Energy Efficient Application Provisioning in Virtualized Internet of Things

Vahid Maleki Raee

A Thesis

In

The Concordia Institute

For

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy (Information and Systems Engineering) at

Concordia University

Montréal, Québec, Canada

March 2023

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Vahid Maleki Raee**

Entitled: **Energy Efficient Application Provisioning in Virtualized Internet of Things**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Information and Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Ahmed Soliman

_____ External Examiner
Dr. Michael A. Bauer

_____ External to Program
Dr. Juergen Rilling

_____ Examiner
Dr. Chadi Assi

_____ Examiner
Dr. Jamal Bentahar

_____ Supervisor
Dr. Roch Glitho

Approved by _____
Dr. Abdessamad Ben Hamza
Chair of Department or Graduate Program Director

Date of Defence: February 20, 2023

_____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Energy Efficient Application Provisioning in Virtualized Internet of Things

Vahid Maleki Raee, Ph.D.

Concordia University, 2023

The Internet of Things is a new paradigm that allows an enormous number of devices i.e., sensors, actuators, RFID tags, etc. to cooperate to reach a common goal. The wireless sensor networks as the key components of the IoT are extensively being used in various domains and applications. However, in traditional WSNs, applications are embedded into the sensor, precluding them from being re-used by other applications. Therefore, the sensor become application-specific and task-oriented devices with increased deployment and maintenance costs. To cope with these issues, a viable approach is to apply virtualization to WSNs. Virtualization abstracts the physical sensing capabilities of the sensors into logical units, allowing them to be reused by multiple applications. WSN virtualization can be done at either node- and/or network-level. However, virtualization challenges the energy consumption in WSNs. Thus, inefficient application provisioning can have a drastic impact on the energy consumption of the WSNs, leading to faster depletion of sensor nodes' batteries. This thesis proposes algorithmic approaches to tackle the key challenges related to energy consumption in virtualized IoT-based networks.

The first challenge faced by virtualized IoT networks is the energy efficiency in dynamic task assignment considering node-level virtualization. Addressing this challenge is critical for energy efficiency. Considering that the IoT devices (e.g., sensors) need to interact and exchange messages, the second challenge is the problem of energy efficiency in dynamic network embedding in virtualized IoT networks considering both node- and network-level virtualization. Yet, another challenge is energy-efficient distributed task assignment in virtualized IoT networks. The IoT nodes are constrained devices with limited available energy and processing capabilities. Thus, it is not always feasible to have powerful nodes in the network to execute the algorithms.

To tackle the first challenge, we modeled the problem using integer linear programming and proposed a heuristic to solve the problem. For the second challenge, after modeling the problem using ILP, we proposed our Dynamic Network Embedding heuristic to solve the problem in an energy efficient manner. When it comes to the third challenge, we modeled the problem using non-cooperative game theory and proposed an energy-efficient heuristic to solve the problem.

Acknowledgments

I would like to begin the acknowledgments with a quote from my supervisor that I was told during the first term of my Ph.D. studies:

"*The research is all about patience. You need to be patient. There are no shortcuts, no rush, and no turnovers. It's a journey. We know the objective, and we know we want to get there, but we don't know how. We should try and find the way to it.*"

At the outset, it is my obligation to extend my sincere appreciation to my Ph.D. supervisor, Prof. Roch Glitho, and express profound thanks to him for his patience, great support, continuous guidance, constructive feedback, and encouragement. Surely, this challenging yet exciting and interesting journey would have not been completed without him, and I would have not been stood where I am today.

I gratefully acknowledge my thesis committee members, Prof. Chadi Assi, Prof. Jamal Bentahar, Prof. Juergen Rilling, Prof. Ferhat Khendek, and Prof. Michael A. Bauer for their time, insightful comments, encouragement, and the questions that had immense impacts to enhance the quality of this dissertation in various aspects.

I would also like to take this opportunity and thank my collaborators Prof. Diala Naboulsi, Dr. Amin Ebrahimzadeh, and Dr. Zoubeir Mlika for the time they dedicated, and for all their support and contributions. Collaborating and working with them has been nothing but joy, honor, pleasure, and reward.

Moreover, I reserve my appreciation to Telecommunication Service Engineering (TSE) research lab members, colleagues, and friends for their support and encouragement. Thanks to Milad Zaheri, Mohammad Nazmul Alam, Sasan Sabour, Aida Rangy, Razieh (Raha) Abbasi, Nasim Rahmani, Carla Mouradian, Abbas Soltanian, and Nattakorn Promwongsa for all their support. It was a great pleasure to know them, and the interesting conversations we always had. Many thanks to Saeed Sarencheh for his mentorship and guidance. Appreciations to Matt Naslcheraghi for about three years of collaboration at IEEE Young Professionals (YP) Montréal Section. My special thanks to Marsa Rayani and Sepideh Malektaji for all the insightful discussions we have always had, for their kind supports, encouragement, and motivation before the deadlines, and of course, for all the fun and coffee breaks during those memorable days in the research lab, since the beginning of our Ph.D. journey till the end.

I also would like to appreciate our department's graduate program directors and administrative staff, Mireille Wahba, Silvie Pasquarelli, and Kim Adams for their help and support.

Last but not least, I would like to extend my sincere gratitude and appreciation to my parents and brothers for their support, care, and love throughout my Ph.D. journey. No words can describe or express my gratitude and love for you. Certainly, none of my achievements and dreams would have become true without you all.

Finally, I would like to conclude the acknowledgments with a quote from my supervisor:

"*Have you seen a kid? When he grows up, at first, he starts crawling. A while later, he starts walking slowly, then he can walk with no help, then he starts running. You as a researcher are simply similar to that kid. At first, you have to start learning to crawl, then stand up and walk, then only start running. If you are too good and exceptional, you will be like Usain Bolt! Don't jump to conclusions so quickly.*"

Hope I have made it through, and learned to run.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Overview

Internet of Things (IoT) is centered around machine-to-machine (M2M) communications with a focus on smart devices (e.g., sensors and actuators) [7]. These connected devices enable sensors, actuators, radio-Frequency IDentification (RFID) tags, and mobile phones to interact with each other and cooperate with their neighbors to reach a common goal [8]. According to Cisco, it is anticipated to have over 500 billion connected devices by 2030 [9]. Wireless Sensor Networks (WSNs), as the main enabling technology to realize the IoT, comprise distributed devices, which gather ambient information by sensing, processing, and communicating wirelessly. They are widely being used in various application domains ranging from smart home, smart city, and smart disaster management to e-healthcare [10]. In traditional WSNs, applications are embedded in sensor nodes, thus hindering them from being re-used by other applications [6]. This raises critical issues, including the redundant deployment of WSNs, leading to inefficient resource utilization and potentially drastic impacts on maintenance and energy costs [6]. To address this issue, WSN virtualization is known as a promising technology, which enables abstracting the physical sensing capabilities of sensor nodes into logical sensing capabilities, turning them from application-specific, task-oriented devices into multi-purpose devices. The virtualization can be applied at node- and/or network-level. [6]. Despite all the benefits that virtualization may offer, it comes at a cost in terms of energy consumption and delay [11, 12]. This is important as virtualization is intrinsically advantageous, however, there is always virtualization instantiation (node-level virtualization) overhead. These two factors become critical, especially given that sensor nodes typically operate on batteries with limited available energy. Also, increased delays could be detrimental to delay-sensitive WSN applications, such as firefighting applications, flood and earthquake detection, and medical alert responses, among others. It is therefore crucial to design an energy efficient mechanism to allocate resources in virtualized WSNs while meeting the given QoS requirements.

Besides, energy-efficient dynamic network embedding is another interesting aspect that requires further investigation in virtualized WSNs (vWSN). Unlike the previous challenge where the application's sensing tasks are solely executed on the sensor nodes without any cooperation between the sensor nodes, in this problem, the physical and virtual sensor nodes may require to collaborate and interact with each other to address the application's sensing task requests. The main challenge is to create customized Virtual Sensor Networks (VSNs) per application request over deployed vWSN infrastructure [13]. This is commonly known as Virtual Network Embedding (VNE) problem [14]. In VNE, the applications send Virtual Network Requests (VNRs). Each VNR comprises a group of virtual nodes, which are connected by virtual links, forming the so-called Virtual Network (VN). The VNs then must be mapped onto the physical substrate network [13]. However, there are a few challenges associated with this problem. Firstly, the complexity of the substrate network is higher than those of traditional WSNs with no virtualization. This is because the substrate network is constructed from both non-virtualizable sensor nodes and those sensor nodes with virtualization mode enabled. Hence, an inappropriate assignment of the sensing task to this infrastructure challenges energy efficiency. Secondly, considering that the sensor nodes do need to interact with each other, therefore, the SLAs (e.g. end-to-end latency requirements set by the applications) become important especially if there are mission-critical applications such as fire contour applications. Thus, it is critical to have an energy-efficient network embedding algorithm to map these VNs to the pool of physical and virtual sensors while meeting the required QoS.

With the growth of the Internet of Things and massive machine-to-machine communications in these networks, a transformation from centralized systems to fully distributed systems from architectures to technologies is necessary. This is mainly because it is not always feasible to have a resource-rich device inside the WSN to execute the algorithms in a centralized manner, especially in large-scale scenarios such as fire-fighting applications. However, energy efficiency in these networks remains a challenge, given that the WSN sensor nodes as part of the IoT network are resource-constrained devices with limited available energy and computing capabilities. The issue becomes more critical and challenging when dealing with virtualized WSNs as virtualization challenges the energy in terms of overhead. Thus, it is important to design and model a distributed and energy-efficient algorithm in WSNs to assign the sensing tasks to the physical and virtual sensors while satisfying the required QoS parameters.

1.2. Thesis Contributions

Unfortunately, the current solutions proposed thus far do not fully address all these challenges. This Ph.D. thesis proposes algorithmic solutions to tackle the challenges related to application provisioning in virtualized IoT networks. There are three main contributions that are presented as follows. Each of our contributions corresponds to a challenge addressed by this thesis.

1.2.1 Energy Efficient Task Assignment in Virtualized Wireless Sensor Networks [1, 2]

Traditional non-virtualized Wireless Sensor Networks (WSNs) suffer from high deployment and maintenance costs, mainly because their applications are embedded in sensor nodes, making them task-oriented, application-specific devices, leading to redundant physical network deployment per application. Virtualization technologies address these challenges by allowing multiple sensing tasks from different applications to run over the same deployed WSN infrastructure. However, virtualization comes at an energy-delay cost, making it both essential and challenging to allocate physical and/or virtual resources efficiently to applications with different sensing tasks, especially for delay-sensitive applications. As our first contribution, we tackle the problem of energy-efficiency in dynamic task assignment in virtualized WSNs while meeting the given task deadlines. After formulating the problem as an Integer Linear Programming (ILP), we propose a scalable heuristic. We evaluate the performance of our proposed heuristic in different scenarios and compare it with the optimal solution as well as recent work from the literature. The results indicate that our proposed heuristic leads to close-to-optimal solutions with good performance in terms of execution time. It also shows that the proposed heuristic cannot only improve the execution time in the large-scale scenario but also outperforms the existing benchmarks in terms of successful task execution rate.

1.2.2 Energy Efficient Network Embedding in Virtualized Wireless Sensor Networks [3, 4]

The second contribution is focused on energy-efficiency for dynamic network embedding in virtualized WSNs. Considering that there is a wide range of applications deployed in WSNs, some of these applications (e.g. fire contour) may require the sensor nodes to interact with each other and exchange messages to address the application's requirement. This is however in contrast with some other applications (e.g. firefighting) where sensor nodes may not necessarily require to cooperate. This becomes even more critical when we are dealing with virtualized WSNs as the underlying infrastructure is constructed from both virtualizable and non-virtualizable sensor nodes. We address the problem of dynamic network embedding in virtualized WSNs, aiming at minimizing the overall energy consumption while considering the end-to-end latency and bandwidth consumption as the Service Level Agreement (SLA) constraints. We formulate the problem using ILP and propose our Dynamic Network Embedding (DNE) heuristic for large-scale problem instances. The results reveal that our proposed heuristic achieved close-to-optimal results while outperforming the existing solutions in terms of energy consumption.

1.2.3 Energy Efficient Distributed Task Assignment for Virtualized Internet of Things [5]

With the emergence of disruptive applications and their immense demands with stringent requirements, decentralization plays a vital role in dense Internet of Things (IoT) networks. Efficient resource utilization is critical given that the IoT nodes are resource-constrained devices. Unlike the previous contributions where the centralized approaches

were considered, in the third contribution we investigate the problem of energy-efficiency in distributed task assignment for optimal utilization of resources as well as the enhanced admission rate in virtualized IoT networks. After modeling the problem as a distributed non-cooperative game, we design a matching algorithm to solve the problem while considering dynamicity, end-to-end latency, bandwidth, and task deadline constraints. The obtained results demonstrate the feasibility of the proposed solution in large-scale scenarios and its superior performance over a scheme without any virtualization and recent work from the literature.

1.3. Background Information

This subsection presents the background information that is relevant to our research domain. The background information covers the Internet of Things, Wireless Sensor Networks, virtualization, IoT virtualization, Virtual Network Embedding, and distributed system.

1.3.1 Internet of Things

Internet of Things (IoT) is centered around machine-to-machine (M2M) communications with a focus on smart devices (e.g., sensors and actuators) [7]. These connected devices enable sensors, actuators, radio-Frequency IDentification (RFID) tags, and mobile phones to interact with each other and cooperate with their neighbors to reach a common goal [8].

1.3.2 Wireless Sensor Networks

There are several enabling technologies to realize the IoT. As an example, Wireless Sensor Networks (WSNs) consist of distributed autonomous devices that gather ambient information by sensing, processing and communicating wirelessly. The applications' sensing tasks are executed over these devices. These tasks are non-divisible execution units of the requested applications. The sensor nodes execute the tasks to collect information about the surroundings such as a space (location, velocity), an environment (luminosity, level of noise), or physiology (blood pressure, heartbeat) [10].

1.3.3 Virtualization

Virtualization as the enabling technology of cloud computing has been applied in various domains including the IoT. Virtualization in general refers to creating an abstraction layer on top of the physical computer so that its resources (computing, storage, memory) can be divided into multiple virtual computers called Virtual Machines (VMs). Each

of these VMs can run its own Operating System (OS) independently. This results in resource utilization efficiency enhancements [15].

1.3.4 IoT Virtualization

The traditional WSNs are domain-specific and task-oriented, allowing them to support only a single application, once deployed. This causes inefficient resource utilization and leads to redundant WSN deployment once a new application is contemplated. By integrating the virtualization technology with WSNs, it abstracts the physical sensing capabilities into logical units, allowing the WSNs to be shared among different users and applications [6]. Virtualization can be applied both at the node- and network-level. Next, we will discuss each approach in detail and its difference from middleware-based (cloud-based) solutions.

A) Node-level Virtualization

The main concept of node-level virtualization is to allow multiple application tasks to be executed over a single sensor node concurrently. Each task is run by a virtual sensor instantiated on top of the physical sensor node. Figure 1.1 depicts the high-level view of node-level virtualization with applications requesting sensing tasks. Each task is then executed by the virtual sensors instantiated on top of the physical sensor nodes, allowing multiple tasks to co-exist and reside over the same deployed sensor node simultaneously.

From the implementation perspective, to achieve node-level virtualization, the execution must be done either sequentially or simultaneously. Sequential execution means that the application tasks' execution happens in series while simultaneous execution refers to executing the tasks in a time-sliced fashion by rapid context switching among the tasks. Sequential execution is simple to implement; however, it implies that application tasks have to wait in a queue. The simultaneous execution implies less delay for an application task, as it will not be blocked by other tasks with longer running time; however, it is more complex to implement [6].

The node-level virtualization approaches are categorized into Operating System (OS) based solutions and Virtual Machine/Middleware-based (VM/M) solutions. In OS-based solutions, virtualization is part of the sensor OS. In VM/M approach, the virtualization functionality is implemented on top of the underlying sensor OS. If a physical sensor can only run one type of sensing task (e.g., temperature), each virtual sensor running on top of the physical sensor will sense temperature, but at the frequency set by the application that created it. It is therefore not a data-sharing sensor, since in data sharing the data is sensed once at a given frequency and then shared by the applications. If a physical sensor can run several sensing tasks (e.g., temperature + humidity), each virtual sensor instantiated on top of it will sense either temperature or humidity, but only at the frequency set by the application that created it. Several commercial products are now available for node-level virtualization. Two examples are the Advanticsys kit [16], which is an OS-based

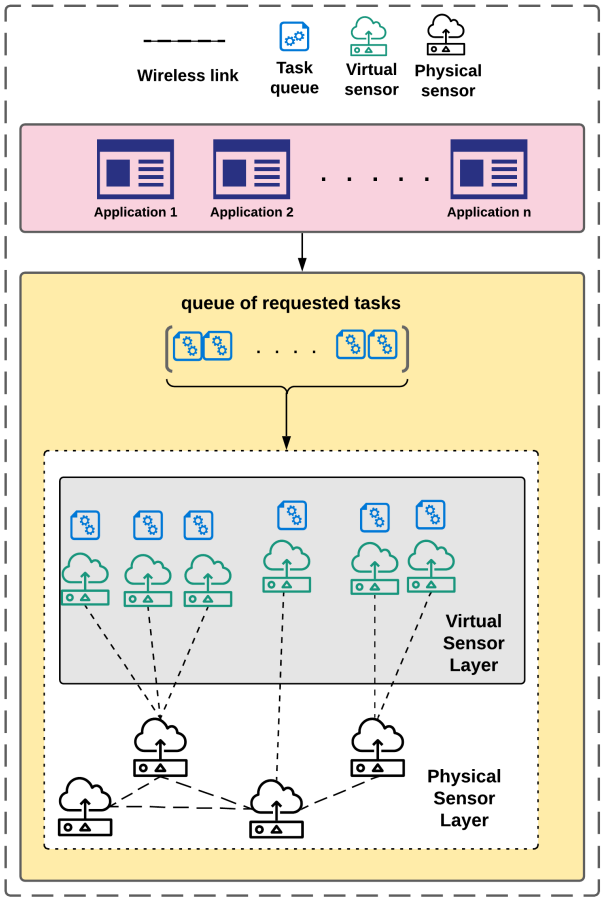


Figure 1.1: Node-level virtualization overview.

approach, and the Virtenio kit [17], which follows a VM/Middleware-based approach. The Advanticsys sensor nodes run the Contiki OS [18] and virtualization is performed by modifying it while the Virtenio sensor nodes are virtualized through Java Virtual Machine (JVM). These sensors allow the creation of Virtual Sensors (VS) on top of them. These VSs can execute different independent applications' sensing tasks concurrently over the very same physical sensor node. Figure 1.2 illustrates different node-level virtualization approaches in detail.

B) Network-level Virtualization

In network-level virtualization, a subset of WSN nodes is dedicated to one application at a given time, giving way to the so-called Virtual Sensor Networks (VSNs) [19]. Network-level virtualization can be realized by either creating multiple VSNs over the same underlying WSN infrastructure or creating a single VSN over different independent WSNs. The first approach partitions the sensor nodes where multiple sensor nodes are grouped (sensors can be part of different groups). The sensor nodes within each group then collaborate to address an application. In the second approach, logical networks are created on top of the physical WSNs. These logical networks enable the data

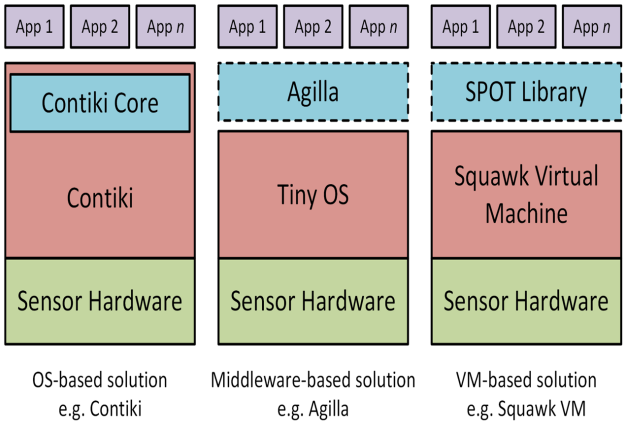
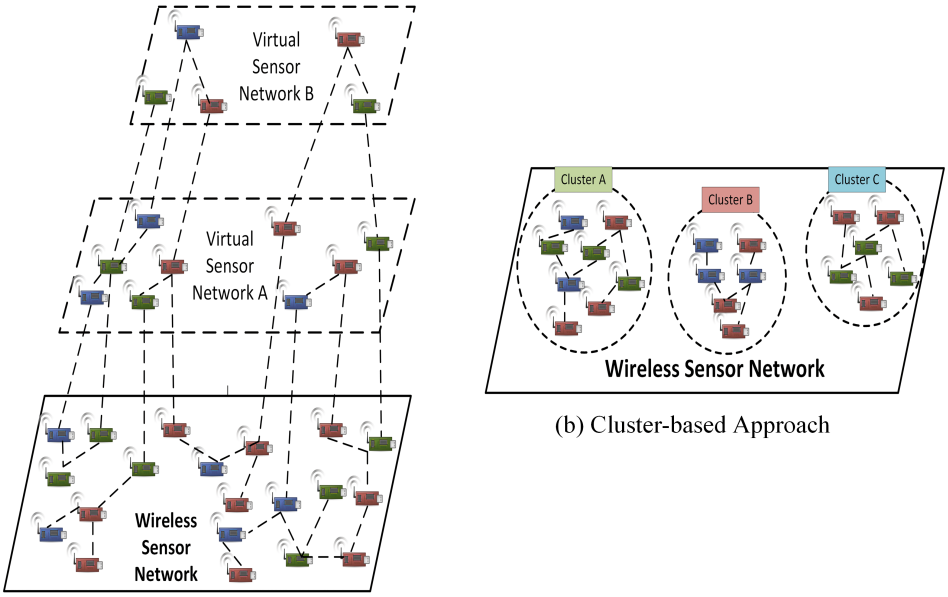


Figure 1.2: Node-level virtualization solutions. [6].



(a) Virtual Network/Overlay-based Approach

(b) Cluster-based Approach

Figure 1.3: Network-level virtualization solutions. [6].

exchange between different sensor nodes even if they are located in different administrative domains [6]. Therefore, the network-level virtualization approaches can be categorized into two groups: Virtual Network/Overlay-based solutions and Cluster-based solutions. Yet, there is a possibility to have both approaches applied together in a network. Virtual Network/Overlay-based solutions are logical networks created on top of a physical network(s). Depending on the application requirements, several virtual networks can be created and embedded onto the physical deployed WSN infrastructure. On the other hand, in a cluster-based solution, the nodes in a physical WSN are grouped to collaborate, which is called clusters. Figure 1.3 shows different solutions to network-level virtualization.

C) Middleware-based Solutions

In a middleware-based solution, the concept of cloud-based WSNs is introduced [20–23]. The middleware-based solutions mainly include creating an instance of the physical sensor nodes in an interface such as VMs which are located in the cloud. These VMs are being managed by Virtual Machine Managers (VMMs)/hypervisors. Then, the data collected by these instances are shared among multiple applications [24].

D) Node-level Virtualization vs. Network-level Virtualization vs. Middleware-based Solutions

Initially, let's consider the node- and network-level virtualization. There are a few points that they have in common. First, both aspects prevent redundant deployment of WSNs and hence reduce the deployment and maintenance cost. Second, they are both scalable which allows them to support a higher number of applications simultaneously. Third, they both support elasticity since the creation and deletion of the virtual sensors over the physical sensor nodes and the composition and decomposition of virtual sensor networks can be done dynamically based on the application tasks. However, the key difference between node- and network-level virtualization is that in node-level virtualization, the capabilities of the sensor nodes are used to execute multiple independent tasks [6], whereas, in network-level virtualization, a subset of the sensor nodes within a WSN is used to cooperate and execute one application while the remaining sensor nodes may be used to run another application [6, 25]. The main advantage of using node-level virtualization is its ability to fully exploit the capabilities of sensor nodes to run various applications' sensing tasks simultaneously, thereby leading to increased efficiency in resource utilization.

