time duration. Task T1 becomes the most expensive of all tasks in a WSN that is running at a maximum sampling frequency because this task pre-empts other tasks for the same WSN. The cost of this task is high at a higher frequency. It decreases as the sampling frequency decreases.

In a WSN, consider a scenario when *n* users have requested for *n* tasks of type T1. These tasks are requested for same set of sensors and for same sensing phenomenon, but for different data frequencies. In this case the physical data frequency of WSN will be the minimum data frequency of all *n* tasks. Remaining *n-1* frequencies will be virtual data

frequencies. For *n-1* tasks, selection of minimum frequency will lead to some delay in data received from sensors. However, this approach will conserve the sensor's energy because, $minimum\ frequency = \min(f1, f2 \dots fn)$. In another approach, LCM (Least Common Multiple) of all *n* frequencies can be set as the minimum frequency. LCM approach will reduce delay in response, but will significantly increase sensor's energy because, $minimum\ frequency \leq \min(f1, f2 \dots fn)$.

We prioritized saving sensor's energy over latency in receiving data. Therefore, the minimum frequency of all requests was used in this study. Users in need of data (based on type T1) need to send information (e.g., location, sensing phenomenon, sampling frequency, and sampling duration) to the Sensor Cloud.

4.1.2 Task T2. Type 2 tasks (referred to as T2) are requested by users who need one time data on the fly. This type of task is also known as a Pull Task [1], is designed to serve ad-hoc requests. BS will send request details (e.g. location, sensing phenomenon) to the sensors, and sensors will respond with the latest data.

4.1.3 Task T3. Task Type T3 (referred to as T3) is used for event-based requests. During this type of request, the BS sets the event on sensors according to the requested condition on the sensor data. These applications will need several inputs, including location, sensing phenomena, event condition, monitoring frequency, and monitoring duration. Sensors continue sensing the data at a requested frequency and respond when the event condition occurs. Task T3 is useful for a number of applications, including fire detection, intrusion detection, and so forth. The algorithm used for T3 is a trade-off between the event occurrence frequency and the cost of event detection. If the event occurs very frequently, then a large number of duty cycles must be assigned to T3, increasing the effective cost.

T3 task can be further classified into ‘Notify once’ and ‘Notify until the condition is false’.

The behavior of ‘Notify once’ would be similar to Task T2. In case of ‘T3 with Notify Once’ sensors send data to the BS only once when event condition is met. On the contrary, for T2 tasks, on request, sensors respond with data.

Behavior of ‘Notify until the condition is false’ would be similar to Task T1. When event occurs, sensors executing task ‘T3 with Notify until condition is false’ send data to BS until condition turns false. However for task T1, sensors send data to the BS at given frequency. As T3 does not require data to be always sent from sensors to the BS, cost of T3 is lesser than T1.

4.2 HANDLING REDUNDANT REQUESTS

The handling of redundant requests benefits application users, WSN owners, and cloud service providers within a Sensor Cloud. The BS handles the redundant requests without affecting the physical frequency. For an instance, assume that a node ‘A’ is already serving request 'R1' for task T1 at frequency ‘f’. The following redundant requests to node 'A' can be served by the BS from the data received for request 'R1'.

1) Other T1 requests for node 'A', with a frequency in multiples of ‘f’, until the duration of R1 ends
2) T2 requests for node 'A'. The maximum delay is = ‘f’
3) T3 requests for node 'A'. The maximum delay is = ‘f’ until the duration of R1 ends

Similarly BS will serve requests for other nodes according to requests already made on the WSN.

4.3 SCHEDULING SCHEME

Scheduling scheme proposed here is divided into Sensor Scheduling and Task Scheduling.

4.3.1 Sensor Scheduling. The scheduling scheme proposed in [2] is extended here to avoid the problem of packet collision during transmission and reception. Both wake up and sleep modes are used for all types of tasks. For tasks T1 and T2, the BS determines in what order the sensors must transmit and receive data. Thus path wakeup is not required for tasks T1 and T2. In task T3, however, the event triggers the transmission of data from node(s) to the BS. The schedule for T3 is not defined beforehand. On the occurrence of event, data is pushed into the vacant duty cycle. Hence, T3 needs the path wakeup strategy proposed in [2] to transmit the data from the node(s) to BS.

Consider the tree topology given in Figure 4.1. Messages between sensors are divided into following categories according to the direction of packet transmission. They are further classified based on the level of granularity:

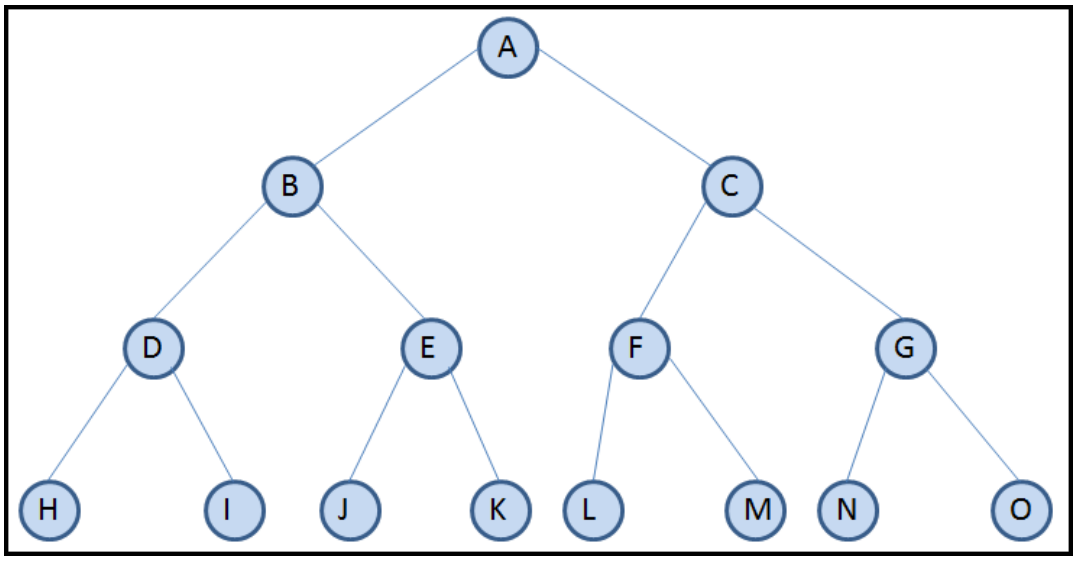


Figure-4.1 WSN in a Tree Topology

4.3.1.1 BS to node(s) command messages. This type of message has both wakeup and command packets [2]. Time is divided (see Figure 4.2) to avoid a collision between packets. Wakeup messages have a short duration; Command messages have a longer duration. The command packet's time period is determined by the command's maximum length of the command. These messages are transmitted from the BS to the node(s) for every new request. They are not, however, transmitted for redundant requests.

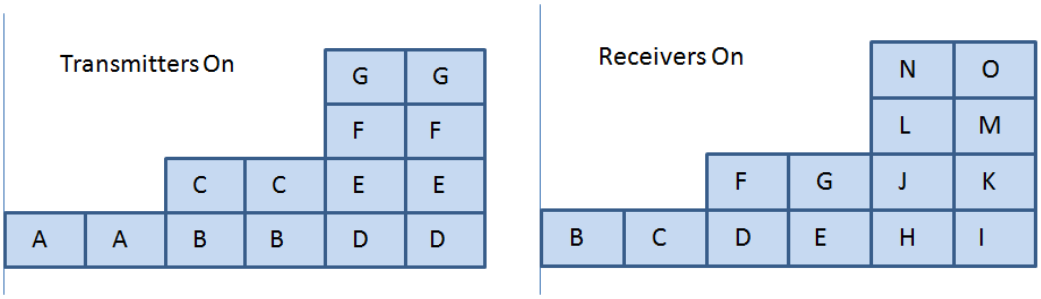


Figure-4.2 BS to Node(s) messages

Node 'A' sends data to nodes 'B' and 'C'. Transmitter of 'A' is turned on for the first two slots. Receivers of 'B' and 'C' are turned on in the first and second timeslots, respectively. Next, nodes 'B' and 'C', in parallel, send data to 'D' and 'F', followed by 'E' and 'G'. This procedure continues until the algorithm reaches the leaf nodes. This sequential transmission of packets helps in avoiding collision between packets.

4.3.1.2 Node(s) to BS data messages. This type of messages has both wakeup and data packets [2]. The, data packets in task T1 are pushed to the BS at a scheduled frequency. The data for tasks T2 and T3 is pushed at the next available duty cycle. Data transmission occurs (see Figure 4.3). Nodes 'J', 'K', 'H', and 'I' are leaf level nodes that need to send data to the BS. The 'J' and 'H' leaf nodes send data to the 'E' and 'D' leaf nodes, respectively. Later, nodes 'K', 'I' send data to 'E' and 'D', in order. Accordingly, the transmitters that are sending data are turned on, and receivers receiving data are turned on for a specified amount of time.

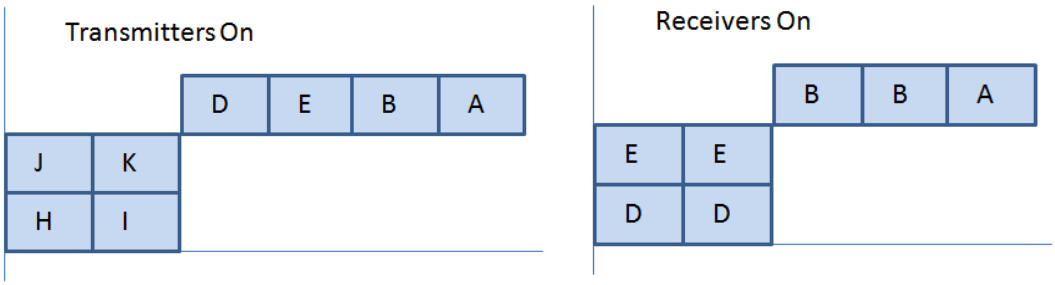


Figure-4.3 Node(s) to BS messages

4.3.2 Task Scheduling. The scheme for task scheduling is an extension of the sensor scheduling scheme explained in Section 4.3.1. Two types of cycles are given in Figure 4.4.

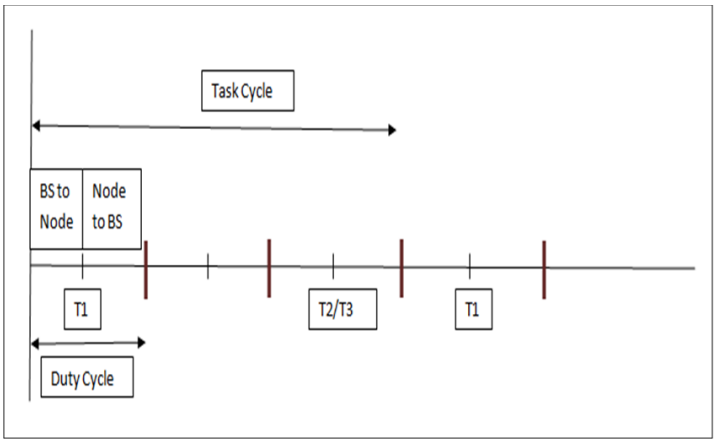


Figure-4.4 Duty Cycle and Task Cycle

4.3.2.1 Duty cycle. The duty cycle includes time slots for the transmission of data from BS to all nodes and from the node to the BS. For a particular WSN, the time duration of a duty cycle is equal to the time duration of the duty cycle that contains the longest task. Any task (T1/ T2/T3) can be assigned to a duty cycle according to whether or not it can serve the task. The duty cycle's length is selected as the length of the largest task in the requests currently served. Each slot in Figure 4.4 separated by a brown color line is a duty cycle. Duty cycle T1 is further divided into two messages: BS to Node(s) and Node(s) to BS. Duty cycles are used for tasks T2 and T3 as per allocated tasks.

4.3.2.2 Task cycle. A task cycle is a combination of multiple consecutive duty cycles that may contain tasks T1, T2, and T3. The task cycle's length changes when the physical operating frequency of the WSN changes. Essentially, task cycle depends on the T1 frequency.

4.4 DUTY CYCLES IN DETAIL

A duty cycle's length is variable. The minimum length of all requested tasks is considered to be the length of a duty cycle for a given WSN. When tasks are short in length, this strategy reduces the delay in response. Correlation between tasks and messages in duty cycle is explained further.

4.4.1 Duty Cycle For Task T1. Duty cycles for T1 are divided into two types of messages, BS to Node(s) messages and Node(s) to BS messages.

4.4.1.1 BS to node(s) message. The BS to Node(s) message is transmitted on a T1 request to the WSN. For the first T1 duty cycle, time is allocated on BS to Node(s) message. Transmission will start from BS and propagate to the leaf level nodes.

4.4.1.2 Node(s) to BS message. A Node(s) to BS message is transmitted at each occurrence of a sampling frequency. Time slots are assigned by beginning at the leaf level node and moving to the BS. These messages are driven by both, as the BS and the Node(s) are each aware of the transmission time slots.

4.4.2 Duty Cycle For Task T2. Duty cycles for T2 are divided into two types of messages, BS to Node(s) messages and Node(s) to BS messages.

4.4.2.1 BS to node(s) message. Task T2 are designed to serve ad-hoc requests. Thus, these messages are transmitted when nodes find an empty duty cycle.

4.4.2.2 Node(s) to BS message. If a request is already received by a node, then the response is sent in next available duty cycle. Node(s) to BS slot of the duty cycle is selected to transfer the message. Unlike T1 Node(s) to BS messages, these messages are driven completely by the BS, because the BS tells the node(s) when to transmit the data.

4.4.3 Duty Cycle For Task T3. Duty cycles for T3 are divided into two types of messages, BS to Node(s) messages and Node(s) to BS messages.

4.4.3.1 BS to node(s) message. These messages can be transmitted in next available duty cycle. The event is set to the given node(s) once a request is sent to the node(s) with monitoring frequency and condition.

4.4.3.2 Node(s) to BS message. This message transmission uses path wakeups from node to the BS. Wake up messages are sent at the beginning of this message to all nodes in the path. Data messages are sent once path to the BS is established. These messages are completely driven by the monitoring nodes because the BS is unaware of the data received.

4.5 PREEMPTION CONDITION

The preemption of nodes in a WSN depends on two factors: the availability of duty cycles at the root level node and the frequency of requests in T1 mode. The minimum frequency of T1 limits a network by restricting other nodes from transmitting the data.

$$\frac{1}{\sum_{P=1}^{existingT1Freq}\left(\frac{1}{Freqp}\right)} \geq D \qquad (1)$$

Equation (1) defines the pre-emption condition by rejecting requests that do not satisfy the above condition. Here, ‘existingT1Freq’ is the list of T1 frequencies operating on a WSN, Freqp is the p_{th} operating frequency (of non-redundant requests), and D is the length of the duty cycle.

The, Algorithm 1 ensures that at least one duty cycle is available in the task cycle for T2 and T3 requests. This algorithm, however accepts T1 and T3 requests only when the value

at L.H.S. in (1) is greater than the value of R.H.S. The algorithm rejects the request when the condition is not satisfied. Rejected requests inform users that the WSN does not currently have the capacity to serve the given request.

4.6 SCHEDULING ALGORITHM

Scheduling Algorithm contains Algorithm 1, BS Scheduling that executes on BS, and Algorithm 2, Sensor Scheduling that executes on the sensors.

In Algorithm 1, *R* is the input request, $task$ is the type of task from set {T1, T2, T3}, $phenom$ is the phenomena to be sensed, $sens$ is the set of sensors chosen to both sense and forward data, fwd is the set of sensors chosen to forward data, $freq$ is the frequency at which the user needs the data, $cond$ is the event condition for the T3 $task$, D is the length of the duty cycle, $minslot$ is the minor timeslot in the task cycle, and $majslot$ is the major timeslot in the task cycle.

Algorithm 1 takes request *R* as an input. It returns *true* if the request is served and *false* if the request is not served. This algorithm evaluates the preemption condition $\left(\frac{1}{\sum_{P=1}^{existingT1Freq}\left(\frac{1}{Freqp}\right)} \geq D\right)$ first. If its value is true, the request can be scheduled else the request will be rejected by returning *false*. If the value of preemption condition is true and *R.task* is either *T1* or *T3* (e.g. a task with a frequency), then it evaluates whether or not the desired frequency is in multiples of a duty cycle or vice versa by finding either of $mod(R.freq, Task\ Cycle)$ and$(mod(TaskCycle, R.freq) = 0))$ condition is true. If the frequency is in multiples i.e. one of these conditions is true, then the algorithm assigns values to *minslot* and *majslot* by using methods GetMinorSlot(*R*) and GetMajorSlot(*R*) respectively.