Besides, there are key points that distinguish node-level virtualization from middleware-based solutions. In node-level virtualization, the sensing capabilities of the physical sensor nodes are shared among multiple applications whereas, middleware-based solutions share the collected data among the applications. There are multiple advantages that node-level virtualization has as compared to the middleware-based solution. First, in a middleware-based solution, the collected sensed data by the underlying deployed WSNs are being reported at a fixed interval rate while the data might not have been requested by any application. Second, as an example, let's assume that the sensor nodes are programmed to report an event at every 30sec time interval, but the application requires the data every second, therefore, the middleware either has to provide staled data or it can't provide the data at all. Hence, the application requirement is limited to the features exposed by the middleware, even if the requirement can be fulfilled by the IoT device. However, in node-level virtualization, a virtual sensor can be instantiated on top of the sensor node, on-demand as per application request. Finally, all the sensor node's capabilities need to be reported to the middleware even though it is not used by any application. The reason is that if any application requests that capability, then there will be no way to provide it other than reprogramming the underlying IoT device, which may lead to downtime for other applications that rely on it.

So, the middleware solutions are less efficient in terms of cost, energy, and resource utilization than the node-level virtualization solutions [11]. Similarly, what differentiates network-level virtualization from middleware-based solutions is the fact that in network-level virtualization, multiple tasks can coexist simultaneously in the same deployed WSN. In contrast, in middleware-based solutions, the sensor nodes have to report the gathered data at a fixed time interval, otherwise reprogramming of the sensor nodes is required which can cause an interruption in the information they deliver. Moreover, the application is constrained to the features that are exposed by the WSN without the possibility of receiving what is required.

E) Virtual Sensor vs. Virtual Machine

The similarity between a VM and a VS at an abstract level is that both provide a mechanism to decouple physical resources from their underlying infrastructure to be utilized by multiple users [26]. However, there are several key differences. First, the VM is a logical unit that enables sharing of the resources (e.g. computing and storage) of the host machines by dividing them into several dedicated execution environments. Each of the VMs has a guest OS that can access underlying hardware/infrastructure. On the other hand, a VS is a logical representation of the physical sensor node that enables the sharing of the sensor node's sensing capabilities (e.g., temperature, humidity) among multiple applications concurrently. Second, multiple VMs can be deployed on the host machine simultaneously and each may have a different OS. However, the VSs on the other hand are tightly dependent on the sensor nodes' OS/middleware. This implies that a sensor node cannot have two OS (e.g. Contiki and Tiny OS) at the same time. Third, there is a standard for VM addressing which allows every VM to be recognized by an IP address, however, there is no standard mechanism for VS addressing. The VS are mainly being addressed by local IDs. Fourth, VMs may not encounter power/energy-related issues, however, the creation of the VSs is highly dependent on the available energy of the host sensor node [26].

1.3.5 Virtual Network Embedding

The primary concept in Virtual Network Embedding (VNE) is the mapping of virtual nodes and links onto physical resources. Virtual nodes are interconnected by a set of virtual network links, forming a Virtual Network (VN) on top of the Substrate Network (SN) [13].

The main idea of network embedding is to create customized Virtual Sensor Networks (VSNs) per application request over deployed WSN infrastructure [13]. In VNE, the applications send Virtual Network Requests (VNRs). Each VNR comprises a group of virtual nodes, which are connected by virtual links, forming the so-called Virtual Network (VN). The VNs then must be mapped onto the physical substrate network [13].

Unlike wired networks and traditional non-virtualized WSNs, in virtualized WSNs the substrate network consists

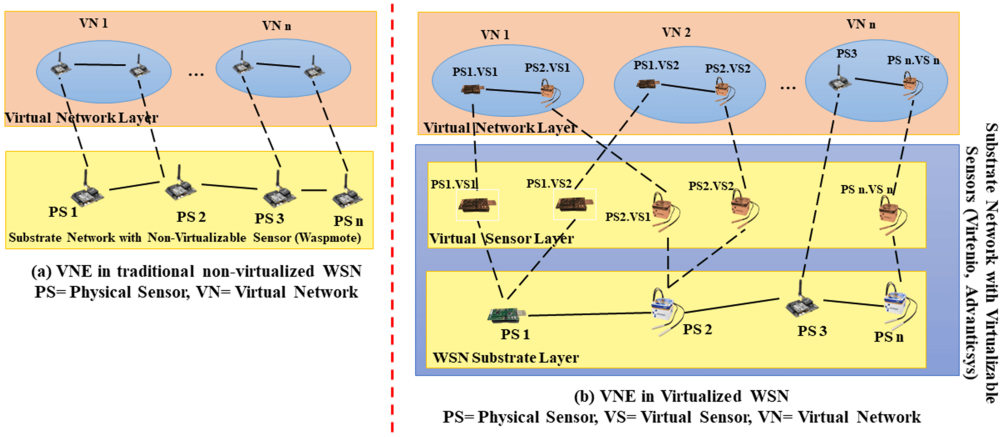


Figure 1.4: VNE in traditional WSNs vs. VNE in virtualized WSNs.

of both physical and virtual sensors which can be assigned to the application sensing tasks. Figure 1.4 illustrates a generic schematic of substrate networks in traditional non-virtualizable WSNs vs. virtualized WSNs. As shown in Fig. 1.4 (a), the substrate network in traditional WSNs consists of non-virtualizable sensor nodes (e.g., Waspmote [27]), which can only execute one sensing task at a time. By contrast, multiple virtual sensors can be instantiated on top of the virtualizable sensor nodes (e.g., Advanticsys [16] and Virtenio [17]), thus allowing for executing multiple sensing tasks of different applications concurrently (see Fig. 1.4 b).

1.3.6 Distributed System

By definition, "a distributed system is a collection of independent entities that cooperate to solve a problem that cannot be individually solved" [28]. Given that the WSNs are intrinsically distributed, hence, it is not always feasible to execute an algorithm by a single entity within the WSN. On the other hand, considering the energy limitations of the sensor nodes, different nodes (e.g. sink nodes) within a WSN may require to execute different processes of an algorithm and cooperate to accomplish a given application request. Therefore, a distributed algorithm in such systems is needed. The key difference between a centralized algorithm and a distributed one is that in a centralized algorithm, a resourceful entity (e.g. base station) or a sink node outside of the network is responsible to execute the algorithm without any necessity for possible cooperation with other sensor nodes and devices inside the network. It is critical to note that there are differences between distributed computing and parallel computing and concurrency. As described earlier, in a distributed system, different entities may process a different function of a defined algorithm and then cooperate to complete a given application request, while in parallel computing, a given task may be divided into several sub-tasks and each sub-task to be completed by a different CPU [29]. On the other hand, concurrency mainly refers to the completion of multiple computations simultaneously on a shared CPU [30].

Figure 1.5 illustrates a generic schematic of centralized vs. decentralized vs. distributed traditional non-virtualized

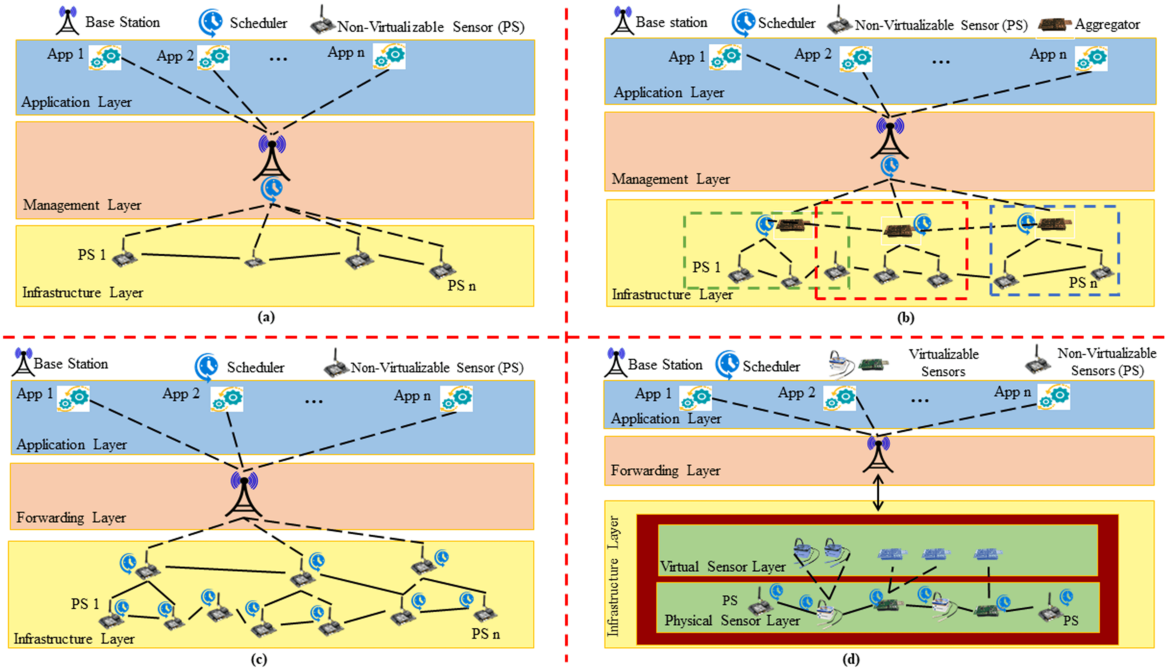


Figure 1.5: Task assignment in (a) centralized traditional non-virtualized IoT, (b) decentralized traditional non-virtualized IoT, (c) distributed traditional non-virtualized IoT, (d) distributed virtualized IoT.

IoT vs. distributed virtualized IoT. As shown in Fig. 1.5 (a), in centralized traditional non-virtualized IoT there exists a scheduler deployed at a resource-rich node (e.g., base station), which is located outside of the deployed network. The scheduler holds global knowledge about the deployed network and runs the assignment algorithm in a centralized manner. In the decentralized task assignment shown in Fig. 1.5 (b), the scheduler has partial knowledge of the underlying networking infrastructure. There are several aggregators inside the network, each holding a scheduler. The aggregators are communicating with the base station and the underlying deployed infrastructure within their network in parallel without any collaboration with other aggregators. The results are then forwarded to the base station. Figure 1.5 (c) depicts the distributed task assignment in a traditional non-virtualized IoT, where each sensor has the knowledge of its neighboring nodes. Each node along with its neighbors cooperate with each other to assign and execute the sensing tasks. As shown in Figs. 1.5 (a-c), the deployed IoT network consists of non-virtualizable sensor nodes (e.g., Waspmote [27]), which can only execute one sensing task at a time. In addition, the scheme shown in Fig. 1.5 (a) suffers from the centralized execution of the assignment. This problem has been partially solved in decentralized algorithms shown in Fig. 1.5 (b), though it depends on the decision made by the schedulers located at the base station and aggregators, which execute the applications in parallel without any cooperation. The distributed task assignment shown in Fig. 1.5 (c) has overcome the above issues, yet it is subject to non-virtualized sensor nodes that can execute only one task at a time. By contrast, in a distributed virtualized IoT scheme shown in Fig. 1.5 (d), multiple virtual sensors can be instantiated on top of the virtualizable sensor nodes, allowing for simultaneous execution of multiple sensing tasks.

1.4. Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 discusses the motivating use case scenarios, and requirements, and provides a critical review of the state-of-the-art. We organize the thesis based on the algorithmic contributions. Accordingly, Chapter 3 presents the energy efficient dynamic task assignment in virtualized wireless sensor networks. Chapter 4 presents the proposed solution for energy efficient dynamic network embedding in virtualized wireless sensor networks. In Chapter 5, we present energy efficient distributed task assignment in virtualized Internet of Things. Finally, we conclude this manuscript in Chapter 6 and provide future directions for this research work.

Chapter 2

Related Work

In this chapter, we first present motivating use case scenarios. Then, we present a set of requirements derived from the scenarios. Finally, we review the state-of-the-art in light of these requirements.

2.1. Motivating Use Case Scenarios

There are several motivating use case scenarios that can be considered. As an illustration of the problems at hand, let us consider a disaster management system in an earthquake zone close to the ocean where there is a possibility of destructive tsunamis. Sensor nodes with multiple sensing capabilities (sensors with multiple functions) could be deployed, both on the sea and in the coastal city next to it. These sensors could measure water pressure to detect seismic waves [31], as well as the changes in the gravity for earthquake detection, using gravity sensors [32]. In addition, temperature and air humidity sensors could be used to detect possible fire incidents after an earthquake. Some examples of applications that may assign tasks to sensors in such an environment are fire detection, earthquake prediction, and earthquake detection.

Another example could be a smart heritage monitoring system, where sensor nodes with multiple sensing capabilities (e.g., temperature and humidity) are deployed. Some examples of applications that may assign tasks to sensors in such an environment are fire contour, humidity expansion map, and heating and ventilation air conditioning fault detection. The sensing tasks of such applications may require to use same physical sensor nodes to address the applications' requirements. Thus, the deployed WSN substrate network is required to execute the sensing tasks of such applications simultaneously, while exchanging messages between the sensor nodes.

2.2. Requirements

Considering the above motivating use case scenarios, different requirements are derived. We categorize the requirements into four main categories: general requirements that apply to all contributions, requirements specific to task assignment in virtualized WSNs, requirements specific to network embedding in virtualized WSNs, and requirements specific to distributed task assignment virtualized IoT.

2.2.1 General Requirements

We define the following requirements that are applied to all contributions.

i. It is critical to avoid redundant WSN deployment. Therefore, the system should support WSN virtualization to allow the co-existence of both physical and virtual sensors. This leads to the first general requirement that is concurrent execution of tasks.

ii. Considering that the sensor nodes are battery powered and need to be active and functional for a long period of time, thus, the proposed model should satisfy the energy efficiency requirement. In this thesis, we assume that all of the sensors are battery-powered with no access to any powerful source of energy.

iii. The third requirement that can be applied to all contributions is dynamicity. The dynamicity can be considered from both application and substrate network perspectives. Any new application deployed in the environment should be able to assign tasks, and any previously deployed application should also be able to assign new tasks if necessary. Therefore, tasks cannot be assigned statically in such an environment where they are known in advance. In other words, the applications' tasks may arrive at random time instants (rather than being periodic or predictable). On the other hand, from a substrate network perspective, a virtual network needs to be created dynamically as per application requests. This implies that the substrate network should not be pre-set/-configured in advance.

2.2.2 Requirements Specific to Task Assignment

i. The system should be scalable to cope with a large number of tasks. Hence, the first requirement is to satisfy scalability that is defined in terms of the number of tasks assigned to the sensor nodes.

ii. Different tasks may hold different deadlines. This is critical as it specifies the priorities of the tasks. Thus, the second requirement is task deadline which must be satisfied to avoid a task to be dropped. The task deadline is defined as the maximum time a task can wait in a queue upon its arrival and before it can be executed.

2.2.3 Requirements Specific to Network Embedding

i. Given that the IoT network operates on wireless communications with limited bandwidth, it is important to satisfy the application's bandwidth requirements.

ii. Similarly, it is also necessary to meet the given end-to-end (E2E) latency constraints. Essentially, each application may have a specific latency requirement from the source to the destination node which needs to be met.

2.2.4 Requirements Specific to Distributed Task Assignment

i. The IoT networks are intrinsically distributed with limited capacity and computing capabilities. Moreover, it is highly unlikely to always have a resource-rich (in terms of energy and computing) node available in the network to run the computation-intensive algorithms. Therefore, this leads to satisfying the message exchange requirements in which the sensor nodes do require to communicate with each other to address the requirements of the application through in-network processing. Therefore, it is necessary to have a distributed model for task assignment which needs to be satisfied. In addition, task deadline, bandwidth, and E2E latency are other requirements specific to distributed task assignment which were defined earlier in Sections 2.2.2 and 2.2.3, respectively.

2.3. Related Work

This section presents the state-of-the-art for this research that is presented in three parts. The first part discusses task assignment approaches in WSNs. The second part reviews the works focused on network embedding in WSNs. In the third part, the algorithmic solutions for distributed task assignment in IoT are reviewed.

2.3.1 Task Assignment in WSNs

In this sub-section, we review the algorithmic solutions for task assignment in WSNs. It covers three main categories. The first category is focused on single-objective-based solutions, while the second category discusses multi-objective-based approaches. In the third category, the algorithmic solutions that consider the static aspect of task assignment in addition to the dynamic aspects are discussed.

A) Single-Objective-based Solutions

The single objective targeted by all the works discussed in this sub-section is the minimization of energy consumption. However, none of these works meets all our requirements. A Software-Defined Sensor Network (SDSN) is proposed in [33]. It was formulated as a Mixed Integer with Quadratic Constraints Programming (MIQP) problem,

which was then re-formulated to a Mixed Integer Linear Programming (MILP) problem. Although the authors considered software-defined sensor nodes, allowing multiple tasks to run over the same sensor node, the overhead that is induced by sensor node softwarization is ignored. This is important, as it significantly affects decision-making when assigning tasks. In addition, they did not address the scalability requirement.

Price-based adaptive task allocation for WSNs is proposed in [34]. The authors studied the problem of fair energy balance among sensor nodes while considering the task deadlines. The authors extended their work in [35] by proposing a combinatorial auction-based task assignment algorithm. However, neither of these works considers the node-level virtualization and scalability requirements. The work in [36] studied the problem of task assignment, in which the sensor nodes may be shared among multiple similar tasks. Refs. [37–40] introduced the concept of a sensor-cloud where an instance of the sensor is created in the cloud and referred to as a virtual sensor and thereby shared among multiple applications. However, none of these works meet the node-level virtualization, scalability, and task deadline requirements.

The authors of [41–43] considered creating multiple WSNs on top of the deployed WSN using different techniques. For example, in [41] they used the concept of the overlay. The authors modeled the applications as a Directed Acyclic Graph (DAG) where the nodes represent the tasks and the edges represent the dependencies between tasks. The authors of [42] used network slicing and proposed a heuristic to solve the problem of task assignment. In [43], researchers proposed a multi-agent clustering model using Adaptive Distributed Artificial Intelligence (ADAI) for cluster formation and Adaptive Particle Swarm Optimization to solve the resource allocation problem. However, none of these works consider sensor node-level virtualization or task deadlines. Furthermore, the authors of [44] considered a WSN with amplified-and-forward (AF) relay and energy harvesting (EH) sensor nodes. They formulated the problem as Markov Decision Problem (MDP) and used reinforcement learning (RL) to find an approximate transmission strategy. Although energy efficiency was considered in their proposed model, the concurrent execution of tasks was not satisfied. Moreover, Ref. [44] did not discuss task deadlines or scalability.

B) Multi-Objective-based Solutions

Besides single-objective-based solutions, several works have solved the problem by considering multi-objectives in their approaches. The work in [45] aimed at maximizing the profit while considering energy consumption. The authors proposed an analytical model and a distributed heuristic for task assignment in WSNs with energy harvesting capabilities. However, they did not cover node-level virtualization, scalability, or task deadline requirements. In [46], the authors allowed for the creation of multiple Virtual Sensor Networks (VSNs) on top of the deployed physical WSN, aiming to maximize both the residual energy of the network and the residual energy of the sensor node with the lowest available energy. They also considered the weighted sum of the first two objectives. Although virtualization is considered, the virtualization overhead is ignored. This is critical as the virtualization overhead may impact the

decision-making process. Moreover, the scalability and task deadline requirements are not met.

In [47], the objective was to minimize energy consumption and satisfy the delay-sensitive applications' requirements. The authors showed that their system is scalable. However, their work does not satisfy the node-level virtualization requirement. An algorithm for resource allocation in WSNs was proposed in [48], in which the sensor nodes are pooled. Virtual sensor networks could then be built through network slicing. These researchers aimed at minimizing the overall energy consumption of the network while considering the sensors' energy harvesting. However, they did not address the scalability, or task deadline requirements. In addition, the virtualization overhead was also ignored. These factors have enormous impacts on assignment strategies. An energy-efficient task offloading (EETO) was proposed in [49], which aimed at minimizing energy by jointly optimizing task scheduling and offloading in real-time IoT applications. The authors of [50] proposed a multi-hop cooperative caching to improve both energy efficiency and data fetching delay, where sensor nodes frequently and periodically execute sensing tasks and generate related data. However, none of these works considered node-level virtualization and scalability requirements. Moreover, our task deadline requirements were not satisfied by Ref. [50].

C) Solutions with Static and Dynamic Aspects

There are some works that focused solely on the static aspects of the problem (e.g., [51–55]). We do not discuss these works in detail here, because they do not meet the dynamicity requirement, which is key to our work. Instead, we concisely review those works that considered node-level virtualization. The authors of [53] aimed at maximizing the revenue while minimizing the number of active sensor nodes. However, as discussed, this work only solved the problem in its static settings. In addition, Ref. [53] ignored the virtualization overhead and did not discuss scalability. That work also did not address the task deadline requirement and instead focused on sensor nodes' storage and bandwidth requirements. The authors of [54] tackled the problem of task assignment in static settings while considering node-level virtualization. They proposed a non-linear weight discrete particle swarm optimization (NWDPSO) to solve the problem. Similarly, the virtualization overhead factor was ignored and there was no discussion on scalability or task deadline requirements. The authors of [55] proposed both short- and long-term solutions for the task assignment problem while considering node-level virtualization. While they considered virtualization overhead and task deadlines, they only solved the problem in its static settings and targeted maximizing the revenue for the operator as their main objective. In addition to the above-mentioned works, there are some works that do not meet the scalability requirement [56, 57], while others do not meet the energy efficiency requirement [58]. Some do not meet the task deadline requirement [59, 60].

There are several studies that consider the dynamicity aspect of the problem in addition to the static aspects. In [61], the static setting of the problem is analyzed first. The task assignment problem is formulated to maximize the profit, where the profit is the amount of task demands satisfied. The authors then considered dynamic settings

with the same objectives as their static settings and proposed an energy-aware algorithm for extending the network's lifetime. Similarly, in [62] the problem is first studied for resource conservation. The authors defined a budget for the maximum resources that can be utilized by an application to avoid overtaxing an application's resources. The authors in [62] also considered dynamic settings in addition to their static settings. They defined the budget for the dynamic settings in terms of the battery life of the sensor nodes. This was accomplished by evaluating the trade-offs between the expected profit and the value of the task assignments. While these works evaluated the energy consumption, the other requirements such as tasks' deadlines, scalability, and node-level virtualization are not covered by their model.

2.3.2 Network Embedding in WSNs

This sub-section reviews the algorithmic solutions for network embedding in WSNs. At first, we review the virtual network embedding solutions in conventional (wired) networks. Then, we cover the works that focused on virtual network embedding in WSNs.

A) VNE in Conventional Networks

The works that are discussed in this subsection satisfy the energy efficiency and bandwidth requirements. Yet, some works solve the problem in static settings only. For example, the authors of [63] proposed their so-called VNE-Energy-Aware (VNE-EA) method, aiming at minimizing energy by allocating virtual network requests to the reduced group of physical network infrastructure using Mixed Integer Programming (MIP). Although they considered energy efficiency and bandwidth requirements, they did not meet requirements on mixed virtualizable and non-virtualizable sensor nodes, dynamicity, and E2E latency. Ref. [64] aimed at minimizing energy consumption considering the minimal product of energy cost per CPU that is allocated from virtual nodes to substrate nodes in the physical network. In [65], the authors proposed Energy-Aware Virtual Optical Network Embedding (EA-VONE) in Elastic Optical Networks (EONs) and modeled the problem using Integer Linear Programming (ILP). The main objective was to switch off as many active resources (nodes and links) as possible. They solved the problem in the static mode, where the assumption was that all virtual network requests are known in advance. Then, they released this assumption and tackled the problem in the dynamic mode. Ref. [66] applied Bacterial Foraging Optimization (BFO) and proposed Energy-driven BFO (EBFO) algorithm for efficient resource utilization with energy consumption at a minimal cost. In this work, they followed the information interaction between bacteria in the foraging stage to design an energy-efficient network embedding algorithm. Ref. [67] proposed a Greedy Randomized Adaptive Search Procedure (GRASP) algorithm, aiming at minimizing energy by switching off the unused nodes and links. However, while Refs. [64–67] cover the energy efficiency, dynamicity, and bandwidth requirements, none of them address the E2E latency, nor did they cover the mixed substrate network requirements.

Alternatively, some works focused on objective functions other than energy efficiency in VNE, e.g., [68–78]. We do not discuss them in detail as they did not cover energy efficiency, which is the main focus of this work. For example, Refs. [68–70] focused on cost minimization of VNE in conventional networks, while others [71–74] considered the acceptance rate ratio. Ref. [75] studied the VNE problem in conventional networks considering multiple objectives, namely, maximizing profit and minimizing cost as the first objective, and minimizing energy and maximizing the profit as the second objective. Other works [76, 77] tackled the problem of maximizing the profit, whereas [78] solved the problem of minimizing the delay by partitioning the network into multiple smaller networks.

B) VNE in WSNs

There are several works that aim to solve the problem of virtual network embedding in WSNs. Some works solely focused on the static settings of the problem, where the application requests are known in advance and the time factor is eliminated. These works do not meet our dynamicity requirement. The authors of [79, 80] proposed a Mixed Integer Linear Programming (MILP)-based framework for service embedding in IoT, aiming at minimizing the energy consumption while accounting for the traffic demands in a static setting. However, none of the Refs. [79, 80] addressed the latency, dynamicity, or bandwidth requirements. In addition, they only considered the substrate network with physical sensors. There are works [81] that solved the problem considering multiple objectives in dynamic settings. Ref. [81] proposed an MILP framework to solve the multi-objective problem of joint optimization of latency, processing, and network energy consumption while considering resiliency. As for resiliency, they proposed an approach in which the traffic from the source node to the destination node is split into two paths. Each path handles only half of the traffic. In case of a failure in any of the paths during transmission, the traffic will be re-routed through the other path. Although Ref. [81] considered energy efficiency, dynamicity, and latency, they did not satisfy the bandwidth requirement. In addition, they did not cover the virtualization requirement, which is to allow the co-existence of both physical and virtual sensors in the substrate network.

Alternatively, some works have applied the concept of network slicing in solving the resource allocation problem in WSNs [42, 48, 54], while others tackled the problem using clustering [43]. The authors of [42] solved the joint problem of admission control and resource allocation to dynamically optimize resource utilization in terms of energy efficiency. They proposed their so-called Application Admission Control-Sensor Network Slicing framework (AAC-SNS), where it is first decided whether to admit an application. Then, multiple applications are allowed to share the deployed infrastructure concurrently using network slicing. In [54], the authors proposed a Software-Defined Wireless Sensor Network (SDWSN) framework that applies network slicing to have multiple applications co-exist over the same deployed WSNs. Then, they proposed Non-linear Weight Discrete Particle Swarm (NWDPSO) algorithm to map the virtual networks (slices) to the physical substrate network. In [54], sensor nodes can be part of different coalitions at the same time to address multiple tasks, though they are shared among the tasks that require a particular sensing capability.

Although Refs. [42,54] addressed the energy efficiency and bandwidth requirements, they did not consider the latency. Moreover, substrate network with mixed physical and virtual sensors is not addressed by [42], while the dynamicity requirement is not met in [54]. In [48], the authors proposed a strategy in which the sensor nodes are virtualized and then, virtual sensor networks are created through network slicing for resource allocation. They aimed at minimizing the energy consumption of sensing and transmission while accounting for the energy gained through wireless channels (energy harvesting). However, Ref. [48] did not address our latency requirement. The authors of [43] proposed a multi-agent clustering technique using Adaptive Distributed Artificial Intelligence (ADAI) for resource allocation in WSNs. They used DAI for inter-cluster communications while for the intra-communications, Adaptive Particle Swarm Optimization (APSO) was applied. Although Ref. [43] considered energy efficiency and dynamicity, the other requirements are not met.

Moreover, there are some works that focused on other objectives without touching on energy efficiency requirements. For example, Refs. [82–85] worked on minimizing the latency. Others proposed multi-objective solutions, Ref. [86] considered latency and fault tolerance in virtualized wireless sensor networks, while Ref. [55] considered reward maximization for the operators calculated by the difference between the revenue gained from the buyers and the cost paid to the suppliers. In [87], the authors aimed at maximizing the profit and minimizing the resource usage (link capacity). Alternatively, Refs. [88,89] focused on minimizing the cost (load on links) and at the same time maximizing the acceptance rate. The concept of sensor-cloud was introduced in [22,23] in which an instance of the sensor node is created in the cloud and referred to as a virtual sensor. Then, the collected data by these instances are shared among multiple applications (data sharing). Refs. [1,2,33] solved the problem of energy efficiency in task assignment considering node-level virtualization only.

In addition, Refs. [90–92] introduced the concept of Virtual Network Function (VNF) placement in IoT networks. In [90], the authors addressed the problem of VNF placement across geographically distributed clouds considering application requirements and locations, while Ref. [91] studied the problem of VNF placement on three pre-defined layers: (i) resource-constrained IoT nodes layer to host VNFs such as data aggregations and relay, (ii) IoT gateway layer, which has higher processing and storage capabilities, and (iii) cloud layer that consists of powerful nodes in terms of computing, storage, and networking, and therefore the simple and complex VNFs can be deployed on cloud nodes if needed. In [92], the authors investigated the problem of VNF placement in the cloud and Multi-Access Edge Computing (MEC) infrastructure, aiming at minimizing the end-to-end communication delay and reduction of operational and capital expenses. However, all of these works [90–92] differ from ours in various aspects. Specifically, they do not consider virtualized IoT infrastructure, thus not fully exploiting the capabilities of sensor nodes. In addition, in these works, the VNFs receive the sensing data from the IoT infrastructure, and then these VNFs are shared among multiple applications.

2.3.3 Distributed Task Assignment in IoT

This sub-section reviews the algorithmic solutions for distributed task assignments in IoT. Initially, we review the centralized algorithmic approaches followed by the decentralized algorithmic works. Finally, we cover the distributed algorithmic solutions in task assignments in IoT.

A) Centralized Task Assignment Approaches

Several works solved the problem of task assignment in IoT in a centralized manner. All of the works discussed in this subsection satisfy our energy efficiency and dynamicity requirements. In all of these works, it is assumed that there exists a centralized entity (e.g., scheduler) that can run the algorithms and manage the IoT network. The authors of [48] studied the problem of resource allocation in WSNs aiming at minimizing the energy consumption of sensing and transmission of the sensors. Assuming that all sensors are virtualized, virtual sensor networks are built through network slicing. However, they did not consider task deadlines, E2E latency, and bandwidth requirements, nor did they consider the virtualization overhead. In [2], the authors solved the problem of task assignment in virtualized WSNs in a dynamic manner while considering energy efficiency and task deadlines. However, they did not cover the other requirements. The authors in [81, 93] investigated the problem of virtual network embedding in IoT using a multi-objective model. In [93], the authors focused on minimizing the energy consumption, and average traffic latency, aiming to realize a trade-off between these objectives. In [81], a mixed integer linear programming (MILP) framework was proposed to solve the joint optimization of latency, processing, and network energy consumption while considering resiliency. Although Refs. [81, 93] satisfied the energy efficiency, dynamicity, and latency requirements, the other requirements are not met.

Some works focused on the static version of the problem, e.g., [1, 3, 53–55]. We do not discuss them in detail as they do not satisfy our dynamicity requirement. The authors of [22, 23] introduced the concept of sensor-cloud wherein an instance of the sensors is instantiated in virtual machines in the cloud that are referred to as virtual sensors. These instances collect the data from the sensors and share them among multiple identical applications (data sharing). However, these works are different from ours as they do not fully exploit the capabilities of the sensors. Thus, they do not satisfy the requirement of concurrent execution of sensing tasks over the same WSN.

B) Decentralized Task Assignment Approaches

There are some papers that attempted to solve the task assignment problem in a decentralized manner. The works discussed in this subsection satisfy our dynamicity requirement. Some works [94, 95] proposed both centralized and decentralized approaches. In [94] the authors investigated the problem of efficient transmission of information and power from the access-points to the IoT devices while satisfying the bandwidth QoS requirements. Initially, they

solved the problem in a centralized manner and static environment by using the Lyapunov optimization. Then, they proposed a decentralized approach in which the access point determines the resource scheduling process in a centralized manner, while the IoT devices schedule their tasks based on harvested energy from the signals and the QoS criteria in a decentralized manner. In [95], the authors first studied the problem of task scheduling in a static environment and a centralized manner. They modeled the problem as Binary Integer Linear Programming (BILP) aiming at maximizing the network lifetime while accounting for task deadlines and bandwidth requirements. Also, they solved the problem in a dynamic and decentralized way by breaking the centralized approach into multiple small-sized sub-problems using Dantzig Wolf decomposition optimization. Therefore, each sensor executes one task (i.e., sub-problem) independently. Although these works [94, 95] considered energy efficiency and bandwidth consumption, they do not satisfy the other requirements.

Reference [43] focused on task assignment in WSNs, aiming at minimizing energy consumption. They modeled the system as a multi-agent clustering WSN and proposed Adaptive Distributed Artificial Intelligence (ADAI) technique to solve the problem in a decentralized manner. In their work, the clusters are pre-set and cluster heads communicate with each other to decide which cluster may execute an application. Then, the clusters run the application independently. They applied Distributed Artificial Intelligence (DAI) for inter-cluster communications and Adaptive Particle Swarm Optimization (APSO) for intra-cluster communications. While they covered energy efficiency and dynamicity, the other requirements are not discussed. Moreover, other works focused on other objectives e.g. minimizing latency. The authors in [96] proposed a greedy algorithm for task scheduling considering task deadlines, E2E latency, and bandwidth requirements. Their algorithm consists of task dispatching and task scheduling steps. In task dispatching the end device may offload the task to a nearby edge server or a remote cloud to execute the task. Task scheduling includes the decision-making on the start time of task execution considering the defined requirements. In Ref. [96], the dynamicity, task deadline, E2E latency, and bandwidth requirements are satisfied, but the other requirements related to energy efficiency, message exchanges between sensors, and concurrent task execution are not met.

C) Distributed Task Assignment Approaches

Despite the centralized and decentralized approaches, there are works that solved the task assignment problem in IoT in a distributed manner. Reference [97] designed a 6G-enabled massive IoT architecture to solve the dynamic resource allocation problem. Accordingly, AI-driven collaborative dynamic resource allocation (ACDRA) algorithm was proposed based on a nested neural network and further combined with Markov Decision Process (MDP) to address resource allocation optimization in real-time. The tasks were executed by devices collaboratively. They considered minimization of delay as their objective function followed by E2E latency as a QoS metric. However, energy efficiency, task deadline, bandwidth, and concurrent execution of tasks were not discussed in this work. The authors of [98] proposed a scheme for minimization of energy and delay by sharing the IoT resources with nearby nodes as per

application requirements. They considered a visual camera monitoring system as their illustrative scenario, where each camera may execute a specific function. The cooperation of the camera nodes completes the application requirements. In their proposed scheme, each node determines which task it executes considering the latency QoS requirements, while maximizing the resource utilization. Although the virtualization, energy efficiency, dynamicity, latency, and message exchange requirements are satisfied, the other requirements such as bandwidth and task deadline constraints are not met.

Reference [99] proposed the so-called Distributed Optimal On-line Task Allocation (DOOTA) algorithm to balance the workload among the sensors efficiently to maximize network lifetime. In their model, the network is partitioned into multiple clusters, and the slave nodes complete the task in collaboration with the cluster-head. More specifically, the salve node sends the raw data to the cluster-head with/out pre-processing it. Then, the cluster-head may further process it to accomplish the task accordingly. The authors of [100] aimed at minimizing the cost in terms of the required energy for transmission, and proposed a greedy algorithm to solve the resource allocation problem. They considered a set of cameras and the cameras may join the coalition to address an application requirement based on their utility function which is the energy required for communication and processing. Although Refs. [99, 100] considered energy efficiency, dynamicity, and sensors message exchange, they did not cover any other requirements. Further, some other works aim to solve the problem in conventional wired networks, e.g., [101, 102], but we do not discuss them in detail as they are different from ours, mainly because they do not address the limitations of wireless networks. In addition, there exist some other works that solved the problem in a distributed manner, e.g., [103, 104], though they differ from ours as their primary focus was on mobile crowdsensing [103] and mobile offloading [104]. These works are heavily dependent on mobility and the users that may join/leave the network over time under certain constraints, e.g., location and mobility patterns.

2.4. Conclusions

In this chapter, we first presented potential motivating use case scenarios, from which we derived a set of algorithmic requirements. After that, we surveyed the related work. Table 2.1 provides a summary of the reviewed algorithmic papers. It should be noted that the following acronyms are used in the table 2.1: Concurrent Execution of Tasks (Con. Exe. Tas.), Energy Efficiency (En. Eff.), Dynamicity (Dynam.), Scalability (Scalab.), Task Deadline (Task DL.), Bandwidth (Bandw.), E2E Latency (E2E Lat.), and Message Exchange (Msg. Exch.). For each paper, we show the requirements which are met and the ones which are not met. As it can be seen, none of the reviewed works satisfy all our requirements.

Table 2.1: Related work evaluation.
(met ✓, not met ✗)

Requirements / Related Works	Con. Exe. Tas.	En. Eff.	Dynam.	Scalab.	Task DL.	Bandw.	E2E Lat.	Msg. Exch.
Task Assignment in Virtualized WSNs								
Zeng et al. [33]	✓	✓	✓	✗	✗			
Edalat et al. [34]	✗	✓	✗	✗	✓			
Edalat et al. [35]	✗	✓	✓	✗	✗			
Le et al. [36]	✗	✓	✓	✗	✗			
Zhao and Zhao [44]	✗	✓	✓	✗	✗			
Porta et al. [45]	✗	✓	✓	✗	✗			
Delgado et al. [46]	✗	✓	✓	✗	✗			
Santos et al. [47]	✗	✓	✓	✓	✓			
Hazra et al. [49]	✗	✓	✓	✗	✓			
Yang and Song, [50]	✗	✓	✓	✗	✗			
Rowaihy et al. [61]	✗	✓	✓	✗	✓			
Johnson et al. [62]	✗	✓	✓	✗	✗			
Network Embedding in Virtualized WSNs								
Delgado et al. [42]	✗	✓	✓			✓	✗	
Li and Zhong [48]	✓	✓	✓			✓	✗	
Chen et al. [54]	✓	✓	✗			✓	✗	
Botero et al. [63]	✗	✓	✗			✓	✗	
Chen et al. [64]	✗	✓	✓			✓	✗	
Zhu et al. [65]	✗	✓	✓			✓	✗	
Xu et al. [66]	✗	✓	✓			✓	✗	
Lira et al. [67]	✗	✓	✓			✓	✗	
Al-Shammari et al. [79]	✗	✓	✗			✗	✗	
Al-Shammari et al. [80]	✗	✓	✗			✗	✗	
Al-Shammari et al. [81]	✗	✓	✓			✗	✓	
Distributed Task Assignment in Virtualized IoT								
Mukherjee et al. [43]	✗	✓	✓		✗	✗	✗	✗
Zhong et al. [55]	✓	✗	✗		✗	✓	✓	✗
Al-Shammari et al. [93]	✗	✓	✓		✗	✗	✓	✗
Lee and Lee [94]	✗	✓	✓		✗	✓	✗	✗
Yu et al. [95]	✗	✓	✓		✓	✓	✗	✗
Meng et al. [96]	✗	✗	✓		✓	✓	✓	✗
Lin et al. [97]	✗	✗	✓		✗	✗	✓	✓
Avasalcai et al. [98]	✓	✓	✓		✗	✗	✓	✓
Yu et al. [99]	✗	✓	✓		✗	✗	✗	✓
SanMiguel and Cavallaro [100]	✗	✓	✓		✗	✗	✗	✓

Chapter 3

Energy Efficient Dynamic Task Assignment in Virtualized Wireless Sensor Networks [1]

3.1. Introduction

Wireless sensor networks as a key enabler technology of the Internet of Things are widely being used in various application domains [8]. Energy consumption has always been a bottleneck in these networks. This is mainly because, in traditional WSNs, the applications are embedded into the sensor nodes, preventing them from being re-used and possibly shared by other applications. Therefore, it results in inefficient resource utilization and redundant WSN deployment, leading to increased maintenance, deployment, and energy costs [6]. To cope with this issue, WSN virtualization is introduced. WSN Virtualization allows multiple applications' sensing tasks to be executed over the same WSN infrastructure concurrently [6]. However, there is additional energy and delay associated with the virtualization in terms of the virtualization overhead [11, 12]. This is critical considering that the sensor nodes are constrained devices with limited available energy. Also, an increase in delay may have drastic impacts on delay-sensitive applications. Therefore, designing and modeling a mechanism for the efficient allocation of these virtualized resources to the applications' sensing tasks becomes a challenge. In this chapter, we tackle these issues.

The rest of the chapter is organized as follows. At first, we present a high-level system model and problem definition. Next, we put forward a mathematical formulation, followed by the proposed heuristic that considers the virtualized WSN infrastructure and the given QoS (i.e. task deadline) requirement. Then, we present the performance evaluation, and finally, We conclude the chapter in the last subsection.

[1] This chapter is based on two published papers: [1] V. M. Raee, D. Naboulsi, and R. Glitho, "Energy efficient task assignment in virtualized wireless sensor networks," in Proc. IEEE Symposium on Computers and Communications (ISCC), pp. 00976–00979, 2018, and [2] V. M. Raee, A. Ebrahimzadeh, R. H. Glitho, and H. Elbiaze, "Ensuring energy efficiency when dynamically assigning tasks in virtualized wireless sensor networks," IEEE Transactions on Green Communications and Networking, 2021.

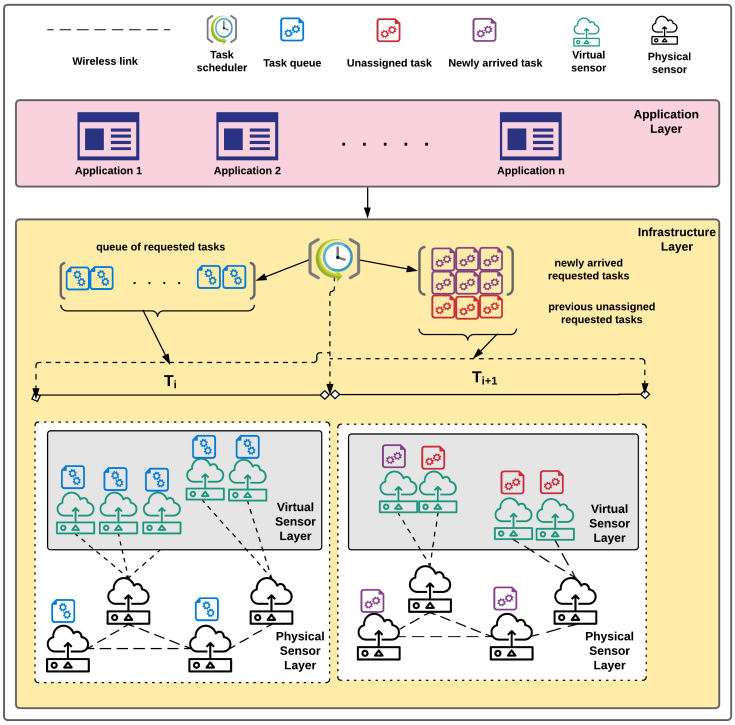


Figure 3.1: High-level system model.

3.2. Dynamic Sensing Task Assignment Problem

This section presents our system model, wherein the problem definition is elaborated, followed by the problem formulation. Our developed formulation leads to an ILP formulation of the problem, given that we use a linear objective function along with linear constraints with integer decision variables, to be explained in technically great detail in Section 3.2.1.

A high-level view of our system model is depicted in Fig. 3.1, where we consider a WSN with node-level virtualization capability. The WSN includes a set of deployed sensor nodes that execute sensing tasks assigned to them. The applications (e.g., fire detection) generate the set of sensing tasks (e.g., temperature and humidity). The task scheduler receives the sensing task requests from the application layer (see Fig. 3.1). The scheduler is responsible for managing the sensing tasks and assigning them to the sensor nodes. Once the tasks are executed by the sensor nodes, the results of the readings will be sent back to the scheduler to address the applications' requests. The system includes a set of time slots during which the tasks arrive in batches and are queued at the beginning of each time slot. The number of time slots has been selected such that a statistical stability is achieved. The scheduler runs the assignment algorithm and manages the application sensing tasks accordingly, assigning them to physical and virtual sensor nodes. If there are application sensing tasks that cannot be assigned at their arrival time slot, they will be scheduled for the next time slot together with the queue of the new incoming tasks.

We aim at minimizing the WSN's overall energy consumption while respecting the deadlines. Once a sensing task is assigned to a sensor node (physical or virtual), it will be executed immediately. As it was discussed earlier, when a task is assigned to a physical sensor the virtualization functionality of the physical sensor is disabled.

Given that time is discretized as $\mathcal{T} = \{T_n, n \in \mathbb{N}\}$, where time slot $T_n \in \mathcal{T}$ corresponds to the total task assignment time frame and $\mathcal{T}$ is the total observation time, we model the overall energy consumption of the virtualized WSN considering the given constraints i.e., the sensing task deadline, set by the application. We consider a virtualized WSN composed of a set $\mathcal{I} = \{1, 2, ..., k\}$ of sensor nodes. Each sensor node $i \in \mathcal{I}$ is placed in a location with multiple functions, allowing it to execute specific tasks within its proximity. Each application sensing task (e.g., temperature, humidity) requires reading from a sensor node [105]. This is different from computing tasks, where additional processing may be required by servers or the IoT nodes prior to transmission of the results. In this work, we assume that the tasks are uniform and they require similar amount of energy for transmission and have similar transmission delay. We define a matrix L accordingly, such that $L_{i,j} = 1$ if task $j \in \mathcal{J}$ is in the proximity of the sensor node $i \in \mathcal{I}$. Otherwise $L_{i,j} = 0$. An application sensing task is considered to be in proximity of the sensor node if it is within the sensor node's sensing range.

Tasks are generated by applications according to a general process and then served by the scheduler in batches at the beginning of each time slot T_n. Let T_n be the n^{th} inter-arrival time of the group of tasks and α_n be the size of the n^{th} group of tasks. Variables T_n and α_n are supposed to be independent (i.e. their distributions do not depend on n). Let $a_j^{T_n}$ be the arrival time of task j at time T_n and $d_j^{T_n}$ its deadline respectively. The task deadline is defined as the time instant by which a task execution result needs to be delivered to the sink node and it is set by the application. In this work, we assume that all the application sensing tasks will execute immediately after being assigned to the physical and virtual sensors. Tables 3.1 and 3.2 summarize the main notations and problem inputs, respectively.

3.2.1 Problem Formulation

The problem aims at dynamically assigning sensing tasks to sensor nodes for their execution at the lowest energy consumption with respect to their deadlines. A task can be executed in two ways: i) directly on a Physical Sensor (PS), so that just one task can execute on a sensor node; or ii) on a Virtual Sensor (VS) on top of the sensor node, with the possibility to run more than one task on a single sensor node.

At time T_n, the problem considers all tasks in the WSN that are not yet executed including i) tasks $j \in \mathcal{J}$ arriving at time $a_j^{T_n}$ and ii) tasks arriving at time $a_j^{T_m}$, such that $T_m < T_n$, and are not executed yet. In case (i) the problem makes an assignment decision for a newly arriving task j at T_n. In case (ii), the problem allows an assignment decision to be made for a task j that arrived during a previous time slot T_m ($m < n$) and to assign it to a PS or VS, at time T_n. We formulate the dynamic sensing task assignment problem as an ILP problem and define the decision variables

Table 3.1: General notations of the problem.