These methods get *minslot* and *majslot* by finding the next vacant timeslot that can be scheduled. Further ScheduleWSN(*R, minslot, majslot*) method schedules the request on appropriate duty cycle according to the values of *minslot*, *majslot* and returns *true*. Returned value *true* means that the request is scheduled for *T1* or *T3*. In other case when preemption condition is met and *R.task* is *T2*, steps same as *T1* and *T3* will be executed except a difference that, as task *T2* doesn't have any frequency; condition "$mod(R.freq, Task\ Cycle)$ and$(mod(TaskCycle, R.freq) = 0))$" will not be checked. Rest of the steps for *T2* will be same as *T1* and *T3*.

In Algorithm 2, the *DutyCycleTimer* fires every D seconds. The *currentRequests* refers to the collection of all requests running on a given sensor, $mincurrentslot$ is the minor slot id for the current Duty Cycle, $majcurrentslot$ is the major slot id for the current Duty Cycle, $currentnode.id$ is the id for the sensor on which code is executing, and $currentnode.parent$ is the id for parent sensor of the sensor on which the code is executing. The $notified$ is a flag; its value is true if the child of the current node has detected an event.

After Algorithm 1 sends requests to sensors, a new request is added into the *currentRequests* in Algorithm 2. The *DutyCycleTimer* is set to fire ("*DutyCycleTimer fired event*") on each duty cycle frequency tick. On this event, for each request *R* in *currentRequests*, the Algorithm 2 checks whether a $mincurrentslot$ is equal to *R.minslot* and $majcurrentslot$ is equal to *R.majslot*. If both slots match for any request R, the Algorithm 2 continues else no action is taken. Further if *R.task* is *T1* or *T2*, and current sensor is selected for sensing (i.e. $R.sens.contains(currentnode.id)$) then, data is sensed for the phenomenon *R.phenom* using method SenseData(*R.phenom*). Sensed data is aggregated with the data received from

immediate children by method AggregateData() and sent to the parent using SendData(*currentnode.parent*). If the task is either *T1* or *T2* and the current sensor is selected to forward the data (i.e. $R.fwd.contains(currentnode.id)$), then sensor just forwards the data to its parent using method SendData(*currentnode.parent*). Now if the request is *T1* i.e. *R.task* is *T1,* algorithm completes. In case if *R.task* is *T2*, it continues further. The *T2* requests are one-time requests, hence after processing *T2* request $R.minslot$ and $R.majslot$ are set to 0, so that this request will not be served further.

In another case, if the timeslots match i.e. ($mincurrentslot$ is equal to *R.minslot* and $majcurrentslot$ is equal to *R.majslot*), and *R.task* is *T3,* it checks whether current sensor is selected to sense the phenomena by using a check, $R.sens.contains(currentnode.id)$. Then a phenomenon is sensed and condition is tested by checking CheckEvent(SenseData($R.phenom$), $R.cond$). If the condition check has met the event criteria i.e. if this check returns *true*, then the sensor sends a notification to its parent using method SendNotification(*currentnode.parent*). However, if the current sensor is selected as the forwarding node (i.e. value of $R.fwd.contains(currentnode.id)$ is *true*), then sensor assesses whether or not it has received any notifications from its children by checking whether *notified* flag is *true.* If *notified* flag is *true*, it notifies its parent using method SendNotification(*currentnode.parent*). If it has not received a notification (*notified* flag is *false*), it remains idle.

Algorithm 1: BS Scheduling

Objective: Scheduling the input request

Input: $R\{task, phenom, sens, fwd, freq, cond\}$

Output: $true$ if request is scheduled, $false$ otherwise

If $\frac{1}{\sum_{P=1}^{existingT1Freq}\left(\frac{1}{Freqp}\right)} \geq D$

If $R.task \in \{T1, T3\}$

If $((mod(R.freq, D) = 0) \| (mod(D, R.freq) = 0)$) &&

$((mod(R.freq, Task\ Cycle) = 0) \| (mod(TaskCycle, R.freq) = 0))$

$minslot$= GetMinorSlot(R)

$majslot$= GetMajorSlot(R)

ScheduleWSN($R, minslot, majslot$)

Return $true$

Else

Return $false$

If $R.task =$ T2

$minslot$= GetMinorSlot(R)

$majslot$= GetMajorSlot(R)

ScheduleWSN($R, minslot, majslot$)

Return $true$

Else

Return $false$

Algorithm 2: Sensor Scheduling

Objective: Serving the scheduled requests

DutyCycleTimer fired event

For each R in currentRequests

If $(R.minslot = mincurrentslot)$ & $(R.majslot = majcurrentslot)$

If $R.task \in \{T1, T2\}$

If $R.sens.contains(currentnode.id)$

SenseData($R.phenom$)

AggregateData()

SendData($this.parent$)

If $R.fwd.contains(currentnode.id)$

SendData($currentnode.parent$)

If $R.task = T2$

$R.minslot = 0$

$R.majslot = 0$

If $R.task = T3$

If $R.sens.contains(currentnode.id)$

If CheckEvent(SenseData($R.phenom$), $R.cond$) = $true$

SendNotification($currentnode.parent$)

If $(R.fwd.contains(currentnode.id))$ & $(notified = true)$

SendNotification($currentnode.parent$)

5. ALLOCATION

Reducing number of active sensors will result in increasing lifetime of the WSNs. It can be achieved by using a spatial correlation between the sensing ranges of sensors. In Sensor cloud we are using Voronoi diagram to represent deployment of sensors in a WSN. Voronoi diagram can be further used for allocating sensors to a task. This scheme works with the following:

- Densely Deployed WSNs
- Sparsely Deployed WSNs
- Combination of Densely and Sparsely Deployed WSNs

5.1 VORONOI DIAGRAM FOR WSN

Figure 5.1 shows the Voronoi diagram for a WSN in a Sensor Cloud. The rectangular region represents the area from which the user needs data. The area covered by the WSN is divided into small cells that are centered at points. Each point in the cell is represents a wireless sensor in a WSN. The edges of each cell are formed by connecting perpendicular bisectors of the segments joining all neighboring points. The sensor located at the center of a cell can sense data for the area covered by cell. This data is more accurate as compared to the data sensed by other sensors for that region. This Voronoi diagram is built when WSN is initialized.

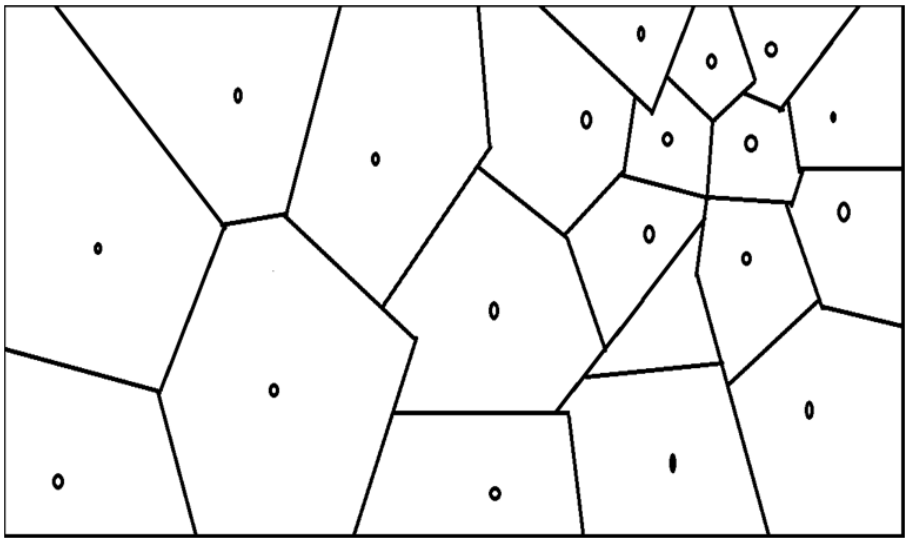


Figure-5.1 Voronoi Diagram for WSN

5.2 MINIMUM NUMBER OF SENSORS REQUIRED TO SENSE THE AREA

Assume a user needs data from a rectangular area in Figure 5.2. Note that, if the required area is not rectangular, it can be extended into a rectangle. This calculation is only to find minimum number of nodes required to cover area selected by the user. Thus, this calculation can be used to extend any irregular area to a rectangular area.