Notation	Definition
$\mathcal{I}$	The set of sensor nodes
$\mathcal{J}$	The set of tasks where j represents a single task within set: $j \in \mathcal{J}$
$\mathcal{T}$	Total observation time
T_n	Total task assignment frame time
α_n	The size of each group of tasks at T_n
$a_j^{T_n}$	The arrival of task j at time T_n
$d_j^{T_n}$	The deadline of task j at time T_n
$L_{i,j}$	A matrix specifying that the task j is within the sensing range of the sensor node i
w_{i,i',j,T_n}	Routing energy consumption of the task j that is assigned to sensor node i at time T_n passing through sensor node $i^{'}$
M	The maximum number of virtual sensor nodes over a physical sensor node
h	The number of intermediary sensor nodes along the routing path
$E_{T_n}^i$	The available energy of sensor node i at time T_n
E^{tr}	Forwarding energy consumption
E^{re}	Receiving energy consumption
E^{ov}	Virtual sensor node creation energy consumption
D^{pr}	Transmission delay
D^{ov}	Virtual sensor node creation delay

Table 3.2: Problem inputs.

Input	Definition
PS	A physical sensor node
VS	A virtual sensor node
E_{T_n}	Total energy of assignment at time T_n
$E_{T_n}^P$	Energy consumption of a physical sensor node i at time T_n
$E_{T_n}^V$	Energy consumption of a virtual sensor node i at time T_n
$E_{T_n}^R$	Energy consumption of routing from sensor node i at time T_n
$D_{T_n}^P$	Delay of a physical sensor node i at time T_n
$D_{T_n}^V$	Delay of a virtual sensor node i at time T_n
$D_{T_n}^R$	Delay of routing from sensor node i at time T_n

accordingly, as follows.

$$x_{i,j,T_n} = \begin{cases} 1, & \text{if task } j \text{ is assigned to sensor } i \text{ at } T_n \text{ in its physical mode,} \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

$$y_{i,T_n} = \begin{cases} 1, & \text{if sensor } i \text{ is virtualized at } T_n, \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

and,

$$z_{i,j,T_n} = \begin{cases} 1, & \text{if task } j \text{ is assigned to sensor } i \text{ at } T_n \text{ in its virtualized mode,} \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

We model the energy consumption and the delay of the WSN in the following. Then, we define the objective function and constraints.

A) Energy Consumption

When solving the problem at time $T_n \in T$, we consider the total energy consumption, which can be divided into three components as follows.

- **PS Energy Consumption ($E^P_{T_n}$)**: We define $E^P_{T_n}$ as the energy consumption resulting from assigning sensing tasks to physical sensors at time T_n as follows:

$$E^P_{T_n} = \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} x_{i,j,T_n} \, L_{i,j} \, E^{tr}, \tag{3.4}$$

where, E^{tr} is the energy consumption for running one sensor node. $E^P_{T_n}$ includes the energy consumption for all task executions on the PSs at time T_n, both the current decisions concerning upcoming task executions on PSs and those that arrive in previous time slots but have not yet been executed.

- **VS Energy Consumption ($E^V_{T_n}$)**: The VS energy consumption at each time T_n represents the energy consumption resulting from assigning sensing tasks on VSs over sensor nodes. It covers the energy consumption for running the physical sensor nodes in addition to a virtualization overhead for each VS. $E^V_{T_n}$ is obtained as follows:

$$E^V_{T_n} = \sum_{i \in \mathcal{I}} y_{i,T_n} \, E^{tr} + \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} z_{i,j,T_n} \, L_{i,j} \, E^{ov}, \tag{3.5}$$

where E^{ov} is the energy consumption overhead incurred by creating a VS over a physical sensor node [11].

- **Routing Energy Consumption** ($E^R_{T_n}$): The routing energy consumption includes the total energy consumed by routing data from sensor nodes that are executing the tasks, as the source, to the sink node as the destination, at time T_n. We assume the shortest path routing protocol is used. For one task execution, all the intermediary sensor nodes between the source and the sink node will consume energy due to the reception and transmission of data. The energy consumption of the network due to routing is obtained as follows:

$$E^R_{T_n} = \sum_{i \in \mathcal{I}} \sum_{i' \in \mathcal{I}} \sum_{j \in \mathcal{J}} w_{i,i',j,T_n} \, h_{i,i'} \, E^{tr\_re}, \tag{3.6}$$

where, E^{tr_re} is the energy consumed for both the transmission and reception of data on one sensor node. For every sensor node i, we define set I_i. This set includes all sensor node's whose path to the sink node crosses sensor node i. $h_{i,i'}$ represents the number of hops between sensor node i and sensor node $i' \in I_i$, and $i \neq i'$. Accordingly, for every $i \in I$, considering $i' \in I_i$ we define:

$$w_{i,i',j,T_n} = x_{i,i',j,T_n} + z_{i,i',j,T_n} \tag{3.7}$$

where $w_{i,i',j,T_n} = 1$ if task j is assigned to sensor node i' at T_n and has to route the packet from i.

- **Total Energy Consumption** (E_{T_n}): The total energy consumed in the WSN at a time T_n is calculated as given by:

$$E_{T_n} = E^P_{T_n} + E^V_{T_n} + E^R_{T_n}. \tag{3.8}$$

B) Delay

When solving the problem at time $T_n \in T$, we consider the total delay factors, which can be divided into three components as follows.

- **PS Delay** ($D^P_{T_n}$): At time T_n, we define $D^P_{T_n}$ as the delay (in milliseconds) resulting from transmission of a task from the sensor nodes in its physical mode, obtained as follows:

$$D^P_{T_n} = \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} x_{i,j,T_n} \, L_{i,j} \, D^{pr}, \tag{3.9}$$

where D^{pr} is the transmission delay of running a task on a sensor node.

- **VS Delay** ($D^V_{T_n}$): The VS delay at each time T_n represents the time (in milliseconds) spent by the sensor node to transmit a task from a the sensor node when it is on its virtualized mode. It covers the delay for running the

physical sensor node in addition to a virtualization overhead for each VS, and is obtained as follows:

$$D^V_{T_n} = \sum_{i \in \mathcal{I}} y_{i,T_n} \, D^{pr} + \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} z_{i,j,T_n} \, L_{i,j} \, D^{ov}, \tag{3.10}$$

where D^{ov} is the delay overhead incurred by creating a VS over a physical sensor node [11, 12].

- **Routing Delay** ($D^R_{T_n}$): The routing delay reflects the total delay (in milliseconds) for data transmission in the network from the source sensor nodes to the sink node at time T_n, considering the shortest path-to-destination routing protocol. There will be a transmission time for all the intermediary nodes between the source and the sink nodes. The network delay cost is obtained as follows:

$$D^R_{T_n} = \sum_{i \in \mathcal{I}} \sum_{i' \in \mathcal{I}} \sum_{j \in \mathcal{J}} w_{i,i',j,T_n} \, h_{i,i'} \, D^{pr}. \tag{3.11}$$

C) Objective Function and Constraints

In our problem, we seek to offer sensing task assignment decisions at the lowest energy consumption while respecting the task deadline's requirements in the constraints. Accordingly, q is the total number of time slots. Thus, we define our objective function as follows:

$$\min \sum_{T_n=0}^{T_n=q} (E^P_{T_n} + E^V_{T_n} + E^R_{T_n}) \tag{3.12}$$

Constraints: We consider the following constraints in our problem. Each task should be assigned to either a PS or a VS at only one time slot, as indicated in Eq. (3.13):

$$\sum_{i \in \mathcal{I}} \sum_{T_n \in [0, d_j^{T_n}]} x_{i,j,T_n} \, L_{i,j} + \sum_{i \in \mathcal{I}} \sum_{T_n \in [0, d_j^{T_n}]} z_{i,j,T_n} \, L_{i,j} = 1 \quad \forall \, j \in \mathcal{J}. \tag{3.13}$$

Moreover, a task cannot be assigned to a PS or a VS before its arrival time, as shown in Eq. (3.14) and Eq. (3.15), respectively:

$$\sum_{i \in \mathcal{I}} \sum_{T_n \in [0, a_j^{T_n})} x_{i,j,T_n} \, L_{i,j} = 0 \quad \forall \, j \in \mathcal{J}, \tag{3.14}$$

$$\sum_{i \in \mathcal{I}} \sum_{T_n \in [0, a_j^{n})} z_{i,j,T_n} \, L_{i,j} = 0 \quad \forall \, j \in \mathcal{J}. \tag{3.15}$$

A sensor node can operate in a physical mode or in a virtualized mode. If a sensor node operates in a virtualized mode, no task should be assigned to it on a PS. However, if a sensor node is operating in a physical mode, it can support a

maximum of one task on its PS. We enforce these conditions using Eq. (3.16):

$$\sum_{j \in \mathcal{J}} x_{i,j,T_n} L_{i,j} \leq 1 - y_{i,T_n} \quad \forall\, i \in \mathcal{I},\ T_n \in \mathcal{T}. \tag{3.16}$$

Further, a sensor node can support a maximum number of VSs, referred to as M. We enforce this condition with Eq. (3.17):

$$\sum_{j \in \mathcal{J}} z_{i,j,T_n} L_{i,j} \leq M y_{i,T_n} \quad \forall\, i \in \mathcal{I},\ T_n \in \mathcal{T}. \tag{3.17}$$

A task should not be assigned to a sensor node at a certain time if the sensor node does not have enough energy. This condition is specified by Eqs. (3.18) and (3.19):

$$x_{i,j,T_n} E^{tr} \leq E^i_{T_n} \quad \forall\, i \in \mathcal{I},\ j \in \mathcal{J},\ T_n \in \mathcal{T}, \tag{3.18}$$

$$y_{i,T_n} E^{tr} + \sum_{j \in \mathcal{J}} z_{i,j,T_n} E^{ov} \leq E^i_{T_n} \quad \forall\, i \in \mathcal{I},\ T_n \in \mathcal{T}, \tag{3.19}$$

where $E^i_{T_n}$ refers to the available energy of sensor node i at time slot T_n, which is obtained using a recursive model, as follows:

$$E^i_{T_n} = E^i_{T_{n-1}} - E^P_{i,T_n} - E^V_{i,T_n} - E^R_{i,T_n}, \tag{3.20}$$

where $E^i_{T_{n-1}}$ represents the available energy on sensor node i, at time slot T_{n-1}. For every task j, it should be ensured that it will be executed by the sensor node i before the task's deadline. This constraint is enforced by Eq. (3.21) and Eq. (3.22):

$$\sum_{i \in \mathcal{I}} \sum_{T_n \in [a_j^{T_n}, d_j^{T_n}]} x_{i,j,T_n} L_{i,j} + \sum_{i \in \mathcal{I}} \sum_{T_n \in [a_j^{T_n}, d_j^{T_n}]} z_{i,j,T_n} L_{i,j} = 1 \quad \forall\, j \in \mathcal{J}. \tag{3.21}$$

$$\sum_{T_n \in \mathcal{T}} (D^P_{T_n} + D^V_{T_n} + D^R_{T_n}) \leq d_j^{T_n} \quad \forall\, i \in \mathcal{I}, j \in \mathcal{J}. \tag{3.22}$$

3.2.2 Problem Analysis

Theorem 1. *The dynamic application sensing task assignment problem in Virtualized WSNs is an NP-hard problem and is at least as hard to approximate as a Dynamic Generalized Assignment Problem (DGAP).*

Proof. The problem can be proven to be NP-hard by a reduction to show that our application task assignment problem in virtualized WSNs can be expressed as a known NP-hard problem. We use a Dynamic Generalized Assignment Problem (DGAP) that is known to be an NP-hard problem [106] as our reference problem. The proof is based on mapping our problem to the DGAP, thereby showing that they are identical.

The Dynamic Generalized Assignment Problem (DGAP) is known as NP-hard in combinatorial optimization problems [106]. There is a set T of time where $T = \{1,\ldots,\bar{t}\}$ and a set K where $K = \{1,\ldots,\bar{k}\}$ is the number of tasks to be assigned to the storage locations during this time horizon. Set M indicates storage locations where $M = \{1,\ldots,\bar{m}\}$. The set K can correspond to the set of tasks $\mathcal{J}$ in our work. There are also two parameters namely task arrival time and departure time that are defined as $a^k \in T$ and $b^k \in T$, respectively. These parameters can fairly correspond to our task arrival time of $a_j^{T_n}$ and task deadline $d_j^{T_n}$, respectively. In addition, for the sake of notational convenience, the authors in [106] defined the following sets: $T(k) = \{t \in T : a^k \leq t \leq b^k\}\ \forall k \in K$ which shows the stay of task k; the set $T^-(k) = \{t \in T : a^k \leq t \leq b^k - 1\}\ \forall k \in K$ that indicates the stay of the task k before its reallocation; and the set $K(t) = \{k \in K : t \in T(k)\}, \forall t \in T$, which represents the tasks that are in the system at time t.

The task space requirements and the task positions upon their arrival and departure are also considered. In addition, the problem considers the reallocation and the maximum travel time of each task between its arrival position and its first storage position. These parameters are shown via the variables q^k, o^k, d^k, r^k, c_a^k and c_b^k. Furthermore, the DGAP uses a binary decision variable, z_{lt}^k, which is equal to 1 if task k is assigned to position l in period t, and 0 otherwise, and a decision variable, α_{lmt}^k, that is equal to 1 if task k is assigned to position l in period t and to position m in period $t+1$, otherwise it is equal to 0. The objective is to minimize the overall handling cost for each task $k \in K$, including the handling from the last position to the departure position, and all the re-allocations during the planning horizon T.

There are a few constraints associated with the DGAP. The first constraint confirms that each task is assigned to one position at a time. The second constraint ensures that the task's required space does not exceed the location's available space. The DGAP also considers the arrival and departure of the tasks, followed by the reallocation possibilities in the other time slots shown in their last constraint.

$$\min \sum_{k\in K}\{\sum_{l\in M}(c_{o^k l}\, z_{la^k}^k + c_{ld^k}\, z_{lb^k}^k) + \sum_{l\in M}\sum_{m\in M}\sum_{t\in T^-(k)} c_{lm}\alpha_{lmt}^k\}$$

s.t.

$$\sum_{l\in M} z_{lt}^k = 1 \quad \forall\, k \in K,\ t \in T(k),$$

$$\sum_{k\in K(t)} q^k z_{lt}^k \leq Q_l \quad \forall\, l \in M,\ t \in T,$$

$$\sum_{l\in M} c_{o^k l}\, z_{la^k}^k \leq c_a^k \quad \forall\, k \in K,$$

$$\sum_{l \in M} c_{ld^k} z^k_{lb^k} \leq c^k_b \quad \forall\, k \in K,$$

$$z^k_{lt} + z^k_{m(t+1)} - a^k_{lmt} \leq 1 \quad \forall\, k \in K, l \in M, t \in T^-(k).$$

Let us now consider an instance of the dynamic application sensing task problem in a virtualized WSN to show that the DGAP reduces to our dynamic task assignment problem. Considering the objective function defined in Eq. (3.12) followed by its constraints, it is evident that we are trying to minimize the overall energy consumption that occurs due to task assignment to the physical or virtual sensors over time, with routing energy consumption being considered as well. Our constraints (3.13) to (3.16) are fairly matched with the first constraint defined in the DGAP, as we are concerned about all the tasks being assigned to either a physical or a virtual sensor at a given time, as a task cannot be assigned to both a physical sensor and a virtual sensor simultaneously. Furthermore, by considering that a sensor node can accept a maximum number of virtual sensors over it, our constraint (3.17) is matched with what is shown in the second constraint in the DGAP. This can also be applied to the case of energy as well. As in our constraints (3.18) to (3.20), the task should not be assigned to a physical or virtual sensor if that node does not have enough energy available. The third and fourth constraints in the DGAP restrict the maximum travel time between the arrival position of a task and its first location, and the task's departure position and its last position. These constraints are mapped to our constraint (3.21) that states all the tasks must be assigned before their deadline. In other words, the maximum time that a task can wait in a queue before being assigned to a PS or a VS. Also, given that our model allows the assignment of those tasks that arrived in previous time slots and have not yet been executed, this can be mapped to the last constraint shown in the DGAP. Clearly, our developed formulation matches the DGAP problem, which is known as an NP-hard problem. Therefore, our dynamic task assignment problem in a virtualized WSN is also an NP-hard problem.

3.2.3 DTA: A Heuristic for Dynamic Task Assignment in Virtualized WSN

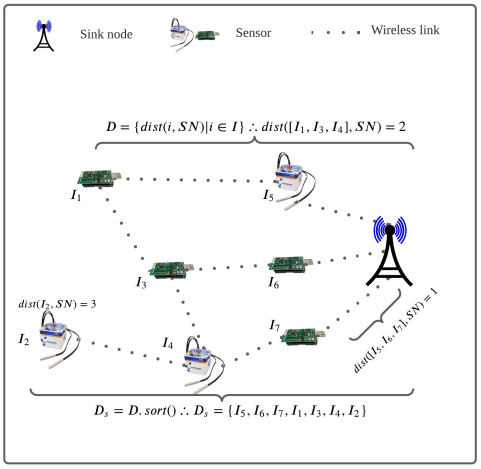
According to the proposed mathematical model, obtaining the solution for large-scale scenarios in a timely manner is rather complex. Hence, we propose the DTA heuristic, as described in Algorithms 3.1 to 3.3, to solve the problem for large-scale scenarios in an appropriate time-frame. The DTA heuristic seeks to determine the assignment of the sensing tasks to the physical and virtual sensor nodes. The main goal is to minimize the overall energy consumption while respecting the QoS constraints, i.e., the task deadlines, set by the application, and it specifies the maximum time a task may wait before it can be executed. We considered the constants shown in Table 3.1 as the inputs of the DTA heuristic. We also considered a set of sensor nodes (i.e. $\mathcal{I}$), a set of tasks (i.e. $\mathcal{J}$), and a sink node (SN). The output of the DTA is sets x and z of senor-task assignments. Our proposed DTA algorithm consists of three main phases, as

Algorithm 3.1 DTA Heuristic

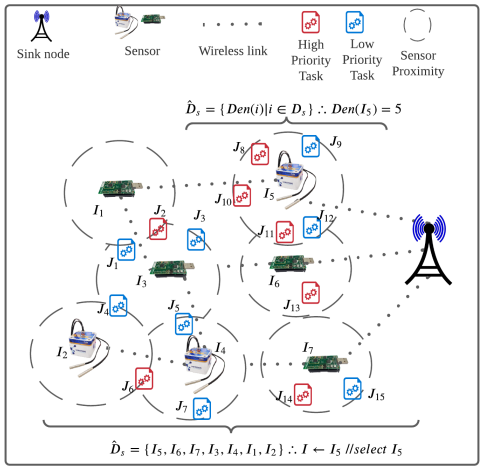
Input: $\mathcal{I}, \mathcal{J}, \mathcal{T}, SN$
Output: x, z /* sets of assignment decisions */
1: **for** $T_n \in \mathcal{T}$ **do**
2: $\mathcal{D}$= {dist(i,SN) / $i \in \mathcal{I}$}; /* distance from the SN */
3: $\mathcal{D}_s$=$\mathcal{D}$.sort(); /* sort $\mathcal{D}$ in ascending order */
4: $\hat{\mathcal{D}}_s$= {Den(i) / $i \in \mathcal{D}_s$}; /* task density over nodes */
5: i=1;
6: **while** $i < \mathcal{D}_s$.size() **do**
7: **if** $\mathcal{D}_s(i) = \mathcal{D}_s(i+1)$ **then**
8: **if** $\hat{\mathcal{D}}_s(i) = \hat{\mathcal{D}}_s(i+1)$ **then**
9: **if** $E(\hat{\mathcal{D}}_s(i)) > E(\hat{\mathcal{D}}_s(i+1))$ **then**
10: select sensor i;
11: **else**
12: select sensor i+1;
13: **end if**
14: **else**
15: select sensor i;
16: **end if**
17: **else**
18: select sensor i;
19: **end if**
20: $i \leftarrow i+1$;
21: **end while**
22: ***TS***(i, T_n); /* Task Selection for sensor node i at time slot T_n */
23: ***CstVerf***(i); /* Constraints Verification and Assignment for selected sensor node i */
24: **end for**

shown in Fig. 3.2. The first phase is sensor selection. As specified in Algorithm 3.1, our proposed heuristic iterates over a set $\mathcal{T}$ of time slots. Next, the DTA finds the distance between each sensor node and the sink node. This part is performed by function *dist()*, which is responsible for finding the distance of each sensor node to the sink node. The Euclidean Distance function is applied to explore the tasks in proximity of the sensor nodes and shortest-path-to-destination is used find the distances between the sensor nodes and the sink node (see Fig. 3.2(a)). For example, as shown in Fig. 3.2(a), sensor nodes I_5, I_6 and I_7 are the closest sensor nodes to the sink node, sensor nodes I_1, I_3 and I_4 are equi-distant from the sink node, and sensor node I_2 is the node furthest from the sink node.

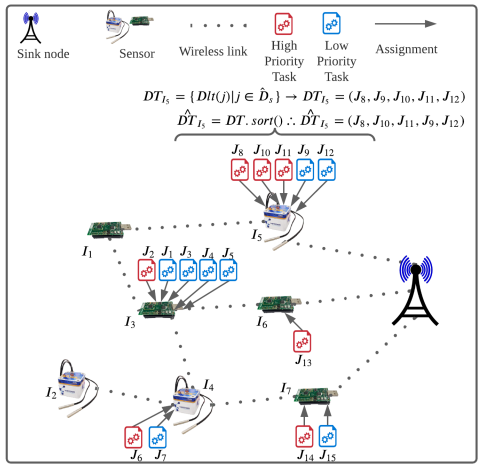
Once the distance of each sensor from the sink node has been found, they are sorted in ascending order, from the closest to the furthest from the sink node, using the $sort()$ function (see line 3 in Algorithm 3.1). It is important to note that all the sort functions used in this work are found in Dual-Pivot Quicksort [107]. The first sensor node from the sorted list is selected as the initial point for the task assignment, as depicted in Fig. 3.2(a). If there are more than two sensor nodes with the same distance from the sink node, the list of candidate tasks for each sensor node will be derived from the $Den()$ function as shown in Algorithm 3.4. Figure 3.2(b) illustrates an overview of the task density around each sensor node. Sensor nodes are surrounded by a set of candidate tasks, i.e., those tasks that are within the sensing range (or proximity) of a sensor node. Let us consider the example shown in Fig. 3.2(b), where sensor nodes I_1, I_3, and I_4 are located at an equal distance (i.e., two hops away) from the sink node. After obtaining the task densities, we notice that sensor node I_3 is surrounded by tasks J_1 to J_5, among which task J_5 is in the sensing range of sensor node I_4 as well. Given that the task density of each sensor node is calculated independently, task J_5 is counted for



(a) Sensor-sink distance.



(b) Sensor-task density.



(c) Task selection and assignment.

Figure 3.2: Illustration of different phases of our proposed DTA algoritm: (a) Sensor-sink distance, (b) Sensor-task density, and (c) Task selection and assignment.

Algorithm 3.2 Task Selection (***TS***)

1: ***TS***(i, T_n)
2: $\hat{\mathcal{D}}_s = \{Den(i) \,/\, i \in \mathcal{D}_s\}$;
3: $\mathcal{DT} = \{Dlt(j) \,/\, j \in \hat{\mathcal{D}}_s\}$; /* deadline time of each task */
4: $\hat{\mathcal{DT}}$=$\mathcal{DT}$.sort(); /* sort $\mathcal{DT}$ in ascending order */
5: j=1;
6: **while** $j < \hat{\mathcal{DT}}.size()$ **do**
7: **if** $(\hat{\mathcal{DT}}(j)) > (\hat{\mathcal{DT}}(j+1))$ **then**
8: select task j+1;
9: **else**
10: select task j;
11: **end if**
12: $j \leftarrow j + 1$;
13: **end while**

both sensor nodes I_3 and I_4. At this point, the sensor node with the highest task density will be selected after applying the $sort()$ function that sorts the list of sensor-task densities obtained by the $Den()$ function. This helps maximize the sensor node's resource utilization, leading to a smaller number of sensors activated in the network. The heuristic also considers the situation in which multiple sensor nodes are the same distance from the sink node and same task densities. In this case, we select the sensor node with the highest available energy (lines 7-19 in Algorithm 3.1).

Algorithm 3.1 depicts the pseudo-code of the main part of our proposed sensor selection method. It should be noted that Algorithms 3.2 and 3.3 are the sub-algorithms of the main Algorithm 3.1, and they are part of the main loop initiated by block **for** $T_n \in \mathcal{T}$ as in line 1. We have indicated them in separate algorithms for the sake of readability. The second phase of the DTA heuristic is the *TS()* function. Once the sensor node is selected, the next step is to select the task that should be assigned from the list of available tasks within the sensing range of the sensor. When the task selection has been completed, the last phase of the algorithm is the constraint verification and assignment function *CstVer(i)*.

The task selection process is essential, as knowing which task should be selected first to be assigned to the physical or virtual sensor node is a crucial step. As described, in the second phase of the algorithm, the $TS()$ function shown in Algorithm 3.2 is responsible for the task selection. For completeness, we depict the task selection phase of our proposed DTA in Fig. 3.2(c). Initially, the list of candidate tasks around each sensor node needs to be discovered. After determining the list of candidate tasks for each sensor node, their deadlines are retrieved, and then, the tasks are sorted based on their deadlines in an ascending order using the $Dlt()$ and $sort()$ functions, respectively (see lines 2-4 in Algorithm 3.2). As discussed earlier, all the sort functions in this work follow the Dual-Pivot Quicksort function [107].

Tasks with shorter deadlines have a higher priority than those with longer deadlines, emphasizing the importance of deciding which tasks should be selected first to be assigned to the sensor node. The task with the shortest deadline will be the one selected by the heuristic. As shown in Fig. 3.2(b), various tasks need to be assigned to the physical and virtual sensor nodes. To better distinguish between the tasks that hold shorter deadlines and those with longer

Algorithm 3.3 ***CstVerf***(i): Constraints Verification for node i

Input: $\mathcal{I}, \mathcal{J}, \mathcal{T}, SN$
Output: ***CstVerf***(i),x,z /* ***CstVerf***(i): boolean */
/* x : sets of assignment decisions on PS i */
/* z : sets of assignment decisions on VS i */
$\mathcal{C} = \{cap(i)/i \in \mathcal{I}\}$; /* sensor capacity: max. VSs that can be created on a sensor */
$\mathcal{E} = \{e(i)/i \in \mathcal{I}\}$; /* sensor node energies */
$\hat{\mathcal{E}} = \{e(i')/i' \in \mathcal{I}\}$; /* relay sensor nodes energies */
$\mathcal{D}$= {dist(i,SN) / $i \in \mathcal{I}$}; /* distance from the SN */
$\mathcal{D}_s$=$\mathcal{D}$.sort(); /* sort $\mathcal{D}$ in ascending order */
$\hat{\mathcal{D}}_s$= {Den(i) / $i \in \mathcal{D}_s$}; /* task density around nodes */
if $\mathcal{C}(i) \leq \mathbb{M}$ *and* $\mathcal{C}(i) > 0$ **then**
  if $E^{tr} \leq E^i_{T_n}$ **then**
    if $h \times D^{pr} < d^n_j$ **then**
      if $\hat{\mathcal{D}}_s(i) == 1$ **then**
        CstVerf$(i) \leftarrow$ true;
        $x_{i,j,T_n=1}$;
        $E^{i'}_{T_n} = E^{i'}_{T_n} - E^{tr\_re}$;/* update relay nodes' energies */
        $E^{i}_{T_n} = E^{i}_{T_n} - E^{tr}$;/* update the sensors' energies */
      else
        CstVerf$(i) \leftarrow$ true;
        $z_{i,j,T_n=1}$;
        $E^{i'}_{T_n} = E^{i'}_{T_n} - E^{tr\_re}$;/* update relay nodes' energies */
        $E^{i}_{T_n} = E^{i}_{T_n} - E^{tr}$;/* update the sensors' energies */
        $\mathbb{M} = \mathbb{M}$ - 1;/* update capacity */
      end if
    else
      CstVerf$(i) \leftarrow$ False;
    end if
  else
    CstVerf$(i) \leftarrow$ False;
  end if
end if
$T_{n+1}.add(j)$; /* Task j goes to T_{n+1}; */
return ***CstVerf***(i), x, z;

deadlines, they are shown in different colors. Tasks with shorter deadlines hold higher priorities requiring the sensor nodes to execute them immediately. For example, in Fig. 3.2(b), tasks J_2, J_6, J_8, J_{10}, J_{11}, J_{13} and J_{14} have short deadlines while the rest of the tasks have longer deadlines.

CstVer(i) function (Algorithm 3.3) is the third phase of the algorithm. Once the sensor is selected, the defined constraints in the model must be satisfied. The heuristic checks if the selected sensor node has enough capacity to accept new tasks. If the sensor node is virtualized it can accept M number of virtual sensors to be created on top of it, but if it has been used as a physical sensor, then $M = 0$. If this constraint is satisfied, the DTA verifies if the selected sensor node has enough energy to execute a new task. This minimum required energy is defined as the minimum energy required for transmission. With this constraint satisfied, the last criterion checked by the heuristic is the delay constraint. The sensor node processes the delay time; the routing delay to send the task execution result to the sink node

Algorithm 3.4 Den(): Task density around each sensor

Input: $\mathcal{I}$, $\mathcal{J}$, SR `/* Sensing Range */`
Output: $\hat{\mathcal{D}}_s$

$L[][]$; `/* A matrix showing if a task is in the proximity of the sensor */`
$\hat{\mathcal{D}}_s[][]$;s `/* A matrix that stores the sensor and the number of the tasks around it */`
for $i \in \mathcal{I}$ **do**
  for $j \in \mathcal{J}$ **do**
    EuDist = $\sqrt{(i_x - j_x)^2 + (i_y - j_y)^2}$
    if $EuDist \leq SR$ **then**
      $L[i][j] = 1$
    else
      $L[i][j] = 0$
    end if
  end for
  if $L[i][j] = 1$ **then**
    $\hat{\mathcal{D}}_s$.add((i).get(j))
  end if
end for
return $\hat{\mathcal{D}}_s$

must not exceed the task's deadline. If all these conditions are met, the heuristic will make the assignment decisions. A task is assigned to a PS if it is the only task in the candidate list of the sensor, otherwise, it is assigned to a VS.

To highlight the importance of the task density calculation phase, we take a closer look at Fig. 3.2(b) and Fig. 3.2(c), where sensor node I_3 has a higher task density than sensor nodes I_1 and I_4. Even though tasks J_1 and J_2 reside in the proximity of both sensor nodes I_1 and I_3, we select sensor node I_3, as it has a higher task density compared to sensor node I_1. Essentially, sensor node I_3 will serve all its surrounding tasks, including tasks J_1 and J_2. As a result, sensor node I_1 will remain unused, thus preserving its energy.

Once a decision has been made, the sensor node's energy is updated, as well as the energy of the intermediary sensor nodes that are affected because of the routing. If the task is assigned to a VS, the capacity of the sensor node will be updated accordingly. If any of the defined constraints are not satisfied, the heuristic checks for the next available sensor node that can satisfy all the constraints. If no sensor node is available, the heuristic schedules the task to be assigned to sensor nodes in future time slots. Our proposed DTA algorithm is instrumental in improving the energy efficiency of WSNs due to the following three reasons. First, in our sensor selection strategy, we select the sensors that are in close proximity to the sink node, thus reducing the energy consumption due to transmission. Second, in the calculation of sensor-task density, selecting the sensor node with the highest task density can reduce the number of active sensor nodes, thereby reducing the number of transmissions. Third, when there is only one task in the proximity of a sensor node, the sensor node will be used with its virtualization mode disabled, thereby preventing the excessive overhead induced by virtualization.

3.2.4 Complexity Analysis

In the following, we present the complexity analysis of our proposed DTA heuristic. The DTA algorithm consists of two main nested loops, three sorting operations using Dual-Pivot Quicksort [107], and verification of a series of linear constraints. The first loop goes through the set I of sensor nodes for $i = 1, \ldots, m$, while the second one explores in the set J of tasks for $j = 1, \ldots, n$. The time complexity of running through these nested loops is $\mathcal{O}(m \times n)$, which reduces to complexity $\mathcal{O}(n^2)$ given that $n > m$. The time complexity of the sorting operation is $\mathcal{O}(A \cdot n \cdot \log n)$, where A is a constant associated with the Dual-Pivot Quicksort scheme [108]. Verification of the given constraints and updating of the sensor-related parameters (e.g., remaining energy) runs at a worse-case complexity $\mathcal{O}(n)$. Therefore, given that the complexity of the nested loops dominates the other operations, the overall complexity of the proposed DTA algorithm reduces to $\mathcal{O}(n^2)$. For comparison, we present the complexity analysis of the SDSN method proposed in [33], where it is evident that three main nested loops have been used. The first loop runs over the set S of sensor nodes $s = 1, \ldots, p$, followed by the second loop that checks the set T of tasks, $t = 1, \ldots, q$, and the last loop that goes through the set G_t of sensing targets $g = 1, \ldots, r$. Given that $p > r > q$, the time complexity of the SDSN algorithm [33] is $\mathcal{O}(p \times q \times r)$, which reduces to complexity $\mathcal{O}(p^3)$.

3.3. Performance Evaluation

In the following, we present our evaluation scenarios followed by the obtained results.

3.3.1 Evaluation Scenarios

We conducted our evaluations over various scenarios. Different numbers of sensor nodes and tasks were considered, creating both small- and large-scale scenarios. Also, the impacts of virtualization overhead and task densities on energy consumption have been examined. Table 3.3 summarizes our simulation scenarios. The sensor nodes are scattered in different geographical areas. The tasks arrive in batches at the beginning of each time slot, with five time slots considered in each scenario. Each task has a geographical location and a deadline $d_j^{T_n}$ in each scenario. We follow the definitions of small- and large-scale scenarios given by Ref. [109], where small- and large-scale scenarios are defined as any scenario with $<$100 and $>$100 sensor nodes, respectively.

We ran several simulations with different parameter settings considering both the delay-tolerant and delay-sensitive status of the network. In the delay-tolerant setting, we set the task deadlines relatively long enough to allow the system to execute as many tasks as possible without dropping them. By contrast, in the delay-sensitive setting, the task deadlines must strictly be met; otherwise, the task has to be dropped. Our smallest scenario is 100 m $\times$ 100 m with 10 sensor nodes and a total of 250 tasks. The second small-scale scenario consists of 25 sensor nodes and a total of 500

Table 3.3: Evaluation scenarios.

	Small-scale Scenarios		Large-scale Scenario
	Scenario 1	**Scenario 2**	**Scenario 3**
Area ($Meter^2$)	100 × 100	250 × 250	1000 × 1000
Total Number of Sensor Nodes	10	25	500
Total Number of Time Slots	5	5	5
Total Number of Tasks	250	500	15500
Sensing Range	30 m	30 m	30 m
Transmission Range	59 m	59 m	59 m
Sensor Node's Initial Energy	2.9 J - 3.4 J	2.9 J - 3.4 J	2.9 J - 3.4 J

tasks across a 250 m × 250 m of area. Finally, our large-scale scenario involves 500 sensor nodes in a 1000 m × 1000 m of an area with a total of 15500 tasks.

In all three scenarios, we consider a sensing range of 30 m [52] and a maximum transmission range of 59 m [60] for the sensor nodes. In addition, for each sensor node, we consider random initial energy between 2.9 J and 3.4 J [35]. Also, we set the energy consumption of a sensor node while operating as a physical sensor to 0.017 mJ, which is the energy required for sensing and transmission [35]. The energy consumption overhead due to the creation of a virtual sensor over a physical sensor is set to 0.005 mJ [11]. Finally, we set the data reception energy at a sensor node to 0.031 mJ [35]. The energy consumption of CPU usage and storage is negligible. The transmission delay and virtual sensor creation overhead delay are set to 0.02 s and 0.06 s, respectively [11].

3.3.2 Evaluation Results

We present the optimal solution results obtained from CPLEX [110] and those obtained from our proposed DTA heuristic. We compare the performance of our proposed DTA algorithm with that of the SDSN algorithm [33] as our benchmark. It is important to note that in the SDSN algorithm [33], all the tasks should run at their arrival time without the possibility of being scheduled for the next time slots, even if the deadline allows. For a fair comparison, we relaxed this particular constraint of the SDSN algorithm to accommodate a larger number of tasks that can be assigned to the physical and virtual sensor nodes. We refer to this non-dropping version of the SDSN algorithm SDSN-i. In small-scale scenarios (i.e., scenarios 1 and 2) we derived the results from CPLEX, DTA, SDSN, and SDSN-i. However, for the large-scale deployment, the results are only obtained from the DTA, SDSN, and the SDSN-i, as it was not possible to obtain the results from the CPLEX engine in a feasible timeframe.

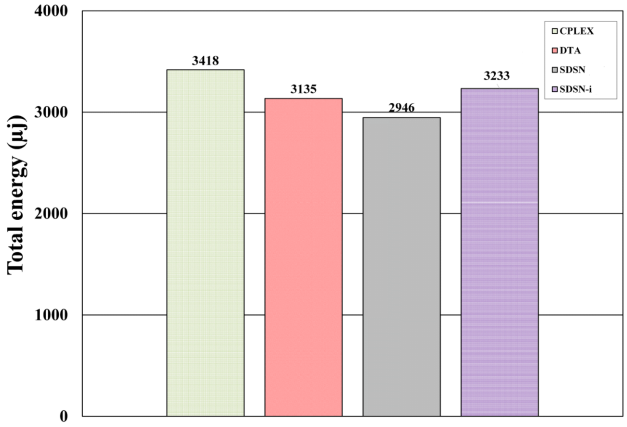
The mathematical model was implemented using CPLEX libraries in JAVA and the heuristics were implemented in JAVA. The tests were run on a machine with 64-bit OS Windows 10 Pro installed, 2.67 GHz Intel ® Xeon ® CPU E5640, and 32 GB of memory.

A) Energy Consumption

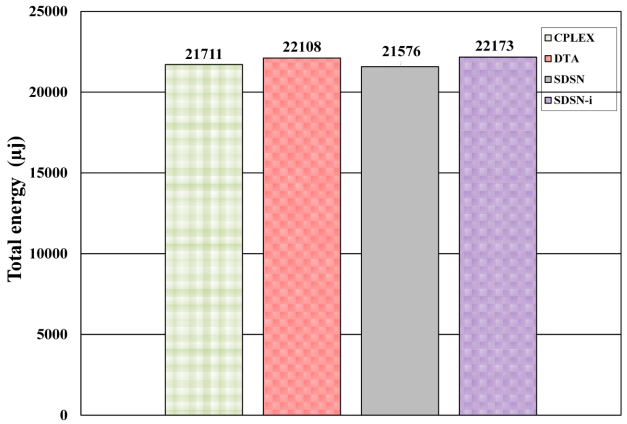
Figures 3.3(a), 3.3(b), and 3.3(c) illustrate the energy consumption records in three different scenarios for CPLEX, DTA, SDSN [33] and SDSN-i [33]. Figures 3.3(a) and 3.3(b) depict the energy consumption for the small-scale scenarios. Figure 3.3(a) reveals that the SDSN algorithm has the smallest energy consumption. This happens because the SDSN algorithm assigns a smaller number of tasks compared to the other methods. More specifically, if there are not enough resources available, the SDSN algorithm drops a given task, even though the task could have been considered for the next timeslot with respect to its deadline. This condition of moving a task to the next timeslot as needed is considered in the CPLEX, DTA, and SDSN-i.

In addition, it can be observed that the DTA has outperformed the SDSN-i in terms of energy consumption and has performed efficiently as compared to the optimal solution. The CPLEX has a slightly higher energy consumption than the other algorithms. This is because the sensor-task ratio in our first scenario (i.e., Fig. 3.3(a)) is about 25, which indicates that on average, one sensor exists per 25 tasks. In this case, there is high competition among the tasks for the available resources. Considering that the model requires the system to run as many tasks as soon as possible, it consumes a little more energy but runs all the possible tasks immediately with no delays. However, the DTA and SDSN-i algorithms tend to have more delays in running the tasks, and they run a smaller portion of the given tasks with no delays as compared to the CPLEX. In scenario 2, the sensor-task density ratio is reduced to 20. In this case, there is less competition between the tasks, and the results show that the CPLEX has a lower energy consumption than the DTA and the SDSN-i (see Fig. 3.3(b)). Similar to the first scenario shown in Fig. 3.3(a), our proposed DTA algorithm outperforms the SDSN-i algorithm. According to Fig. 3.3(c), which depicts the energy consumption in a large-scale scenario, the SDSN algorithm has the lowest energy consumption. As explained above, this happens due to its failure to run all the tasks. We also notice that our proposed DTA algorithm performs better than the SDSN-i algorithm in terms of total energy consumption.

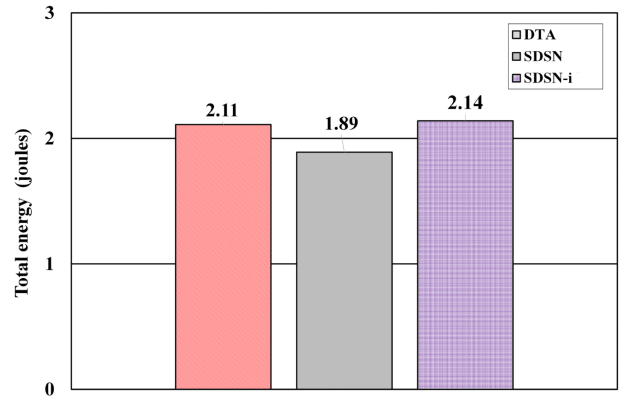
Next, we evaluate the energy efficiency of our proposed DTA algorithm for different values of virtualization overhead. Figure 3.4 illustrates the average energy consumption per task vs. virtualization overhead energy for three different scenarios under consideration. As it can be observed from Figs. 3.4(a) and (b), in two small-scale scenarios, the average energy consumption increases as virtualization overhead energy grows. Clearly, our proposed DTA algorithm achieves a near-optimal solution, outperforming both SDSN and SDSN-i algorithms. In the large-scale scenario, it is also evident that our proposed DTA algorithm outperforms both SDSN and SDSN-i algorithms. As virtualization overhead energy increases, the average energy consumption per task for SDSN-i grows dramatically. This is because of the difference in the algorithms in their assignment strategy to select the starting point. In DTA, the task assignment starts from the sensor nodes that are closest to the sink node, thus reducing the communication energy consumption. By contrast, in SDSN and SDSN-i algorithms, the task assignment starts from the sensor node with the highest task



(a) Scenario 1.

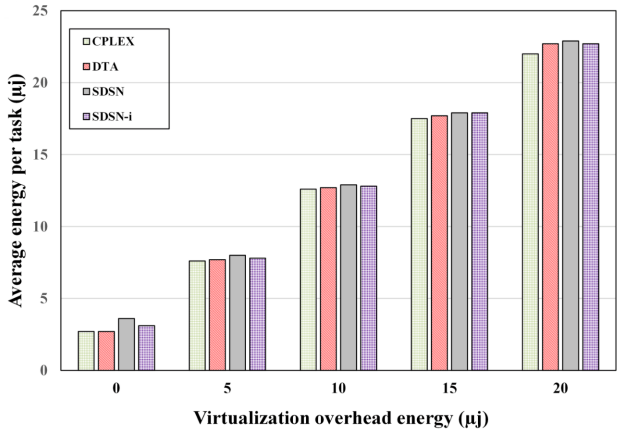


(b) Scenario 2.

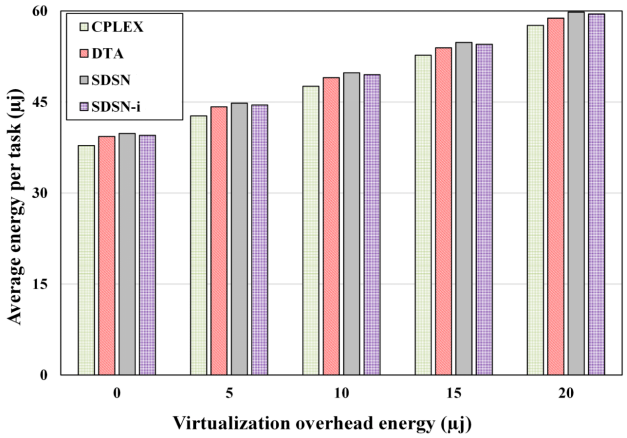


(c) Scenario 3.

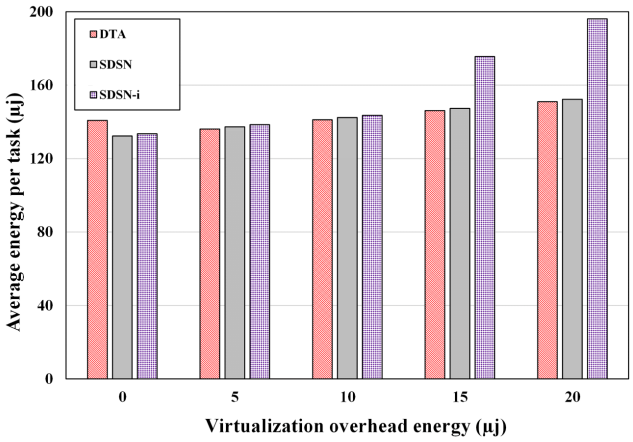
Figure 3.3: Total energy consumption for different assignment methods under study.



(a) Scenario 1.



(b) Scenario 2.



(c) Scenario 3.

Figure 3.4: Average energy per task vs. virtualization overhead energy.

density. Although the SDSN algorithm shows comparable average energy consumption in the large-scale scenario, it is not capable of executing 100% of the given tasks, as discussed earlier.

B) Number of Executed Tasks vs. Delay

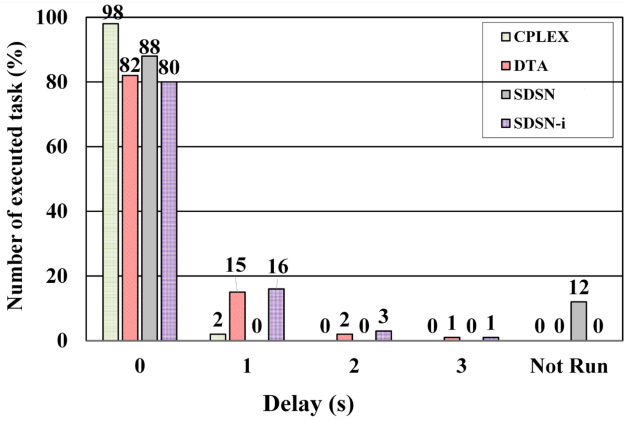
QoS is one of the key factors in delay-sensitive use cases, e.g., for disaster management as analyzed in this work. As described earlier, the QoS parameters here reflect the deadline constraints of the sensing tasks, as well as the requirement to run as many tasks as possible as soon as possible with relatively minimal delay. Accordingly, Figs 3.5(a), 3.5(b), and 3.5(c) show the number of executed tasks vs. delay in different scenarios. The results indicate that the CPLEX managed to assign tasks to the physical and virtual sensors with mainly no delays. That means all the tasks were assigned to the sensor nodes (PS/VS) at the time they arrived, with a minimal number of re-scheduling decisions required.

Accordingly, in small-scale scenarios as depicted in Figs. 3.5(a) and 3.5(b), our proposed DTA algorithm managed to assign 82% and 98% of the tasks with no delays to the physical and virtual sensors, respectively. On the other hand, for the first small-scale scenario shown in Fig. 3.5(a), the SDSN algorithm could run 88% of the tasks with no delays, while failing to run 12% of the tasks. In the second small-scale scenario shown in Fig. 3.5(b), the SDSN algorithm runs 96% of the given tasks, but still 4% of the tasks were dropped. On the other hand, the SDSN-i algorithm ran 80% of the tasks with no delays in our first scenario shown in Fig. 3.5(a). This percentage increased to 95% in the second scenario (see Fig. 3.5(b)). The DTA algorithm outperformed both SDSN and SDSN-i in small-scale scenarios by running a larger portion of the given tasks without dropping any of them. As for the large-scale scenario, Fig. 3.5(c) shows that our proposed DTA algorithm can assign 77% of the tasks to the physical or virtual sensors with no delays. On the other hand, the SDSN-i algorithm is only capable of assigning 73% of the tasks to the sensor nodes (PS/VS). Although SDSN can assign a larger portion of the tasks, 89% with no delays, it cannot run 11% of the tasks and drops them.