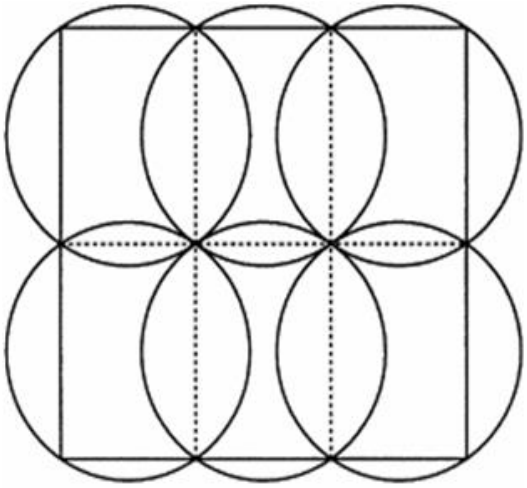


Figure-5.2 Minimum number of sensors needed to cover an area

The outer rectangular area (in Figure 5.2) can be divided into smaller, encircled rectangles of the same size. Center of each circle is a wireless sensor and area encircled represents sensing area of the sensor. Each smaller circle can be represented as shown in Figure 5.3. The radius of circle 'r' represents the sensing range of the sensor.

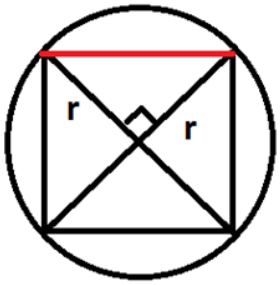


Figure-5.3 Sensed area based on sensing range

$$\varphi = 2 \times r^2 \quad (2)$$

$$\emptyset = \frac{\theta}{\varphi} \quad (3)$$

Where $\emptyset$ is the minimum number of sensors required to cover an area, θ is the area of the entire rectangular surface, and φ is the area of smaller rectangles.

With equation (2) we can find the value of area of smaller rectangles. Equation (2) can be used in equation (3) to find the value of minimum number of sensors required to sense area.

In order to achieve better coverage in implementation, we use twice the minimum number of sensors required.

5.3 ALLOCATION SCHEME

The allocation scheme is based on the concept that sensors in densely deployed zones will have more number of neighbors compared to sparsely deployed zones and hence more number of edges in Voronoi diagram. Similarly, sensors in sparsely deployed zones will have fewer neighbors as compared to densely deployed zones.

Voronoi diagram for a WSN is determined when the network is establishment. It is assumed that BS has knowledge of physical locations of all nodes in the network (as previously noted in section 3.3). The WSN may contain both densely and sparsely deployed zones.

When a user requests data from a specific area, the minimum number of sensors required to cover the area are determined. Then value of selection factor for each sensor is calculated using equation (4).

$$\alpha = \beta \times \gamma \tag{4}$$

Where α is the selection factor, β is the area of each cell in the Voronoi diagram, and γ is the number of edges surrounding a node whose neighbors are not selected for a task.

List of nodes in the sensing area are sorted in descending order of selection factor. Node with the highest selection factor is chosen first. At the beginning when no node has been selected, number of uncovered edges includes all edges surrounding a node. When algorithm makes a selection, number of uncovered edges for all neighboring nodes is decremented by 1. The list of nodes is again sorted in descending order of selection factor. The algorithm iterates through the loop until it exceeds the minimum number of sensors required to sense the area (Please refer to Section 5.2 for more detailed discussion).

5.4 ALLOCATION ALGORITHM

Algorithm 3: Initialization

Objective: Initialization based on Voronoi diagram

Input: $VoronoiDia < cell\ \{sensorid, V, E\ \} >$

Output: $WSN < S\{sensorid, area, neighbors, noofedges, location, selfactor\}>$

For each $cell$ in $VoronoiDia$

$S.sensorid$ = $cell.sensorid$

$S.area$ = GetVoronoiCellArea($cell$)

$S.neighbors$ = GetNeighbors($cell$)

$S.noofedges$ = GetNoOfEdges($cell$)

$S.location$ = GetLocation($cell$)

$S.selfactor = 0$

$WSN.add(S)$

Return WSN

$VoronoiDia$ is the algorithm input that is collection of all voronoi cells within the sensor network. For each $cell$ in $VoronoiDia$, $sensorid$ is the sensor's id, V is the vertex at which the sensor is located, and E is the set of edges enclosing the cell. WSN is output of the algorithm which is collection of sensors in sensor Network. For each S in WSN, $sensorid$ is the sensor's id, $area$ is the area covered by S, $neighbors$ represents the list of neighboring cells of sensor S, $noofedges$ are the number of edges

S has in $VoronoiDia$, $location$ is the sensor's physical location, and $selfator$ is selection factor that will be used in Allocation algorithm (algorithm 4).

Initialization algorithm (algorithm 3) uses Voronoi diagram $VoronoiDia$ as input. It is used to initialize sensor objects S in WSN. Initialized list of S in WSN will be used in Algorithm 4. In algorithm 3, for each *cell* in Voronoi diagram *VoronoiDia*, sensor *S* in collection *WSN* is set. Initially the *S.sensorid* is set to *cell.sensorid.* Later value of voronoi area *S.area* is set using method GetVoronoiCellArea(*cell*). Similarly to find neighbors of the cell *S.neighbors* and number of edges *S.noofedges*, methods GetNeighbors(*cell*) and GetNoOfEdges(*cell*) are used respectively. Value of physical *S.location* is set using methods GetLocation(*cell*). The value of selection factor *S.selfactor* is set to 0 initially which will be changed in algorithm 4 when number of nodes will be selected. At the end of the algorithm, collection *WSN* will be returned.

In Algorithm 4, $SenseReg$ is the input to the algorithm that represents the sensing region requested by the user for sensing, $name$ is the name of the sensing region in the sensor network, $area$ is the area covered by the region, and E is the set of edges enclosing the $SenseReg$. The list of sensor ids selected for sensing purpose is $SelectedSensors$. The variable that stores the value of the minimum number of sensors required for a task is *noofsensors*, and the radius of the sensor's sensing range is *SensingRange.*

Algorithm 4: Allocation Algorithm

Objective: Allocating sensors for a task

Input: $SenseReg\{name, area, E\}$

Output: $SelectedSensors\{sensorid\}$

For each S in Wsn

If $S.location \in$ GetLocation($SenseReg.E$)

$$S.selfactor = S.area \times S.noofedges$$

noofsensors = $\frac{SenseReg.area}{(SensingRange)^2}$

While noofsensors > 0

$WSN.SortDescending(selfactor)$

$SelectedSensors.add(WSN[0].sensorid)$

For each S in WSN

If $WSN[0].sensorid \in S.neighbors$

$S.noofedges = S.noofedges - 1$

$S.selfactor = S.area \times S.noofedges$

$WSN.remove(WSN[0].S)$

noofsensors = noofsensors $- 1$

Return $SelectedSensors$

Unlike Algorithm 3, Algorithm 4 executes for every incoming request to the sensor network. The input is $SenseReg$ and the output is $SelectedSensors$. At the beginning of the algorithm for all sensors (i.e. for each *S* in *WSN*), if they belong to given location which is checked by condition "$S.location \in$ GetLocation($SenseReg.E$)", value of $S.selfactor$

(selection) are calculated. Next, the minimum number of sensors required to sense the area are identified with the using formula *noofsensors* = $\frac{SenseReg.area}{(SensingRange)^2}$. The list of S is then sorted in descending order of $selfactor$ using method $WSN.SortDescending(selfactor)$, and the sensor with a maximum $selfactor$ (which will be the first element in sorted list *WSN*) is selected for sensing. When the sensor is selected, it is added into the list *WSN* using $SelectedSensors.add(WSN[0].sensorid)$. Another *for loop* runs for each *S* in *WSN* to reduce the value of *S.noofedges* of its neighboring sensors and to recalculate value of their selection factor *S.selfactor*. Selected sensor is then removed from list of available sensors in WSNusing *WSN.remove(WSN[0].S)*. After the selected sensor is removed from the list, $noofedges$ for sensors neighboring the selected sensor are decreased by 1. The *while loop* continues in this manner until the number of selected sensors is equal to the *noofsensors*. The list of selected sensors *SelectedSensors* is returned as the output for the task.

6. IMPLEMENTATION AND EXPERIMENTS

In this section, we provide the performance results of our proposed scheduling and allocation algorithms.