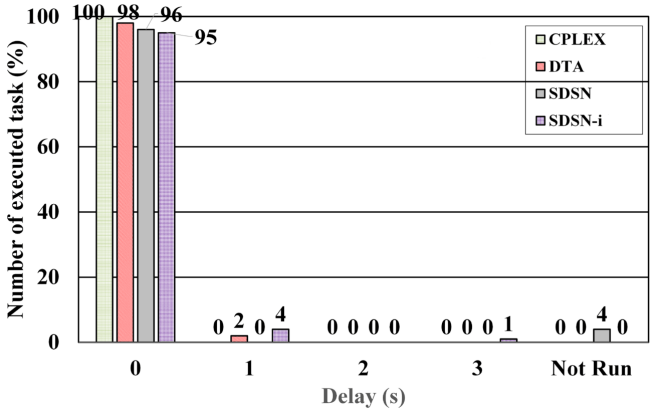
C) Successful Task Execution

Successful task execution rate is another important factor to be evaluated. Figures 3.6(a), 3.6(b), and 3.6(c) depict the successful task execution rate in different scenarios. It can be observed that in all the scenarios, 100% of the tasks can be executed by CPLEX, DTA, and SDSN-i. However, in the first scenario depicted in Fig. 3.6(a), the SDSN algorithm can only execute 88% of the tasks, which increases to 96% in the second small-scale scenario (see Fig. 3.6(b)). In large-scale scenario, the successful task execution ratio drops to 89% (see Fig. 3.6(c)). It is evident that our proposed DTA algorithm outperforms the SDSN algorithm in terms of successful task execution rate by 4% to 12% in various scenarios.

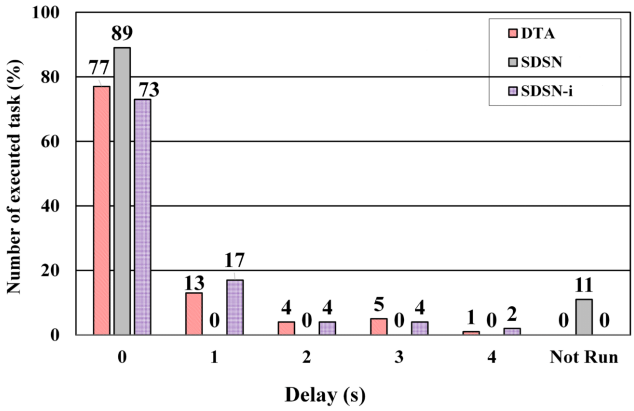
Furthermore, we have conducted simulations under different task load densities with different numbers of sensor



(a) Scenario 1.



(b) Scenario 2.



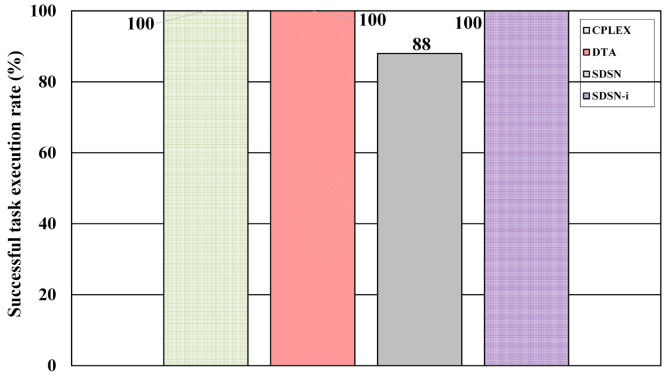
(c) Scenario 3.

Figure 3.5: Task Executed (%) vs. delay for different assignment methods under study under three evaluation scenarios.

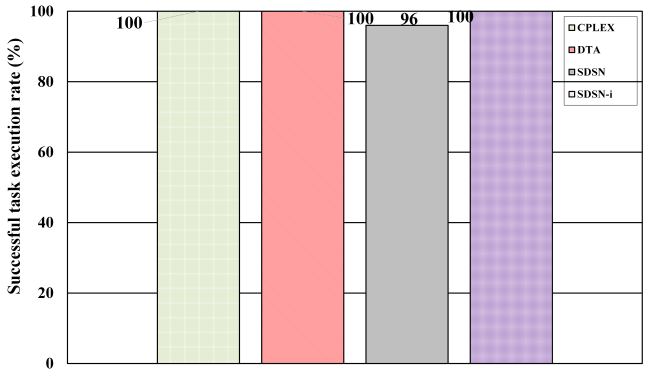
nodes. Figure 3.7(a) shows the successful task execution rate vs. total number of tasks. The simulations were run with 10 and 20 sensor nodes for both DTA and SDSN algorithms. We can observe from Fig. 3.7(a) that both DTA and SDSN algorithms managed to successfully execute 100% of the tasks for small task densities. The successful task execution decreases as the task density increases. Under the heaviest load with 3000 tasks and 10 sensor nodes, the successful task execution rate of our proposed DTA algorithm and SDSN algorithm is 21% and 11%, respectively. It can be observed from Fig. 3.7(a) that for 20 sensor nodes, the successful task execution rate of the proposed DTA algorithm is 29.8%, which is a significant improvement compared to the 15.5% of the SDSN algorithm. For 20 sensor nodes and 4500 tasks, the proposed DTA algorithm managed to successfully execute $\sim$30% of the tasks, while the SDSN algorithm only runs $\sim$15% of the tasks successfully. Figure 3.7(b) shows the total energy consumption vs. total number of tasks for 10 and 20 sensor nodes. According to Fig. 3.7(b), the DTA algorithm has a higher energy consumption compared to the SDSN algorithm. This was expected because our proposed DTA algorithm is associated with a higher successful task execution rate compared to its SDSN counterpart (see also Fig. 3.7(a)). Figure 3.7(b) indicates that the total energy consumption of proposed DTA and SDSN algorithms increases as the number of tasks grows. We note, however, that the energy consumption of both algorithms hits a plateau when the number of tasks reaches a certain amount (e.g., $\sim$1000 tasks for 10 sensor nodes). This happens because once the maximum sensor capacities are reached and a certain number of tasks are executed at each time slot, there will be no new assignment and therefore the energy consumption will not change.

D) Execution Time

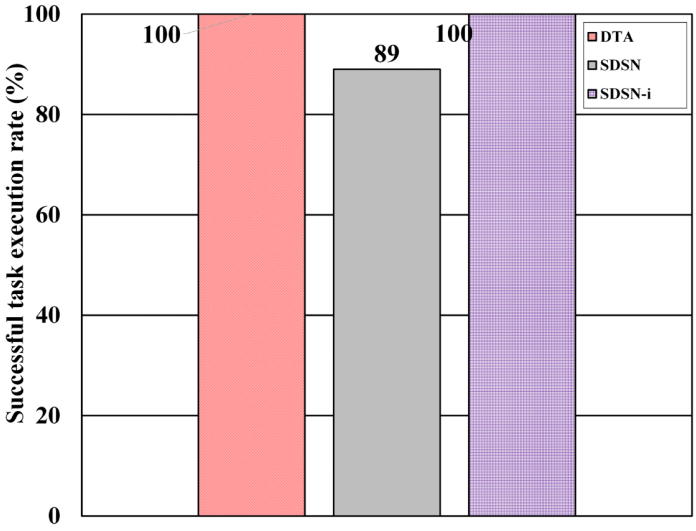
The execution times of the optimal, DTA, SDSN, and SDSN-i solutions are illustrated in Table 3.4 for the small- and large-scale scenarios. In the small-scale scenarios, where we had a very small set of sensors and a small number of tasks, DTA, SDSN, and SDSN-i performed quite similarly without much difference. DTA and the SDSN-i solved the problem in 21ms, while the SDSN solved the problem in 17ms. Similarly, the results for the second small-scale scenario indicate that the SDSN solved the problem 2 ms faster than both DTA and the SDSN-i. Although the SDSN performed 4 ms faster than DTA and SDSN-i in the first small-scale test and 2 ms faster in the second small-scale scenario, it ran a smaller percentage of the tasks, as explained earlier in figures 3.6(a) and 3.6(b), respectively. On the other hand, for the large-scale scenario where a scenario closer to the real world was defined with a denser network and thousands of tasks, DTA outperformed both the SDSN and SDSN-i. DTA solved the problem in 561 seconds whereas SDSN and SDSN-i solved the problem in 591 seconds and 651 seconds, respectively.



(a) Scenario 1.

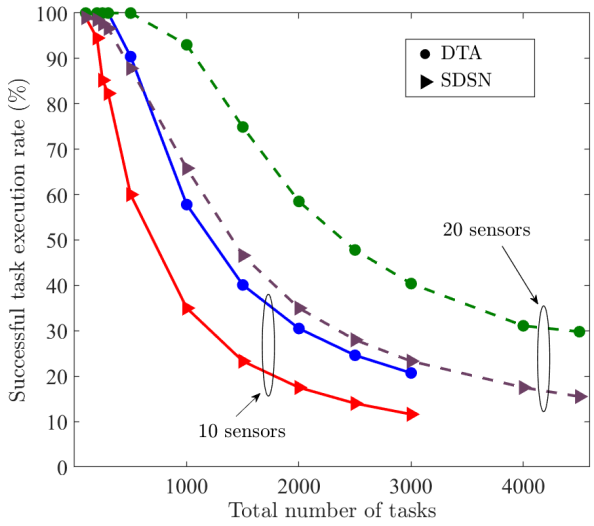


(b) Scenario 2.

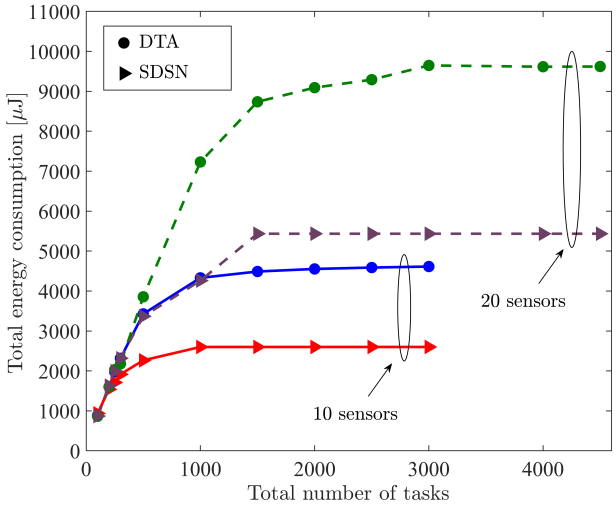


(c) Scenario 3.

Figure 3.6: Successful task execution rate for different assignment methods under study under three evaluation scenarios.



(a) Successful task execution rate (%).



(b) Total energy consumption vs. total number of tasks.

Figure 3.7: (a) Successful task execution rate (%) and (b) Total energy consumption vs. total number of tasks for 10 and 20 sensor nodes.

Table 3.4: Execution time.

	CPLEX	**DTA**	**SDSN**	**SDSN-i**
100m $\times$ 100m	750 ms	21 ms	17 ms	21 ms
250m $\times$ 250m	782 ms	63 ms	61 ms	63 ms
1000m $\times$ 1000m	—	561 sec	591 sec	651 sec

3.4. Conclusions

This chapter addresses the problem of dynamic sensing task assignment in virtualized wireless sensor networks by considering node-level virtualization. We model the problem as an ILP to assign sensing tasks to the pool of

physical and virtual sensors at the lowest energy consumption with the possibility of re-scheduling. We consider the forwarding, receiving, and routing impacts as well as the virtual sensor instantiation overhead as part of the energy factors. Furthermore, we define the QoS constraint: meeting the specific task deadlines and propose our heuristic to solve it in polynomial time. We conducted extensive simulations to evaluate our proposed heuristic against the optimal (CPLEX) solution and a solution from the literature (SDSN). The evaluations were conducted over three different scenarios in both small- and large-scale environments. Our results indicate that the proposed DTA heuristic can perform at a near-optimal level in terms of energy consumption while respecting the task deadline requirements and constraints. It outperforms the SDSN in execution time, task execution successful rate, and delay.

Chapter 4

Energy Efficient Dynamic Network Embedding in Virtualized Wireless Sensor Networks [1]

4.1. Introduction

Wireless Sensor Network virtualization [6] research has become an interesting, yet challenging research domain. Virtualization in WSNs can be applied at the node- and/or network-level. The node-level virtualization enables abstracting the sensing capabilities of the sensor nodes, into logical sensing capabilities, allowing multiple sensing tasks to be executed over the same sensor node concurrently. On the other hand, network-level virtualization allows the creation of multiple Virtual Sensor Networks (VSNs) over the same deployed WSN infrastructure [6] that each VSN is dedicated to one application. However, considering that the WSN sensor nodes are mainly battery-powered, it is crucial to allocate the pool of physical and virtual sensors at the substrate WSN network to the applications in an energy-efficient manner. Therefore, customized VSNs must be created on top of the deployed WSN, while satisfying the given Service Level Agreements (SLAs) requirements (i.e. E2E latency and Bandwidth) constraints. This is commonly known as network embedding problem [14]. This chapter attempts to address the aforementioned challenge.

The rest of the chapter is organized as follows. We first introduce a high-level system model along with problem formulation that includes both substrate network and virtual network. We then present the proposed heuristic that

[1] This chapter is based on a published paper and a submitted IEEE Transaction journal: [3] V. M. Raee, A. Ebrahimzadeh, M. Rayani, R. Glitho, M. El Barachi, and F. Belqasmi, "Energy efficient virtual network embedding in virtualized wireless sensor networks," in Proc. IEEE Consumer Communications & Networking Conference (CCNC), pp. 187–192, 2022. and [4] V. M. Raee, A. Ebrahimzadeh, R. H. Glitho, M. El Barachi, and F. Belqasmi, "E2DNE: Energy efficient Dynamic Network Embedding in Virtualized Wireless Sensor Networks," submitted to IEEE Transactions on Green Communications and Networking, 2022 (Under Review).

considers the virtualized substrate WSN network and the given SLA (i.e. E2E latency and bandwidth) requirements. Next, we present the performance evaluation, and finally, we conclude this chapter in the last subsection.

4.2. Dynamic Virtual Network Embedding: System Model and Problem Formulation

In this section, we present our system model that covers the problem definition, followed by the problem formulation and problem analysis. The developed formulation is an ILP formulation of the problem where a linear objective function and series of linear constraints are used with integer decision variables that are explained in-depth technically in Section 4.2.2.

4.2.1 System Model

We consider a WSN with both node- and network-level virtualization capabilities, as shown in Fig. 4.1. The main substrate network consists of *mixed-physical/-virtual sensors*, meaning that the substrate network with both non-virtualizable and virtualizable sensors coexist. This substrate network offers enough flexibility to the application provider to utilize the available resources based on the desired objectives (e.g., energy efficiency). Therefore, depending on the application type and its requirements, a substrate of all-physical, all-virtual, and mixed-physical/-virtual sensors may get utilized. The system includes a set of time slots during which the applications' requests arrive in batches and are queued at the beginning of each time slot. Applications (e.g., fire contour and humidity expansion map, etc.) send the sensing tasks to the same substrate network to be executed simultaneously in a distributed manner. The scheduler receives the virtual network requests from the application layer and runs the embedding algorithm to map the sensing tasks to the underlying virtualized WSN network. Once the tasks are executed by the sensor nodes, the results of the readings will be sent back to the scheduler to address the applications' requests. The objective is to map these virtual networks to the available substrate network at minimal energy consumption. Once a sensing task is assigned to a sensor node (physical or virtual), it will be executed immediately. Sensor nodes may operate in either physical or virtual mode, meaning that if a sensor node operates on physical mode, its virtualization functionality is disabled[2].

We view time as discretized set $\mathcal{T} = \{t_1, t_2, ..., t_s\}$ of time slots, where time slot $t_s \in \mathcal{T}$ corresponds to the network embedding at t_s. Virtual network requests are generated by applications randomly and then served by the scheduler in batches at the end of each time slot t_s. Let t_s represent the time slot s and α_s be the set of the virtual network requests at t_s. We model the network embedding in virtualized WSNs with the given bandwidth and latency

[2]This assumption is compliant with widely used sensors such as Advanticsys [16] and Virtenio [17]), which may operate in either physical or virtual mode.

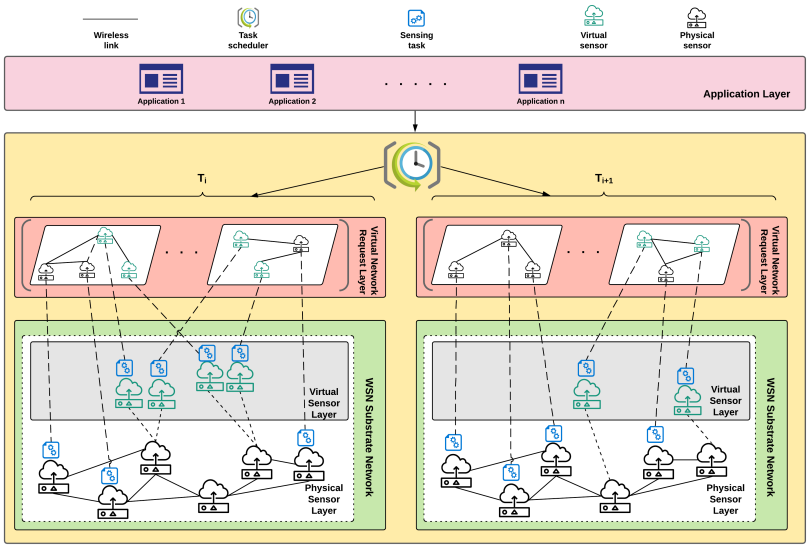


Figure 4.1: High-level system model.

constraints. Similar to [111], we assume that the synchronization error is small. Our system model can be viewed from the following aspects: substrate network and virtual network.

A) Substrate Network

We represent the virtualized WSN (vWSN) substrate network as a weighted undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{G} = \mathcal{G}_p \cup \mathcal{G}_q$. We note that $\mathcal{G}_p = (\mathcal{V}_p, \mathcal{E}_p)$ and $\mathcal{G}_q = (\mathcal{V}_q, \mathcal{E}_q)$ represent the weighted undirected graphs of physical sensors and the virtual sensors, respectively. $\mathcal{V}_p$ and $\mathcal{E}_p$ denote the physical sensors and their associated physical links, while $\mathcal{V}_q$ and $\mathcal{E}_q$ denote the virtual sensors instantiated on top of the physical sensors and their logical connections to the physical sensors, respectively. Accordingly, $\mathcal{V} = \mathcal{V}_p \cup \mathcal{V}_q = \{v_1, \ldots, v_M\}$ is the set of sensor nodes, with each sensor node $v \in \mathcal{V}$ representing a physical sensor or a virtual sensor instantiated on top of the physical sensor with x_{v_i} and y_{v_i} representing the sensor node's coordination in a grid. Set $\mathcal{E} = \{e_1, \ldots, e_{M'}\}$ represents the set of edges between the sensor nodes, where $\mathcal{E} = \mathcal{E}_p \cup \mathcal{E}_q$. Edge $(v_i, v_{i'}) \in \mathcal{E} \mid v_i \neq v_{i'}$, represents the communication link between sensor nodes v_i and $v_{i'}$. Also, let $D_g(v_i, v_{i'})$ and $B_g(v_i, v_{i'})$ represent the latency and bandwidth associated with edge $(v_i, v_{i'})$, respectively.

B) Virtual Network

Let $\mathcal{A} = \{a_1, \ldots, a_k\}$ be the set of Virtual Networks (VNs) that need to be mapped to the substrate network. We build a weighted directed graph $\mathcal{H} = (\mathcal{N}, \mathcal{L})$ to represent a given VN. $\mathcal{N}$ is the set of virtual nodes defined as $\mathcal{N} = \{n_1, \ldots, n_R\}$ and $\mathcal{L} = \{l_1, \ldots, l_{R'}\}$ denotes the logical links between virtual nodes n_j and $n_{j\prime}$ in the VN, where $(n_j, n_{j'}) \in \mathcal{L} \mid n_j \neq n_{j'}$. Let x'_{n_j} and y'_{n_j} denote the requested virtual node's coordination in a grid. Each VN is

associated with latency and bandwidth requirements $D_h(n_j, n_{j'})$ and $B_h(n_j, n_{j'})$, respectively.

Tables 4.1 and 4.2 summarize the problem main notations and problem inputs, respectively.

4.2.2 Problem Formulation

In this paper, we study the problem of dynamic virtual network embedding in virtualized wireless sensor networks. We aim to embed the applications' VNs to the substrate network at a minimal energy consumption while respecting the given latency and bandwidth requirements. The problem can be defined as $f : \mathcal{H} \rightarrow \mathcal{G}$, which is a function that describes the mapping of graph $\mathcal{H}$ over graph $\mathcal{G}$. In the following, we formulate the dynamic vWSN virtual network embedding problem as an ILP by defining the following decision variables:

$$X_{v_i,n_j,t_s} = \begin{cases} 1, & \text{if node } n_j \in \mathcal{N} \text{ is mapped to the physical sensor } v_i \in \mathcal{V} \text{ at time } t_s, \\ 0, & otherwise. \end{cases} \tag{4.1}$$

$$Y_{v_i,t_s} = \begin{cases} 1, & \text{if sensor node } v_i \in \mathcal{V} \text{ is virtualized at time } t_s, \\ 0, & otherwise. \end{cases} \tag{4.2}$$

$$Z_{v_i,n_j,t_s} = \begin{cases} 1, & \text{if node } n_j \in \mathcal{N} \text{ is mapped to the virtual sensor } v_i \in \mathcal{V} \text{ at time } t_s, \\ 0, & otherwise. \end{cases} \tag{4.3}$$

and,

$$U^{n_j,n_{j'}}_{v_i,v_{i'},t_s} = \begin{cases} 1, & \text{if } (n_j, n_{j'}) \in \mathcal{N} \text{ is mapped to } (v_i, v_{i'}) \in \mathcal{E} \text{ at time } t_s, \\ 0, & otherwise. \end{cases} \tag{4.4}$$

In the following, after explaining our energy and delay model, we present the objective function of our ILP followed by the constraints.

A) Energy Consumption

Overall energy consumption can be divided into the following three components.

Table 4.1: General notations of the problem.

Notation	Definition
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Substrate network graph with nodes $\mathcal{V}$ and edges $\mathcal{E}$ linking them
$\mathcal{G}_p = (\mathcal{V}_p, \mathcal{E}_p)$	Substrate network graph with physical sensor nodes $\mathcal{V}_p$ and edges $\mathcal{E}_p$ linking them
$\mathcal{G}_q = (\mathcal{V}_q, \mathcal{E}_q)$	Substrate network graph with virtual sensors $\mathcal{V}_q$ and logical links $\mathcal{E}_q$ linking them to physical sensor nodes which they instantiated over
$v_i \in \mathcal{V}$	A sensor node (physical/virtual) in $\mathcal{V}$
x_{v_i}, y_{v_i}	sensor node's v_i coordination in a grid
$E_{t_s}^{v_i}$	Current available energy of sensor node v_i at t_s
$(v_i, v_{i'}) \in \mathcal{E}$	An edge in $\mathcal{E}$
$D_g(v_i, v_{i'})$	Latency associated with edge $(v_i, v_{i'})$
$B_g(v_i, v_{i'})$	Bandwidth associated with edge $(v_i, v_{i'})$
$\mathcal{H} = (\mathcal{N}, \mathcal{L})$	Virtual network graph with virtual nodes $\mathcal{N}$ and logical links $\mathcal{L}$ linking them
$n_j \in \mathcal{N}$	A virtual node in $\mathcal{N}$
$(n_j, n_{j'}) \in \mathcal{L}$	A logical link in $\mathcal{L}$
x'_{n_j}, y'_{n_j}	virtual node's n_j coordination in a grid
$D_h(n_j, n_{j'})$	Latency associated with edge $(n_j, n_{j'})$
$B_h(n_j, n_{j'})$	Bandwidth associated with edge $(n_j, n_{j'})$
$\mathcal{T}$	Total number of time slots
t_s	Total virtual network embedding frame time
α_s	Size of each group of virtual network request
C_{v_i}	Capacity of sensor node v_i
$E_{v_i}^{tr}$	Forwarding energy consumption of sensor node v_i
$E_{v_i}^{CPU}$	CPU energy consumption of sensor node v_i
E^{ov}	Virtual sensor creation energy consumption
$E_{v_i}^{l}$	Wireless link energy consumption
$D_{v_i}^{tr}$	Transmission delay
D^{ov}	Virtual sensor creation delay
α	The relative degree (in percentage) of connectivity of the network with respect to a complete graph
β	The ratio (in percentage) of the virtualizable sensors to the total number of sensors

Table 4.2: Problem inputs.