6.1 EXPERIMENTAL SETUP

For scheduling scheme, we developed the software to run on telosb motes using TinyOS 2.0 and java. The telosb motes were powered by two AA rechargeable batteries (1.2V - 2600mA). A Sensor Cloud web application programmed using java will communicate with the BSs of different WSNs using RMI and socket communication. BS developed in java then transfers the messages to the motes which are programmed using TinyOS. For allocation scheme, we used an application available on the internet to design the Voronoi diagram. We randomly generated the nodes in the Voronoi diagram in a given region. Then we calculated the area of each cell and gave that as an input to our allocation scheme developed in java and mysql to get results.

6.2 PERFORMANCE EVALUATION

For scheduling scheme we compared the algorithm results with the base case we developed. Also for analyzing the performance of allocation scheme, we compared the results with the results in paper [4].

6.2.1 Scheduling. In order to show the efficiency of our design, we compared scheduling scheme with the base case. In base case, we did not include proposed scheduling scheme but it just has ability to send the data to the BS at a given frequency. There was just one type of task with frequency, which can be executed once at a time. For the experiment, we deployed a WSN with 5 nodes for base case and scheduling algorithm both. The measures we used for performance comparison are, Response Time, Throughput, Network Lifetime and Power Consumption.

6.2.1.1 Response time. To find the Response Time, we considered the tasks of type T1 in our experiment because they are most expensive tasks. For increasing number of tasks we found the values of response time for the best case, the average case and the worst case. Figure 6.1 shows the tasks table used in performing the experiment. Also Figure 6.2 shows the graph of Number of Tasks v/s Response Time (in sec) for scheduling algorithm. In the best-case scenario, the new request for the task arrives when all other previous requests are saved and next duty cycle is vacant to be served. On the other hand we consider a worst case when a vacant duty cycle was just got over and a new request arrives. When the number of tasks are more in worst case, we have taken the tasks with a higher frequency rate which causes the response time to increase rapidly. However, in average case, the new request arrives between the two vacant duty cycles. The response time for average case increases linearly with the number of tasks. The response time for the base case is shown in Figure 6.3, where the overall response time is significantly greater than the response time of scheduling algorithm. The response time of scheduling algorithm is shorter than base case because, in scheduling algorithm, we allow tasks to execute in parallel. On the contrary, for base case they execute serially and hence response time is longer.

Number of tasks	Scenarios - Task Frequency (sec)		
	Worst case	Average case	Best case
1	1. T1=5/ T2/ T3 = 5	1. T1=5/ T2/ T3 = 5	1. T1=5/ T2/ T3=5
	1. T1=10	1. T1=10	
2	2. T1=10/ T2/ T3 = 10	2. T1=10/ T2/ T3 = 10	1. T1=5/ T2/ T3=5
	1. T1=10	1. T1=15	
	2. T1=20	2. T1=15	
3	3. T1=20/ T2/ T3=20	3. T1=15/ T2/ T3=15	1. T1=5/ T2/ T3=5
	1. T1=10	1. T1=20	
	2. T1=20	2. T1=20	
	3. T1=40	3. T1=20	
4	4. T1=40/ T2/ T3=40	4. T1=20/ T2/ T3=20	1. T1=5/ T2/ T3=5
	1. T1=10	1. T1=25	
	2. T1=20	2. T1=25	
	3. T1=40	3. T1=25	
	4. T1=80	4. T1=25	
5	5. T1=80/ T2/ T3=80	5. T1=25/ T2/ T3=25	1. T1=5/ T2/ T3=5
	1. T1=10	1. T1=30	
	2. T1=20	2. T1=30	
	3. T1=40	3. T1=30	
	4. T1=80	4. T1=30	
	5. T1=160	5. T1=30	
6	6. T1=160/ T2/ T3=160	6. T1=30/ T2/ T3=30	1. T1=5/ T2/ T3=5

Figure-6.1 Tasks for the experiment

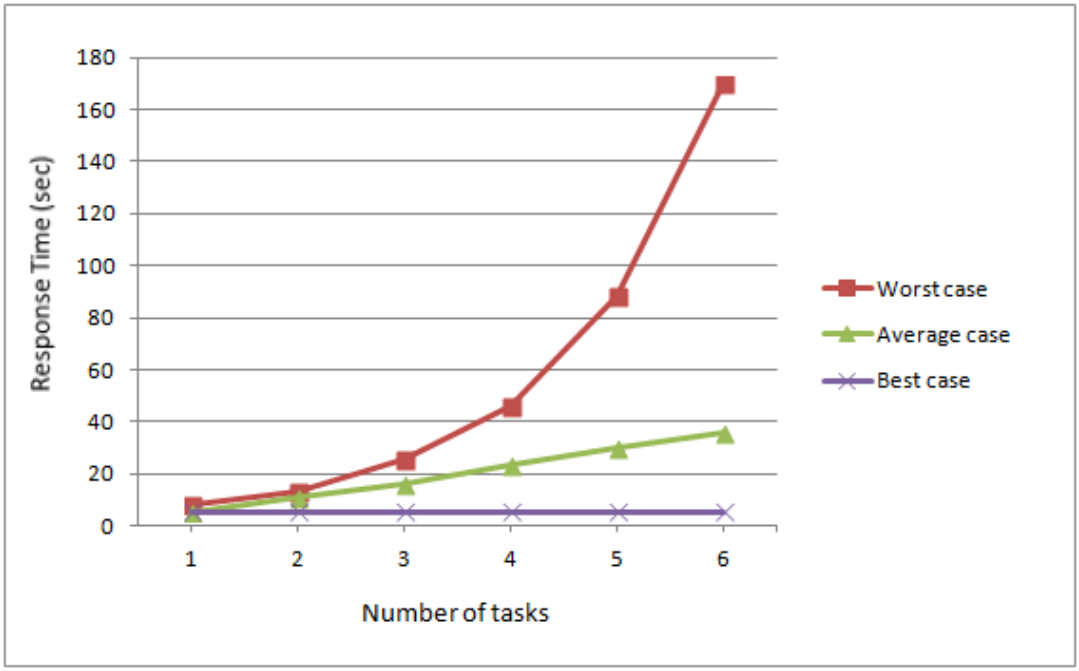


Figure-6.2 Number of tasks v/s Response time (sec) for scheduling algorithm

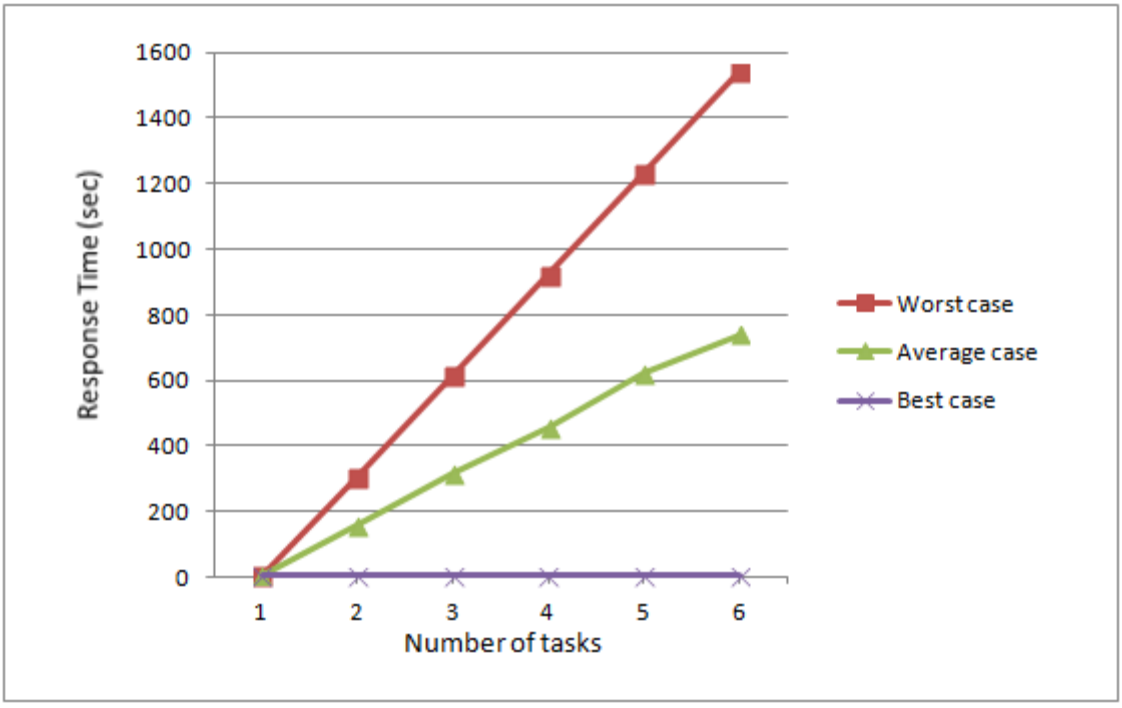


Figure-6.3 Number of tasks v/s Response time (sec) for base case

6.2.1.2 Throughput. For the results of throughput, refer to the table in Figure 6.1. The number of tasks completed with elapsed time for scheduling algorithm is shown in Figure 6.4. Throughput of best and worst case increase linearly, however in the worst case, with increase in time, initially the number of tasks executed goes on increasing, but later they go down. As the elapsed time goes on increasing we select the higher frequencies which cause WSNs to accommodate less number of tasks. Hence in worst case throughput reflects downward slope at higher values of time elapsed. On the other hand, the throughput for the base case in Figure 6.5 was very low as compared to throughput of scheduling algorithm because base case allows serial execution of tasks only.