Input	Definition
$E_{t_s}^{tot}$	Overall energy consumption of embedding at time t_s
$E_{t_s}^{P}$	Energy consumption of physical sensor v_i at time t_s
$E_{t_s}^{V}$	Energy consumption of virtual sensor v_i at time t_s
$E_{t_s}^{comm}$	Energy consumption of wireless communication at time t_s
$D_{t_s}^{P}$	Delay of physical sensor v_i at time t_s
$D_{t_s}^{V}$	Delay of virtual sensor v_i at time t_s

- **Physical substrate execution energy** $(E^P_{t_s})$**:** $E^P_{t_s}$ represents the execution energy consumption of mapping virtual node $n_j \in \mathcal{N}$ to the physical sensor at time t_s in the substrate network. We calculate $E^P_{t_s}$ as follows:

$$E^P_{t_s} = \sum_{n_j \in \mathcal{N}} \sum_{v_i \in \mathcal{V}} X_{v_i, n_j, t_s} \, (\, E^{tr}_{v_i} + E^{\text{CPU}}_{v_i}), \tag{4.5}$$

where $E^{tr}_{v_i}$ and $E^{\text{CPU}}_{v_i}$ are transmission and CPU energy consumption of a sensor v_i, respectively.

- **Virtual substrate execution energy** $(E^V_{t_s})$**:** $E^V_{t_s}$ account for the execution energy consumption of mapping the virtual node $n_j \in \mathcal{N}$ to the virtual sensors instantiated over the physical sensors in the substrate network at time t_s. It takes into account the energy consumption of transmission, CPU, and virtualization overhead. We obtain $E^V_{t_s}$ as follows:

$$E^V_{t_s} = \sum_{v_i \in \mathcal{V}} Y_{v_i, t_s} \, E^{tr}_{v_i} + \sum_{n_j \in \mathcal{N}} \sum_{v_i \in \mathcal{V}} Z_{v_i, n_j, t_s} \, (\, E^{ov} + E^{\text{CPU}}_{v_i}), \tag{4.6}$$

where, E^{ov} is virtualization energy overhead which is incurred due to virtual sensor instantiation on a physical sensor node.

- **Energy consumption** $(E^{comm}_{t_s})$ **of wireless communication links:** $E^{comm}_{t_s}$ includes the energy consumption of the wireless links of the substrate network by mapping the logical communication link $(n_j, n_{j'}) \in \mathcal{L}$ to the wireless link $(v_i, v_{i'}) \in \mathcal{E}$ at time t_s. We obtain $E^{comm}_{t_s}$ as follows:

$$E^{comm}_{t_s} = \sum_{(n_j, n_{j'}) \in \mathcal{L}} \sum_{(v_i, v_{i'}) \in \mathcal{E}} E^l_{v_i} U^{n_j n_{j'}}_{v_i v_{i'}, t_s}, \tag{4.7}$$

where, $E^l_{v_i}$ is the wireless link energy consumption of sensor v_i.

- **Overall energy** $(E^{tot}_{t_S})$**:** Overall energy $(E^{tot}_{t_s})$ is the summation of the above mentioned three energy components, given by:

$$E^{tot}_{t_s} = E^P_{t_s} + E^V_{t_s} + E^{comm}_{t_s}. \tag{4.8}$$

B) Delay

We consider the following two different execution delay components, which are the delay factors affecting the WSN:

- **Physical substrate execution delay** $(D^P_{t_s})$**:** $D^P_{t_s}$ is the execution delay incurred by mapping virtual node $n_j \in \mathcal{N}$ to the physical sensor in the substrate network at time t_s. We estimate $D^P_{t_s}$ as follows:

$$D^P_{t_s} = \sum_{n_j \in \mathcal{N}} \sum_{v_i \in \mathcal{V}} X_{v_i,n_j,t_s} \, D^{tr}_{v_i}, \tag{4.9}$$

where, $D^{tr}_{v_i}$ is the transmission delay of sensor v_i.

- **Virtual substrate execution delay** $(D^V_{t_s})$**:** $D^V_{t_s}$ represents the execution delay of mapping virtual node $n_j \in \mathcal{N}$ to virtual sensor in the substrate network at time t_s. We estimate $D^V_{t_s}$ as follows:

$$D^V_{t_s} = \sum_{v_i \in \mathcal{V}} Y_{v_i,t_s} \, D^{tr}_{v_i} + \sum_{n_j \in \mathcal{N}} \sum_{v_i \in \mathcal{V}} Z_{v_i,n_j,t_s} \, (\, D^{ov}). \tag{4.10}$$

where, D^{ov} is the virtualization delay overhead.

C) Objective Function and Constraints

In our problem, we strive for providing the decision on embedding the virtual networks onto the virtualized WSN substrate network at the lowest energy consumption while respecting the latency and bandwidth SLA-related constraints. Towards this end, let R be the total number of sensor nodes. We define our objective function as follows:

$$\min \sum_{t_s=0}^{\mathcal{T}} (E^P_{t_s} + E^V_{t_s} + E^{comm}_{t_s}), \tag{4.11}$$

which aims to minimize the summation of energies consumed by physical sensors, virtual sensors, and communication links.

Constraints: First, all VN nodes must be mapped to a node from a substrate node:

$$\sum_{v_i \in \mathcal{V}} X_{v_i,n_j,t_s} + \sum_{v_i \in \mathcal{V}} Z_{v_i,n_j,t_s} = 1; \quad \forall \, n_j \in \mathcal{N}, t_s \in \mathcal{T}. \tag{4.12}$$

A sensor node can operate in physical or virtualized mode. If a sensor node operates in a virtualized mode, no task should be assigned to it in the physical mode. However, if a sensor node is operating in the physical mode, it can support a maximum of one task. These constraints are realized as follows:

$$\sum_{n_j \in \mathcal{N}} X_{v_i,n_j,t_s} \leq 1 - Y_{v_i,t_s} \quad \forall \, v_i \in \mathcal{V}, \; t_s \in \mathcal{T}. \tag{4.13}$$

Further, a sensor node can support a maximum of C_{v_i} virtual sensors. Thus, the capacity of each sensor node

should not be exceeded:

$$\sum_{n_j \in \mathcal{N}} Z_{v_i,n_j,t_s} \leq C_{v_i} Y_{v_i,t_s} \quad \forall\, v_i \in \mathcal{V},\ t_s \in \mathcal{T}. \tag{4.14}$$

A task should not be assigned to a sensor node if the sensor node does not have enough energy:

$$X_{v_i,n_j,t_s}(E^{tr}_{v_i} + E^{CPU}_{v_i}) \leq E^{v_i}_{t_s} \quad \forall\, v_i \in \mathcal{V},\ n_j \in \mathcal{N},\ t_s \in \mathcal{T}, \tag{4.15}$$

and

$$Y_{v_i,t_s} E^{tr}_{v_i} + \sum_{n_j \in \mathcal{N}} Z_{v_i,n_j,t_s}(E^{ov} + E^{CPU}_{v_i}) \leq E^{v_i}_{t_s} \quad \forall\, v_i \in \mathcal{V},\ t_s \in \mathcal{T}, \tag{4.16}$$

where, $E^{v_i}_{t_s}$ refers to the available energy of the sensor node v_i at time t_s, which is achieved by using a recursive model. The available energy at time t_s is equal to the available energy at time slot t_{s-1} minus the energy consumption at time slot t_s:

$$E^{v_i}_{t_s} = E^{v_i}_{t_{s-1}} - E^{P}_{t_s} - E^{V}_{t_s} - E^{comm}_{t_s}, \tag{4.17}$$

where, $E^{v_i}_{t_{s-1}}$ represents the initial available energy on sensor node v_i, at time slot t_{s-1}.

In addition, we must ensure that the neighboring virtual nodes in the VN are also connected at the substrate WSN network:

$$U^{n_j n_{j'}}_{v_i v_{i'},t_s} = X_{v_i,n_j,t_s} X_{v_{i'},n_{j'},t_s} + Z_{v_i,n_j,t_s} Z_{v_{i'},n_{j'},t_s}; \quad \forall\, (n_j, n_{j'}) \in \mathcal{L}, (v_i, v_{i'}) \in \mathcal{E}, t_s \in \mathcal{T}. \tag{4.18}$$

Given that Constraint (4.18) is not linear, we linearize it as follows[3]:

$$\sum_{(v_i,v_{i'}) \in \mathcal{E}} U^{n_j n_{j'},t_s}_{v_i v_{i'},t_s} = 1; \quad \forall\, (n_j, n_{j'}) \in \mathcal{L}, t_s \in \mathcal{T}. \tag{4.19}$$

$$U^{n_j n_{j'},t_s}_{v_i v_{i'},t_s} \geq Z_{v_i,n_j,t_s} + Z_{v_{i'},n_{j'},t_s} - 1; \quad \forall\, (n_j, n_{j'}) \in \mathcal{L}, (v_i, v_{i'}) \in \mathcal{E}, t_s \in \mathcal{T}. \tag{4.20}$$

Finally, we ensure that the bandwidth capacity and the maximum latency requirements of the substrate network imposed by the VNs are not exceeded. Thus, we define $\mathcal{H}' = (\mathcal{N}'_h, \mathcal{L}'_h)$ such that $\mathcal{H}' \subset \mathcal{H}, \therefore \mathcal{N}'_h \subset N$, $\mathcal{L}'_h \subset L$ and $(n_j, n_{j'}) \in \mathcal{L}'_h \mid n_j \in \mathcal{N}'_h$ and $n_{j'} \in \mathcal{N}'_h$. The bandwidth and latency requirements are met using the following constraints:

[3]Eq. (4.18) is a nonlinear equation. In this work we have considered a virtualizable substrate network. Accordingly, the virtual links in VNRs mapped to the physical sensors are linearized using Eq. (4.19) which enforces the links to be mapped, regardless of the type of the sensor (physical or virtual), this results in elimination of X, and Eq. (4.20) is applied to linearize those nodes' links in VNRs that are mapped to virtual sensors, avoiding redundant and duplicate mapping of the links.

$$\sum_{(n_j,n_{j'})\in\mathcal{L}'_h} B_h(n_j,n_{j'})U^{n_jn_{j'}}_{v_iv_{i'},t_s} \leq \min(B_g(v_i,v_{i'})) \quad \forall\,(v_i,v_{i'})\in\mathcal{E}, t_s\in\mathcal{T}, \tag{4.21}$$

and

$$\sum_{(n_j,n_{j'})\in\mathcal{L}'_h}\sum_{(v_i,v_{i'})\in\mathcal{E}} D_g(v_i,v_{i'})U^{n_jn_{j'}}_{v_iv_{i'},t_s} \leq D_h(n_j,n_{j'}) \quad \forall\, t_s\in\mathcal{T}, \tag{4.22}$$

where, $D_g(v_i,v_{i'})$ is given by:

$$D_g(v_i,v_{i'}) = D^p + D^v. \tag{4.23}$$

4.3. DNE: A Heuristic for Dynamic Network Embedding in Virtualized WSNs

In this section, after presenting the problem analysis, we explain our proposed solution.

4.3.1 Problem Analysis

Theorem 1. *The dynamic network embedding problem in Virtualized WSNs is an NP-hard problem and is at least as hard to approximate as a Maximum Stable Set Problem (MSSP).*

Proof. The problem can be proven to be NP-hard by a reduction to show that our network embedding problem in virtualized WSNs can be expressed as a known NP-hard problem. We use a Maximum Stable Set Problem (MSSP) that is known to be an NP-hard problem [112] under the assumption that the node deployment is given, as our reference problem. The proof is based on mapping our problem to the MSSP.

A stable set in a graph G is defined as "a subset of pairwise non-adjacent vertices of G and the Maximum Stable Set Problem (MSSP) is to find a stable set of G with maximum cardinality" [113]. In MSSP, the substrate network is defined as an undirected graph $G^0 = (V^0, E^0)$. V^0 and E^0 denote the substrate physical nodes and their corresponding links, respectively. Also, $B_i, i \in V^0$ and $K_{ij}, \{i,j\} \in E^0$ represent the substrate physical node and link capacities, respectively [112]. Every virtual network request $r \in R$ is represented by an undirected graph $G^r = (V^r, E^r)$ too, where $v \in V^r$ and $\{v,w\} \in E^r$ represent the virtual nodes and the virtual links, respectively. t^r_v denotes the node demand and d^r_{vw} represents the traffic demand. A positive integer k is defined to show whether G contains a stable set of cardinality as small as k [112]. It is stated that the virtual network request of G^r corresponding to physical node $i \in V^0$ at the substrate network is isomorphic, that is a star graph with $1 + |\sigma(i)|$ nodes that the central virtual node corresponds to the $i \in V^0$ and $|\sigma(i)|$ represents the virtual leaf nodes. Given the node and link capacity constraints, and under extreme locality constraints (the central virtual node $v \in V^r$ can only be mapped to the corresponding substrate node $i \in V^0$), if a central node in a VN is mapped to the corresponding $i \in V^0$, no other VN $r' \in R$ with the similar central node can be admitted. Thus, k VNs are simultaneously admitted [112].

Let us now consider an instance of the dynamic network embedding problem in a virtualized WSN to show that our problem can be reduced to MSSP. Considering our objective function defined in Eq. (4.11), we aim at minimizing the overall energy consumption while taking into account the energy consumption of transmission, CPU, wireless links, and the virtualization overhead. Our constraints on sensor node's capacities and available energy (given by Eqs. (4.14, 4.15, 4.16)) can be mapped to the constraint defined in MSSP concerning the node capacity. Moreover, our defined constraints on link bandwidth and delay (given by Eqs. (4.21, 4.22)) can also be mapped to the respective link capacity constraints defined in MSSP. It is evident that our developed formulation matches the MSSP problem, which is known as an NP-hard problem. Therefore, our network embedding problem in a virtualized WSN is an NP-hard problem.

4.3.2 DNE: Dynamic Network Embedding

Given the NP-hardness of the problem, we propose our so-called **D**ynamic **N**etwork **E**mbedding (DNE) heuristic to solve the problem for large-scale scenarios in a reasonable execution time. The proposed DNE heuristic aims to determine the embedding of the virtual networks onto the physical and virtual sensors and their associated communication links at the substrate WSN network. The main objective is to minimize the overall energy consumption while meeting the given SLAs (i.e., end-to-end latency and bandwidth requirements). The output of the proposed DNE heuristic is sets X, Z, and U of the obtained node/link assignments.

Algorithms 4.1 to 4.3 illustrate the pseudo-code of our proposed DNE heuristic, which comprises two main phases. In the first phase, which is referred to as the *node mapping* phase (Algorithm 4.1), we determine the assignment of the virtual nodes to the physical and virtual sensors. As specified in Algorithm 4.1, our proposed heuristic iterates over the set $\mathcal{T}$ of time slots followed by iterations over the set of substrate sensors and virtual nodes in VN (lines 3 to 5) [4]. Then, the heuristic explores the list of candidate virtual nodes within the sensing range of the sensors at the substrate network using *Dense()* function shown in Algorithm 4.2. Next, the sensors are sorted in a descending order based on the density of the virtual nodes in their sensing range (see lines 6 and 7 of Algorithm 4.1) using the TimSort [114] function. Similar to [115–117] we use the Euclidean Distance to create the list of candidate virtual nodes within the sensing range of the sensor nodes (see Algorithm 4.2). If there are multiple sensors with similar "virtual node" density, the algorithm checks the sensors' energies and selects the sensor with the highest available energy (see lines 10 to 14). Then, the heuristic assigns the virtual node to the physical sensor, if there is only one virtual node in the sensing range of the selected sensor, considering that the selected sensor complies with both capacity and energy constraints (lines 15 to 18) using Eq. (4.15). Next, the capacity of the sensor will be updated accordingly (line 19). If there is an unsatisfied condition, the next available sensor node will be selected (lines 20 to 22). If the condition in line (15) is not met and there are multiple virtual nodes within the sensing range of the sensor, the sensor will be used in its

[4] It should be noted that Algorithms 4.2 and 4.3 are the sub-algorithms of the main Algorithm 4.1, and they are part of the main loop initiated by block **for** $t_s \in \mathcal{T}$ as in line 3. We have indicated them in separate algorithms for better readability.

Algorithm 4.1 DNE Heuristic

Input: $\mathcal{V}, \mathcal{E}, \mathcal{N}, \mathcal{L}, \mathcal{T}$
Output: X, Z, U /* sets of assignment decisions */
$\bar{C}$= {cap(v_i) / $v_i \in \mathcal{V}$}; /* sensor node capacity */
$E_{t_s}^{v_i}$ = {e(v_i) / $v_i \in \mathcal{V}$}; /* sensor node energy */
for $t_s \in \mathcal{T}$ **do**
  for $v \in \mathcal{V}$ **do**
    for $n \in \mathcal{N}$ **do**
      $\hat{\mathcal{F}}$= {Dens(v, n)}; /* Algorithm 2: find candidate virtual nodes */
      $\hat{\mathcal{F}}_s$ = $\hat{\mathcal{F}}.sort()$;
      $v = 1$;
      while $v < \hat{\mathcal{F}}_s.size()$ **do**
        if ($\hat{\mathcal{F}}_s(v)$ = $\hat{\mathcal{F}}_s(v+1)$) & (E($\hat{\mathcal{F}}_s(v)$) > E($\hat{\mathcal{F}}_s(v+1)$)) **then**
          select sensor v;
        else
          select sensor $v+1$;
        end if
        if ($\hat{\mathcal{F}}_s(v) == 1$) & ($\mathcal{C}(v) > 0$) **then**
          if Eq. (4.15) holds **then**
            $X_{v_i,n_j,t_s=1}$;
            Calculate consumed energy using Eq. (4.5)
            $C(v) = \mathcal{C} - 1$; /* update capacity */
          else
            $v \leftarrow v + 1$
          end if
        else
          if Eq. (4.14) holds **then**
            if Eq. (4.16) holds **then**
              $Z_{v_i,n_j,t_s=1}$;
              calculate consumed energy using Eq. (4.6)
              $C(v) = \mathcal{C} - 1$;
            else
              $v \leftarrow v + 1$
            end if
          else
            $v \leftarrow v + 1$
          end if
        end if
        $v \leftarrow v + 1$;
      end while
    end for
  end for
  return X, Z;
  lmcc($v_i v_{i'}, n_j n_{j'}$); /* Algorithm 3: link mapping */
end for

Algorithm 4.2 Dens(): Task density around each sensor

Input: $\mathcal{V}$,$\mathcal{N}$
Output: $\hat{\mathcal{F}}$ /* sets of assignment decisions */
$\mathcal{F}$[][]; /* A matrix showing if a task is in the proximity of the sensor */
$\hat{\mathcal{F}}$[][]; /* A matrix that stores the sensor and the number of the tasks around it */
for $v \in \mathcal{V}$ **do**
  for $n \in \mathcal{N}$ **do**
    EuDist = $\sqrt{(x_{v_i} - x_{n_j})^2 + (y_{v_i} - y_{n_j})^2}$
    if $EuDist \leq SR$ **then**
      $\mathcal{F}[v_i][n_j] = 1$
      $\hat{\mathcal{F}}$.add((v_i).get(n_j))
    else
      $\mathcal{F}[v_i][n_j] = 0$
    end if
  end for
end for
return $\hat{\mathcal{F}}$

Algorithm 4.3 Link Mapping

1: ***lmcc***(v, n);
2: $P_1, P_2, ..., P_K = \mathcal{KSS}(v_i, v_{i'})$; /* k-shortest simple paths between $(v_i, v_{i'}) \in \mathcal{E}$ */
3: $\hat{P} = \mathcal{KSS}(n_j, n_{j'})$; /* k-shortest simple paths between $(n_j, n_{j'}) \in \mathcal{L}$ */
4: $\mathcal{D}_g(v_i, v_{i'})$; /* link latency of $(v_i, v_{i'}) \in \mathcal{E}$ */
5: $\mathcal{D}_h(n_j, n_{j'})$; /* link latency of $(n_j, n_{j'}) \in \mathcal{L}$ */
6: $\mathcal{B}_g(v_i, v_{i'})$; /* link bandwidth of $(v_i, v_{i'}) \in \mathcal{E}$ */
7: $\mathcal{B}_h(n_j, n_{j'})$; /* link bandwidth of $(n_j, n_{j'}) \in \mathcal{L}$ */
8: **while** $\hat{P}(n_j, n_{j'}) == 1$ **do**
9: **if** $P_K(v_i, v_{i'}) > 1$ **then**
10: **if** $\mathcal{D}_g(v_i, v_{i'})(P_1) < \mathcal{D}_h(n_j, n_{j'})(\hat{P})$ and $\mathcal{D}_g(v_i, v_{i'})(P_2) < \mathcal{D}_h(n_j, n_{j'})(\hat{P})$ (Eq. (4.22)) **then**
11: **if** $\mathcal{B}_g(v_i, v_{i'})(P_1) > \mathcal{B}_h(n_j, n_{j'})(\hat{P})$ and $\mathcal{B}_g(v_i, v_{i'})(P_2) > \mathcal{B}_h(n_j, n_{j'})(\hat{P})$ (Eq. (4.21)) **then**
12: Calculate communication energy using Eq. (4.7)
13: **if** $E^{comm}(P_1) < E^{comm}(P_2)$ **then**
14: $U_{v_i v_{i'}}^{n_j n_{j'}}(P_1) = 1$;
15: update sensors energies using Eq. (4.17)
16: **else**
17: $U_{v_i v_{i'}}^{n_j n_{j'}}(P_2) = 1$;
18: update sensors energies using Eq. (4.17)
19: **end if**
20: **else**
21: **if** $\mathcal{B}_g(v_i, v_{i'})(P_1) > \mathcal{B}_h(n_j, n_{j'})(\hat{P})$ and $\mathcal{B}_g(v_i, v_{i'})(P_2) < \mathcal{B}_h(n_j, n_{j'})(\hat{P})$ **then**
22: $U_{v_i v_{i'}}^{n_j n_{j'}}(P_1) = 1$;
23: update sensors energies
24: **else**
25: $U_{v_i v_{i'}}^{n_j n_{j'}}(P_2) = 1$;
26: update sensors energies
27: **end if**
28: **end if**
29: **else**
30: **if** $\mathcal{D}_g(v_i, v_{i'})(P_1) < \mathcal{D}_h(n_j, n_{j'})(\hat{P})$ and $\mathcal{D}_g(v_i, v_{i'})(P_2) > \mathcal{D}_h(n_j, n_{j'})(\hat{P})$ **then**
31: **if** $\mathcal{B}_g(v_i, v_{i'})(P_1) > \mathcal{B}_h(n_j, n_{j'})(\hat{P})$ **then**
32: $U_{v_i v_{i'}}^{n_j n_{j'}}(P_1) = 1$;
33: update sensors energies
34: **else**
35: $U_{v_i v_{i'}}^{n_j n_{j'}}(P_1) = 0$;
36: **end if**
37: **else**
38: $U_{v_i v_{i'}}^{n_j n_{j'}}(P_1) = 0$;
39: **end if**
40: **end if**
41: **else**
42: **if** $\mathcal{D}_g(v_i, v_{i'})(P_1) < \mathcal{D}_h(n_j, n_{j'})(\hat{P})$ and $\mathcal{B}_g(v_i, v_{i'})(P_1) > \mathcal{B}_h(n_j, n_{j'})(\hat{P})$ **then**
43: $U_{v_i v_{i'}}^{n_j n_{j'}}(P_1) = 1$;
44: update sensors energies
45: **else**
46: $U_{v_i v_{i'}}^{n_j n_{j'}}(P_1) = 0$;
47: **end if**
48: **end if**
49: **end while**
50: **return** U;

virtualized mode provided that it has sufficient capacity and energy (lines 24 to 27). If there is an unmet condition, the heuristic evaluates the next available sensor that can satisfy the requirements and then returns the list of selected physical and virtual sensors accordingly (lines 30 to 40). Given that WSNs are desired to operate for a long period of time, it is important to increase its lifetime. Although the main objective of the proposed algorithm is to minimize the total energy consumption at a given time slot, it implicitly takes into account the dynamics of the network (e.g., some sensor nodes run out of energy) by favoring the nodes with larger available energy in its allocation process, thus