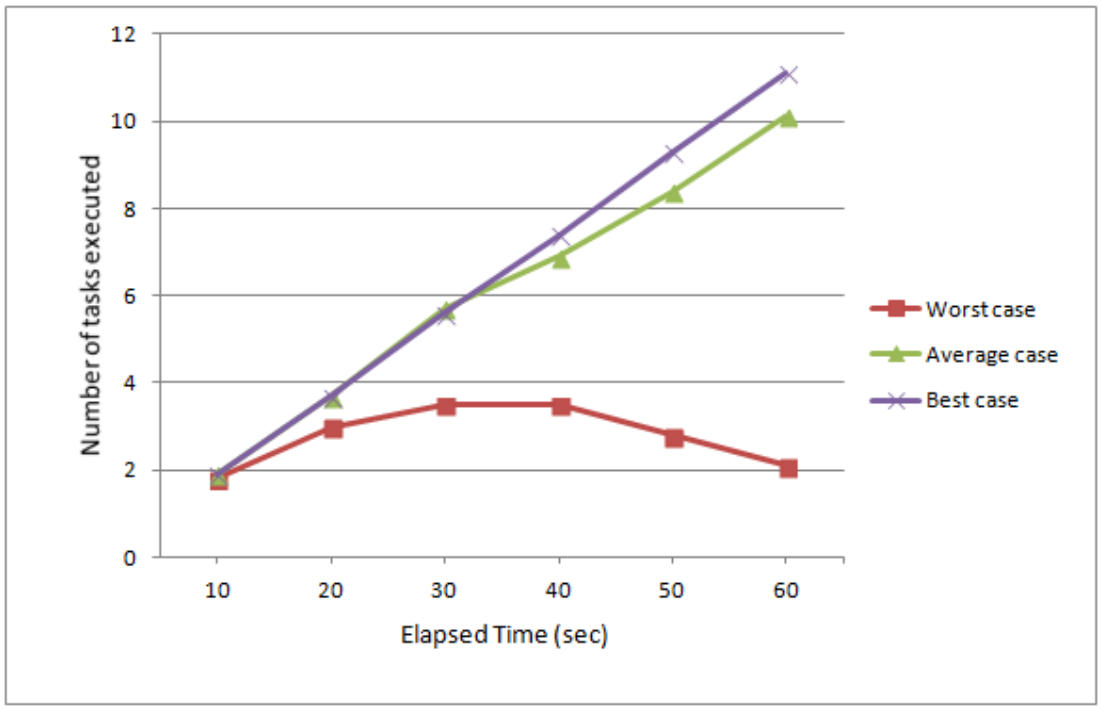


Figure-6.4 Elapsed time v/s Number of tasks executed for scheduling algorithm

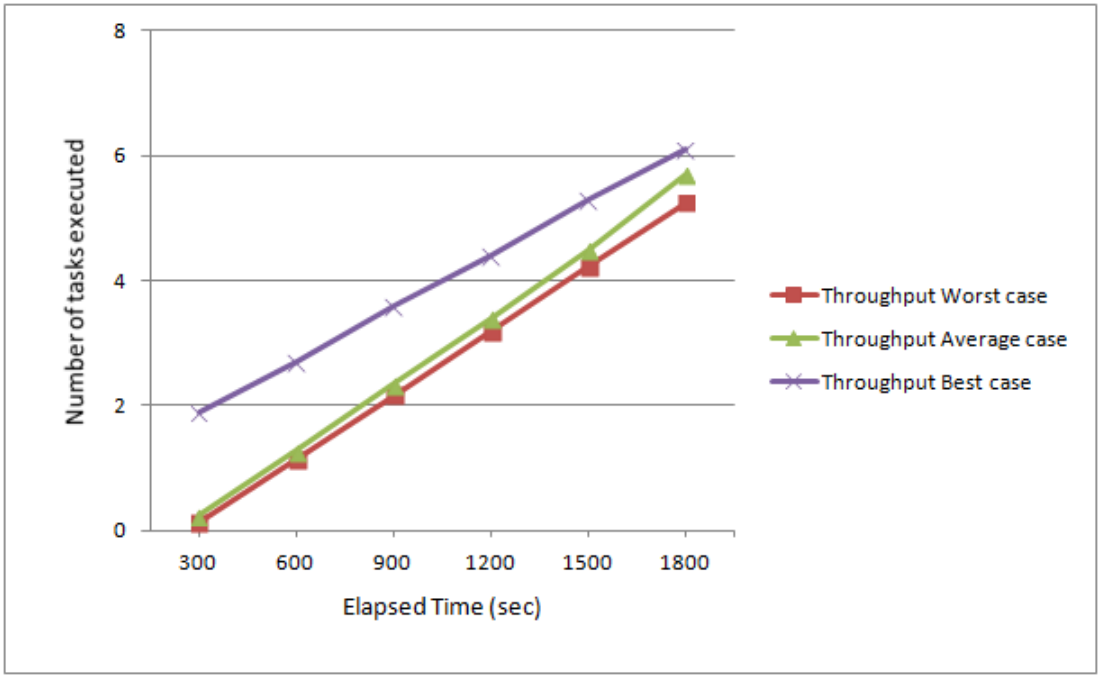


Figure-6.5 Elapsed time v/s Number of tasks executed for base case

6.2.1.3 Network lifetime. In Figure 6.6, we have compared the network lifetime of scheduling algorithm with the network lifetime of the base case. The length of the duty cycle was set as 5 sec. When a task with frequency 5 seconds was executed for infinite duration on the base case, the batteries were discharged in 64 hours. Similarly with scheduling algorithm, we deployed three WSNs each having duty cycle as 5 seconds and were assigned task T1, T2 and T3 respectively. The networks for tasks T1, T2 and T3 lasted for 59, 45 and 80 hours respectively. Four tasks of T1 and T3 are set for WSNs, with frequency of each task as 20 seconds. However, for task T2, we sent a new request every 5 seconds. It is observed that WSN with task T2 has shortest life because it involves the packets sent from BS to node(s) and node(s) to BS both. T1 has more lifespan than T2 because it was a push request, which requires data being pushed from node(s) to BS only. T3 has the highest lifespan as it mostly involves sensing unless the event condition is not met when node(s) have to send notification to BS.

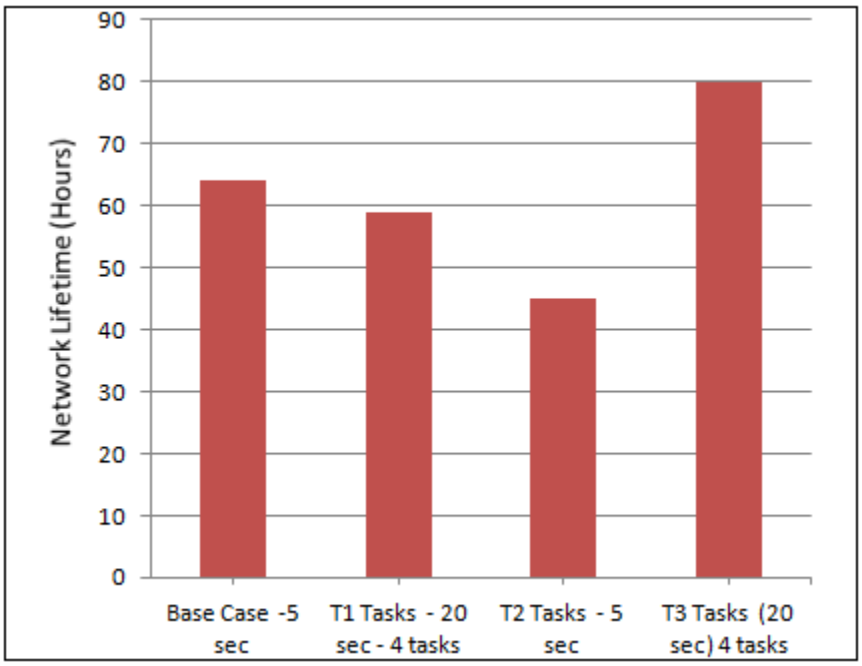


Figure-6.6 Tasks v/s Network Lifetime (hours)

6.2.1.4 Power consumption. Based on the results of Network lifetime the power consumption was calculated and is shown in Figure 6.7. Although the base case consumes lesser power than T1 and T2, the overall power consumption of T1, T2 and T3 is greater than the base case. Task T2 consumes more energy because it involves the packets sent from BS to node(s) and node(s) to BS both. T1 consumed lesser energy than T2 because it was a push request, which requires data being pushed from node(s) to BS only. T3 was the least expensive task because it mostly involves sensing unless the event condition is not met when node(s) have to send notification to BS.

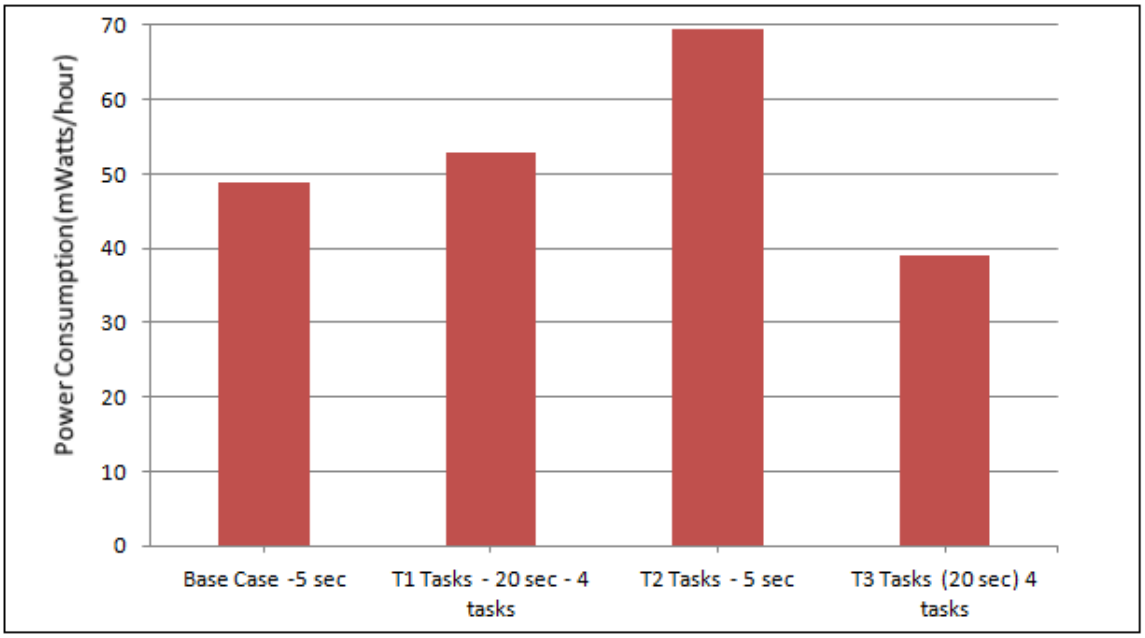


Figure-6.7 Tasks v/s Power consumption (mW/hour)

6.2.2 Allocation. We compared results of allocation scheme with the scheme proposed in [4]. For the experiment, we varied the sensing range of the sensors. Performance comparison was done on the basis of parameters Number of nodes in an ‘on’ state and sensing area coverage.

6.2.2.1 Number of backup nodes. The results of our allocation algorithm and from paper [4] are as shown in Figure 6.8. Similar to [4], we performed experiment on 100 nodes and changed the node density (in *nodes/m*^{2}) by changing the sensing area. For different values of sensing range of the sensors, we found the number of backup nodes against node density. At a lower sensing range of 89m, our scheme selected fewer number of sensors compared to scheme in [4], which covered more than 95% of the sensing area. With increasing values of sensing ranges, our scheme selected lesser number of nodes and provided more than 99% coverage of the sensing area. In addition, for same value of sensing range, at higher node densities, the number of selected nodes goes down. Unlike [4], in our scheme, BS is solely involved while making allocation decision, and therefore, no energy is spend at nodes whereas in [4], allocation is done by sensors and therefore, they consume energy.

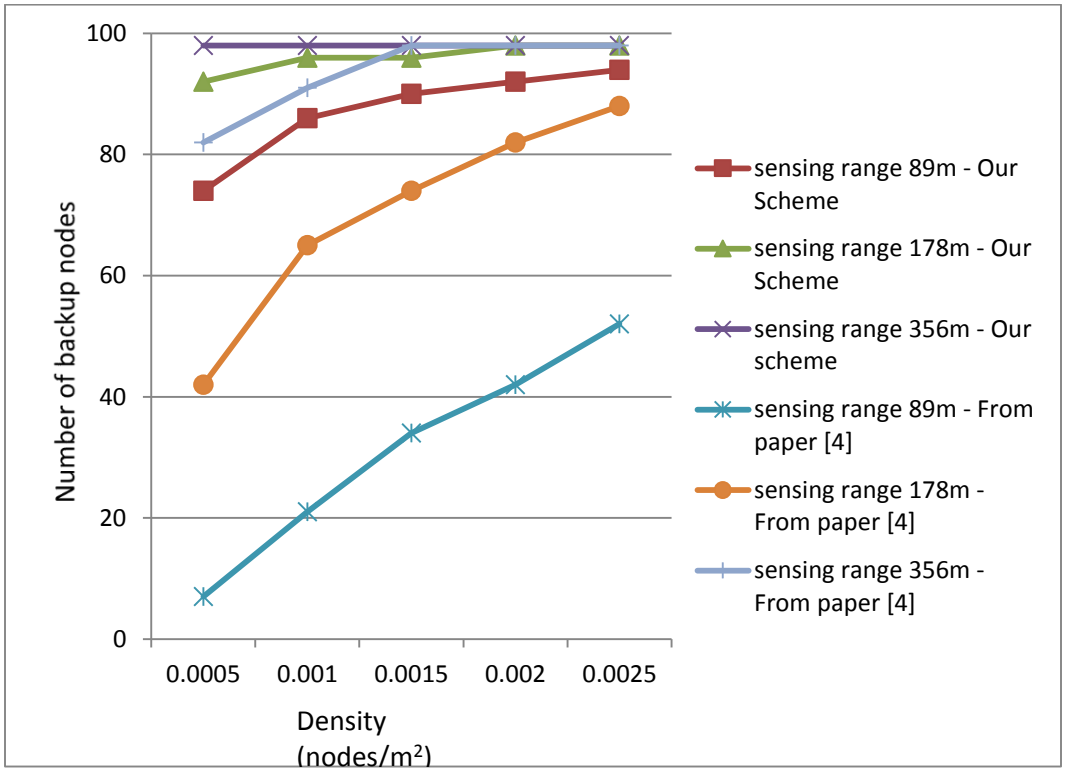


Figure-6.8 Node Density (Nodes/m^2) v/s Number of backup nodes - for our scheme and for scheme from paper [4]

6.2.2.2 Percentage area not covered. Figure 6.9 contains the table for the results of our allocation scheme. It is observed that percentage of area that is not covered goes on decreasing at higher sensing ranges. Although the allocation scheme in this work requires lesser number of nodes compared to the nodes required in [4], it provides area coverage which is close to 100 percent. However, with more number of nodes, approach in [4] always provides 100 percent coverage of the area. When compared with [4], in the worst case, our proposed scheme compromise on the amount of area covered by 4.6% only, but needs 42% lesser number of nodes. At the best case, proposed scheme and scheme in [4] perform same covering area equal to 100% and selecting only 2 nodes for sensing. Thus, in comparison with [4], the percentage improvement in the lesser number of nodes used for coverage compare to a smaller percentage compromise with respect to uncovered area.

	Density(nodes/m2) 0.0025		Density(nodes/m2) 0.002		Density(nodes/m2) 0.0015		Density(nodes/m2) 0.001		Density(nodes/m2) 0.0005	
Sensing Radius	%Backup Nodes	%Area not covered	%Backup Nodes	%Area not covered	%Backup Nodes	%Area not covered	%Backup Nodes	%Area not covered	%Backup Nodes	%Area not covered
89	94	4.26	92	0.26	90	0.08	86	1.46	74	0.04
178	98	0	98	0.16	96	0	96	0.66	92	0.26
356	98	0	98	0	98	0	98	0	98	0.16

Figure-6.9 Percentage area not covered

7. CONCLUSION

In conclusion, the scheduling and allocation scheme for the Sensor Cloud in multi-application environment has been proposed. The scheduling scheme accommodates as many number of user requests as possible. The allocation scheme helps to increase the network lifetime. In our implementation we showed that scheduling and allocation scheme provides energy efficient operation leading to energy conservation in WSNs. The scheduling scheme improves response time, throughput and overall energy consumption over base case. The allocation scheme selects very less number of sensors compared to other scheme and still provides area coverage greater than 95 percent.

REFERENCES

[1] Xu, Y., Helal, S., and Scmalz, M., ”Optimizing push/pull envelopes for energy-efficient cloud-sensor systems.” In proceedings of the 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems., ACM, 2011.

[2] Pantazis, N. A., Vergados, D. J., Vergados, D. D., and Douligeris, C., ”Energy efficiency in wireless sensor networks using sleep mode TDMA scheduling.” In Ad Hoc Networks, 7.2 (2009): 322-343.

[3] Xiong, S., Li, J., Li, Z.,Wang, J., and Liu, Y., ”Multiple task scheduling for low-duty-cycled wireless sensor networks.” In INFOCOM, 2011 Proceedings IEEE, IEEE, 2011.

[4] Viera, M. A. M., Viera, L. F. M., Ruiz, L. B., Loureiro, A. A. F., Fernandes, A. O., and Nogueira, J. M. S., ”Scheduling nodes in wireless sensor networks: A Voronoi approach.” In Local Computer Networks, 2003. LCN’03. Proceedings. 28th Annual IEEE International Conference on, IEEE, 2003.

[5] Pizzocaro, D., Preece, A., Chen, F., Porta, T. L., and Bar-Noy, A., ”A distributed architecture for heterogeneous multi sensor-task allocation.” In Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on, IEEE, 2011.

[6] Vuran, Mehmet C., and Ian F. Akyildiz., ”Spatial correlation-based collaborative medium access control in wireless sensor networks.” In Networking, IEEE/ACM Transactions on, 14.2 (2006): 316-329.

[7] Kapoor, Navdeep Kaur, Shikharesh Majumdar, and Biswajit Nandy., ”Scheduling on Wireless Sensor Networks Hosting Multiple Applications.” In Communications (ICC), 2011 IEEE International Conference on, IEEE, 2011.

[8] Voinescu, A., Tudose, D. S., and Tapus, N., ”Task Scheduling in Wireless Sensor Networks, 2010.” In Networking and Services (ICNS), 2010 Sixth International Conference on., IEEE, 2010.

[9] Cao, Yongle, Shuo Guo, and Tian He., ”Robust multi-pipeline scheduling in low-duty-cycle wireless sensor networks.” In INFOCOM, 2012 Proceedings IEEE, IEEE, 2012.

[10] Aziz, N. A. B. A., Ammar W. Mohemmed, and Daya Sagar., ”Particle swarm optimization and Voronoi diagram for wireless sensor networks coverage optimization.” In Intelligent and Advanced Systems, 2007. ICIAS 2007. International Conference on, IEEE, 2007.

[11] Alsalih, W., Islam, K., Rodrguez, Y. N., and Xiao, H., ”Distributed voronoi diagram computation in wireless sensor networks.” In SPAA, 2008.

[12] Madria, Sanjay, Vimal Kumar, and Rashmi Dalvi, ”Sensor Cloud: A Cloud of Virtual Sensors.” In Software, IEEE, vol. 31, no. 2, pp. 70-77, IEEE, Mar 2014.

4. CONCLUSIONS

This document presents the architecture, design and implementation of the Sensor Cloud which offers sensing as a service to the cloud user. In Sensor Cloud we define virtualization of WSNs and the relations between WSNs to enable virtualization. Implementation of Sensor Cloud is followed by Scheduling and Allocation scheme for WSNs. Scheduling and allocation scheme have been proposed for Sensor Cloud in multi-application environment. The scheduling scheme accommodates as many number of user requests as possible. The allocation scheme helps to increase the network lifetime. In our implementation we showed that scheduling and allocation scheme provides energy efficient operation leading to energy conservation in WSNs. The scheduling scheme improves response time, throughput and overall energy consumption over base case. The allocation scheme selects very less number of sensors compared to other scheme and still provides area coverage greater than 95 percent.

REFERENCES

[1] NIST Cloud Computing Program (http://www.nist.gov/itl/cloud/)

[2] M.M. Hassan, B. Song, and E.-N. Huh, "A Framework of Sensor-Cloud Integration Opportunities and Challenges," Proc. 3rd Int'l Conf. Ubiquitous Information Management and Communication (ICUIMC 09), ACM, 2009, pp. 618–626.

[3] M. Yuriyama and T. Kushida, "Sensor-Cloud Infrastructure—Physical Sensor Management with Virtualized Sensors on Cloud Computing," Proc. 13th Int'l Conf. Network-Based Information Systems (NBIS 10), IEEE CS, 2010; doi:10.1109/NBiS.2010.32.

[4] S. Kabadayi, A. Pridgen, and C. Julien, "Virtual Sensors: Abstracting Data from Physical Sensors," Proc. Int'l Symp. World of Wireless, Mobile and Multimedia Networks (WoWMoM 06), IEEE CS, 2006; doi:10.1109/WOWMOM.2006.115.

[5] K. Aberer, M. Hauswirth, and A. Salehi, "A Middleware for Fast and Flexible Sensor Network Deployment," Proc. 32nd Int'l Conf. Very Large Databases, VLDB Endowment, 2006, pp. 1199–1202.

[6] Xu, Y., Helal, S., and Scmalz, M., "Optimizing push/pull envelopes for energy-efficient cloud-sensor systems." In proceedings of the 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems., ACM, 2011.

[7] Xiong, S., Li, J., Li, Z.,Wang, J., and Liu, Y., "Multiple task scheduling for low-duty-cycled wireless sensor networks." In INFOCOM, 2011 Proceedings IEEE, IEEE, 2011.

[8] Kapoor, Navdeep Kaur, Shikharesh Majumdar, and Biswajit Nandy., "Scheduling on Wireless Sensor Networks Hosting Multiple Applications." In Communications (ICC), 2011 IEEE International Conference on, IEEE, 2011.

[9] Viera, M. A. M., Viera, L. F. M., Ruiz, L. B., Loureiro, A. A. F., Fernandes, A. O., and Nogueira, J. M. S., "Scheduling nodes in wireless sensor networks: A Voronoi approach." In Local Computer Networks, 2003. LCN'03. Proceedings. 28th Annual IEEE International Conference on, IEEE, 2003.

[10] Aziz, N. A. B. A., Ammar W. Mohemmed, and Daya Sagar., ”Particle swarm optimization and Voronoi diagram for wireless sensor networks coverage optimization.” In Intelligent and Advanced Systems, 2007. ICIAS 2007. International Conference on, IEEE, 2007.

VITA

Rashmi Dalvi was born on June 28, 1984 in Mumbai, India. She received her Bachelor of Engineering in Electronics from University of Mumbai, India in May 2006. She completed her PG Diploma in Advanced Computing CDAC from YCP-AIT, Mumbai in Feb 2008. She worked in software industry for 3 years and 6 months on technologies like DotNet, MS-SQL Server, SSAS, SSIS. Since then, she has been a graduate student in the Computer Science Department at Missouri University of Science and Technology. She worked as a Graduate Research Assistant under Dr. Sanjay K. Madria from August 2012 to August 2014. She received her Masters in Computer Science at Missouri University of Science and Technology in December 2014.