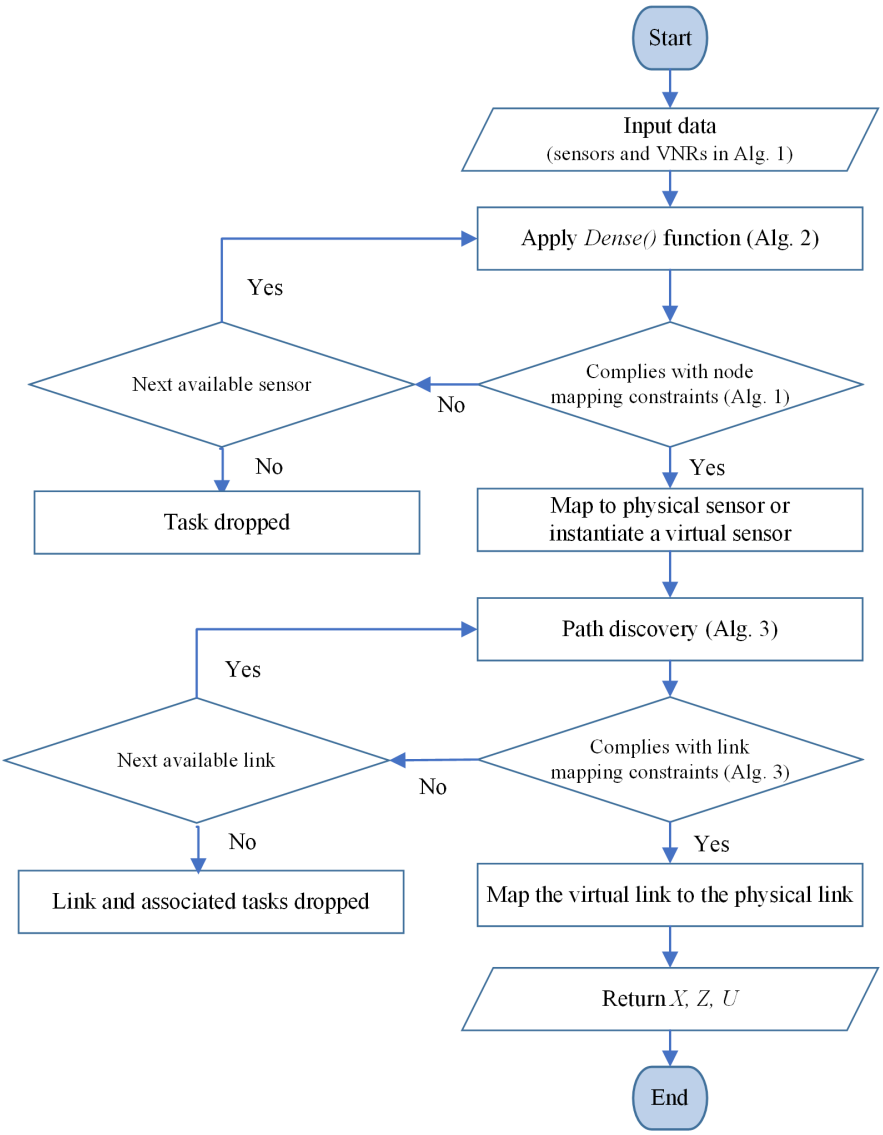


Figure 4.2: High-level flowchart of the proposed solution.

increasing the lifetime of the network.

The second phase of our proposed algorithm is called *link mapping*, which is shown in Algorithm 4.3. In this phase, the virtual links in the virtual network are mapped onto the physical links in the substrate WSN network. At first, the algorithm checks if the virtual nodes in the VN are connected. To this end, it checks to find k shortest paths at the substrate network for their connection (after being mapped considering the node-mapping phase). This is done using function $\mathcal{KSS}(v_1, v_2)$, which calculates K shortest paths $P_1, P_2, ..., P_K$ between nodes v_1 and v_2 (see lines 2 and 3 of Algorithm 4.3). Among the available k paths, if multiple paths comply with both end-to-end latency and bandwidth requirements, the heuristic calculates the consumed communication energy of each path using Eq. (4.7). Then, the path with the lowest energy consumption will be selected (see lines 9 to 19). Next, the sensors' energies will be updated

accordingly based on Eq. (4.17). If any of the explored paths violates either latency or bandwidth requirements, the heuristic selects the next available path (see lines 21 to 40). Finally, if there is only one available path between two sensors, the heuristic checks if that path complies with the required latency and bandwidth SLAs (see lines 42 to 48).

It is worthwhile to mention that although our dynamic embedding problem is developed and solved for each time slot (see Eq. (4.11)), different time slots are not completely independent, because the allocation decision made at a given time slot has a direct impact on the next time slot (see lines 9 to 19 in Algorithm 4.3). Given that our proposed DNE heuristic takes the remaining energy of the sensors as its input (according to Eq. (4.17)), it implicitly considers the integration between different time slots. For illustration, Fig. 4.2 depicts the high-level flowchart of the proposed algorithm.

4.3.3 Complexity Analysis

In the following, we present the complexity analysis of our proposed DNE heuristic. The proposed DNE heuristic consists of two main nested loops, a sorting operation using TimSort algorithm [114], *k-shortest path* algorithm [118] for path exploration, and verification of a series of linear constraints. The first nested loop goes through the set $\mathcal{V} = \{v_1, \ldots, v_M\}$ of sensors, while the second one explores in the set $\mathcal{N} = \{n_1, \ldots, n_R\}$. The time complexity of running through these nested loops is $\mathcal{O}(M \times R)$, which reduces to complexity $\mathcal{O}(R^2)$ given that $R > M$. The time complexity of the sorting operation is $\mathcal{O}(M \cdot \log M)$ [114], and the *k*-shortest-simple-path algorithm is associated with a time complexity of $\mathcal{O}(kM \cdot (M' + M \log M))$. Verification of the given constraints and updating of the sensor-related parameters (e.g., remaining energy) runs at a worse-case complexity of $\mathcal{O}(n)$. Therefore, given that the complexity of the nested loops dominates the other operations and the complexity of the routing algorithm dominates the sorting function, the overall complexity of the proposed DNE heuristic reduces to $\mathcal{O}(R^2) + \mathcal{O}(kM \cdot (M' + M \log M))$.

4.4. Results

In the following, we present our evaluation scenarios followed by the obtained results.

4.4.1 Evaluation Scenarios

We conducted our evaluations over various scenarios. Different numbers of sensors and tasks were considered, creating both small- and large-scale problem instances. Also, the system was examined under different values of virtualization overhead, transmission, and link energy consumption values forming both a homogeneous and a heterogeneous network. Table 4.3 summarizes our simulation scenarios. The sensor nodes are scattered in different geographical areas. The virtual network requests are generated by applications and served at the end of each time slot. We consider

Table 4.3: Evaluation scenarios.

	Small-scale Scenario	Large-scale Scenario
Area (m^2)	150 × 150	1000 × 1000
Total Number of Sensor Nodes	15	500
Total Number of Time Slots	5	5
Total Number of Tasks	300	10000
Sensing Range	30 m	30 m
Sensor Node's Initial Energy	2.9 J - 3.4 J	2.9 J - 3.4 J

a total of five time slots in each scenario. The number of time slots has been selected such that a statistical stability is achieved. Each task has a geographical location with specific latency and bandwidth requirements in each scenario. We follow the definitions of small- and large-scale scenarios given by Ref. [109], where small- and large-scale scenarios involve $<$100 and $>$100 sensor nodes, respectively.

We ran several simulations with different parameter settings considering both homogeneous and heterogeneous networks. In the homogeneous network, the deployed sensor network consists of sensors of the same type, each sensor consuming a fixed amount of energy for transmission, CPU, reception, and link. By contrast, in the heterogeneous network, the sensors are associated with different amounts of energy for different parameters. In addition, we also consider a heterogeneous network in terms of the number of virtualizable and non-virtualizable sensors. Our small-scale scenario is 150 m × 150 m with 15 sensors and a total of 300 tasks and our large-scale scenario involves 500 sensors in a 1000 m × 1000 m of area with a total of 10000 tasks.

In all scenarios, for each sensor, we consider a sensing range of 30 m [52] and a random initial energy between 1.9 J and 3.4 J [34]. Also, we set the energy consumption of a sensor while operating as a physical sensor to 0.017 mJ, which is the energy required for transmission [34]. The energy consumption overhead due to the creation of a virtual sensor over a physical sensor is set to 0.005 mJ [11]. Finally, we set the data reception energy at a sensor node to 0.031 mJ [34]. The energy consumption of CPU usage and wireless communication link are set to 0.005 mJ [79] and 0.007 mJ [119], respectively. The task's required data volume and link bandwidth are set to [100-200] bps and [200-400] kbps, respectively [48]. The energy consumption for storage and sensing is negligible. The transmission delay and virtual sensor creation overhead delay are set to 0.02 s and 0.06 s, respectively [11].

4.4.2 Evaluation Results

We present the optimal results obtained from CPLEX [110] and those obtained from our proposed DNE heuristic. We compare the performance of our proposed DNE algorithm with that of the RAS algorithm [48] as our benchmark. We selected the RAS [48] algorithm because (i) it is the only existing work in the literature that has considered virtualized WSN and (ii) similar to our work, it aims to optimize the overall energy consumption while considering CPU and transmission power of sensor nodes. Despite these commonalities, which make the RAS algorithm a suitable

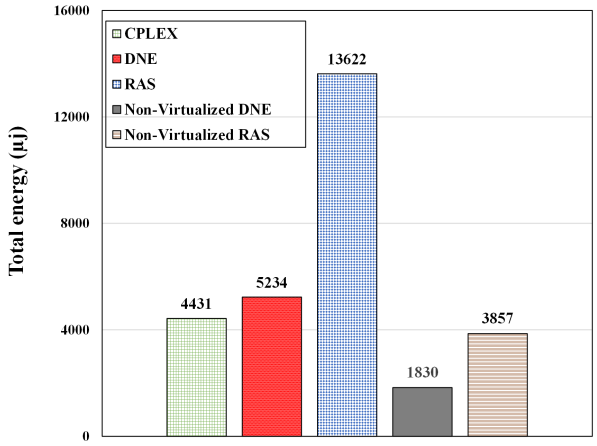
benchmark for this work, there are a few differences between this work and ours. For example, in [48], the authors considered energy harvesting as part of their objective function. Also, this work has used the so-called Leach algorithm for solving the routing sub-problem, while in our work, we use the k-shortest path algorithm. Nevertheless, we have adjusted the energy models and routing algorithms for a fair comparison.

In a small-scale scenario (i.e., scenario 1) we derive the results from CPLEX, the proposed DNE heuristic, and the RAS algorithm. However, for the large-scale problem, the results are only obtained from the proposed DNE heuristic and the RAS algorithm. This is mainly due to the large number of variables that are generated by the CPLEX engine. To better see this, we note that for 5 sensors and 25 tasks, the number of variables becomes $\sim$4500. As the number of nodes is increased to 10 and 15, the number of variables becomes $\sim$15000 and $\sim$30000, respectively. It is evident that the number of variables increases exponentially with response to the growing problem size, thus preventing the CPLEX to return the results in a computationally efficient manner. The mathematical model was implemented using IBM ILOG CPLEX IDE and the heuristics were implemented in Python. The simulations were run on a machine with 64-bit OS Windows 10 Pro, 3.20 GHz Intel ® Corei7-8700 ® CPU, and 16 GB of memory[5].

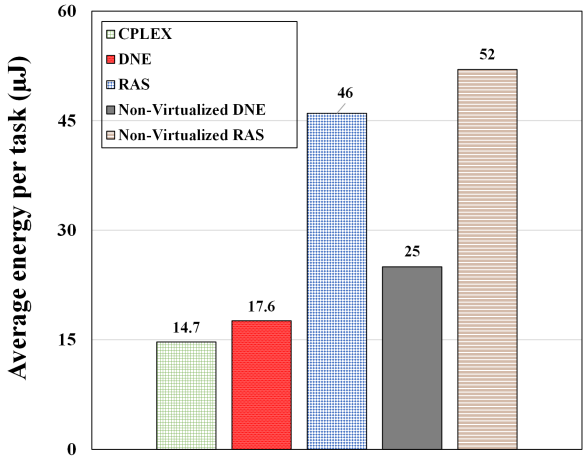
A) Energy Consumption in Small-Scale Scenario

First, we evaluate the energy efficiency performance of all algorithms under consideration in a small-scale scenario, comparing it to the RAS algorithm [48]. We consider both virtualized and non-virtualized variants of the algorithms. Figures 4.3(a) and 4.3(b) depict the total energy consumption and average energy consumption per task in a homogeneous network setting in our small-scale scenario, respectively. According to Fig. 4.3(a), the non-virtualized DNE and non-virtualized RAS benchmarks achieve smaller energy consumption compared to their virtualized counterparts. This was expected since in the non-virtualized solutions, the sensors are only capable of executing a single task at a time, dropping the rest of the tasks in the queue. The figure also reveals that the proposed DNE heuristic can achieve a near-optimal solution with an optimality gap of $\sim$18%, outperforming the RAS algorithm with 207% gap with the optimal solution. This happens mainly because of three main reasons. First, the RAS algorithm does not consider the virtualization overhead energy consumption. This is critical as the virtualization comes with an overhead, which definitely affects the total energy consumption, and therefore should be taken into account in the decision making process. Secondly, the mapping of the tasks to the sensors is critical. An inappropriate mapping of the tasks to the sensors may require activating new sensors in the substrate network, and consequently can lead to an inefficient resource utilization and excessive energy consumption. Third, unlike our proposed DNE heuristic, which selects the path with lowest energy consumption among k available shortest paths, in the RAS algorithm, a multi-hop routing is applied without explicitly considering the energy consumption.

[5]It is important to note that the existing network simulators (e.g. NS3) are not best suited for the implementation and validation of this work as they do not support virtualized WSN.



(a)



(b)

Figure 4.3: (a) Total energy consumption and (b) average energy consumption per task for different algorithms under study (homogeneous network setting, small-scale scenario).

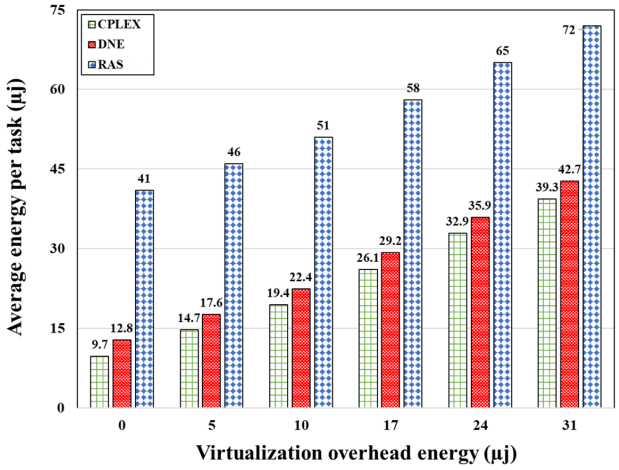
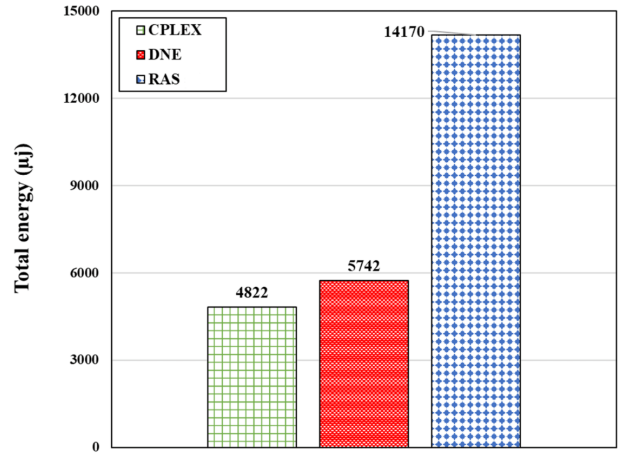


Figure 4.4: Total energy consumption vs. virtualization overhead energy (small-scale scenario).

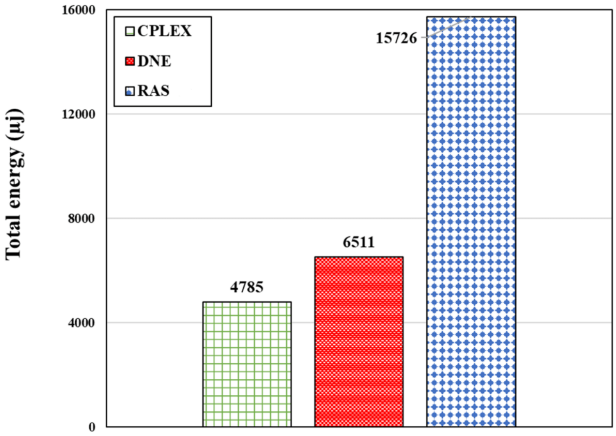
To further evaluate the energy performance in this scenario, we depict the average energy consumption per task in Fig. 4.3(b), where it is observed that the proposed DNE heuristic achieves an optimality gap of 19.7%. We also observe that the proposed DNE algorithm outperforms the RAS algorithm in terms of the average energy consumption per task in both virtualized and non-virtualized scenarios. Moreover, the virtualized DNE algorithm achieves 30% smaller average energy consumption per task compared to the non-virtualized DNE algorithm. This highlights the effectiveness of virtualization over a non-virtualized scenario. Given that the non-virtualized variants of both DNE and RAS algorithms perform relatively poorly, we will not include their results in the subsequent figures.

Figure 4.4 illustrates the average energy consumption per task vs. virtualization overhead energy in the small-scale scenario. Clearly, our proposed DNE heuristic achieves a near-optimal solution, outperforming the RAS algorithm. When the virtualization overhead energy becomes as high as 31μj, the proposed DNE heuristic and RAS algorithm achieve an optimality gap of 8.6% and 83.2%, respectively. Also, the proposed DNE heuristic outperforms the RAS algorithm for any given virtualization overhead energy. As virtualization overhead energy increases, the average energy consumption per task for the RAS algorithm increases linearly. This is because of the differences in assignment strategies of the proposed heuristic and the RAS algorithm to select the starting point. In the proposed DNE heuristic, the task assignment starts from the sensor with highest task density in its sensing range, thus maximizing the sensors' resource utilization and minimizing the communication energy consumption. By contrast, in the RAS algorithm, the task assignment and routing starts from the first available sensor and path.

Next, we evaluate the energy efficiency of the deployed sensors in a heterogeneous substrate network, where the sensors consume different amounts of energy for transmission and wireless communication link. We assess the energy efficiency of the proposed DNE heuristic in comparison with the CPLEX and RAS algorithm in those scenarios. Figures 4.5(a) and 4.5(b) depict the total energy consumption in the small-scale scenario for different algorithms under consideration with a random transmission energy consumption (selected between 17 μj and 31 μj for each sensor) and a random wireless communication link energy consumption (selected between 7 μj to 21 μj for each link). As shown in Fig. 4.5(a), the proposed DNE heuristic achieves a near-optimal solution with an optimality gap of $\sim$19% and outperforms the RAS algorithm by $\sim$59%. Similarly, according to the total energy consumption with different wireless communication link energies shown in Fig. 4.5(b), we notice that the proposed DNE heuristic achieves a near optimal result, outperforming the RAS algorithm by $\sim$58%. This is mainly due to the fact that in the RAS algorithm, the critical impact of the transmission energy is neglected, thus leading to a higher energy consumption. According to Fig. 4.5(b), the proposed solution achieves a $\sim$36% gap with respect to the global optimal solution, compared to the RAS algorithm with 228% gap.



(a)



(b)

Figure 4.5: Total energy consumption with (a) transmission energies selected from $[17, 31]$ μj per sensor node and (b) wireless communication link energies selected from $[7, 21]$ μj per link (heterogeneous network setting, small-scale scenario).

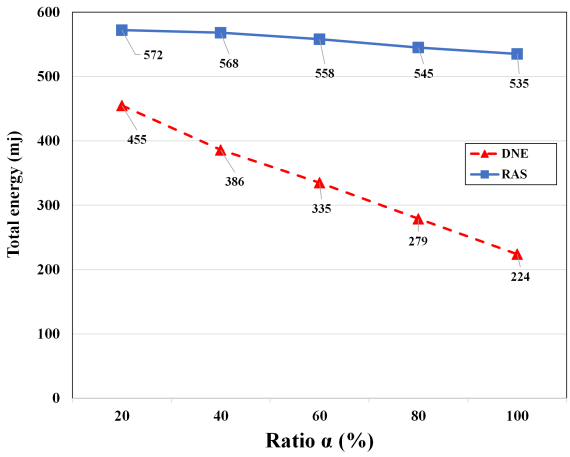


Figure 4.6: Total energy consumption in a heterogeneous network setting (large-scale scenario).

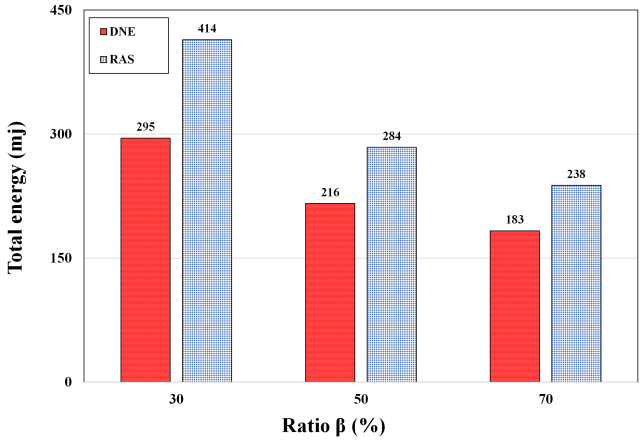


Figure 4.7: Total energy consumption with virtualizable and non-virtualizable sensors (large-scale scenario).

B) Energy Consumption in Large-Scale Scenarios

In the following, we evaluate the energy efficiency of the proposed DNE heuristic in a large-scale scenario and compare the obtained results with that of the RAS algorithm as shown in Figure 4.6. We define a parameter called α to denote the relative degree (in percentage) of connectivity of the network with respect to a complete graph. For example, $\alpha = 20\%$ represents a graph with 20% connectivity compared to a complete graph. The results indicate that by increasing the connectivity ratio α, the DNE heuristic outperforms the RAS algorithm in terms of energy consumption between $\sim$20% ($\alpha = 20\%$) and $\sim$58% ($\alpha = 100\%$), respectively. We also observe from the figure that the total energy consumption in our proposed DNE heuristic decreases at a higher rate for an increasing connectivity ratio α compared to the RAS algorithm. This is because increasing the network connectivity gives the proposed DNE heuristic more chance to find low-energy paths. To further evaluate the effectiveness of the proposed DNE heuristic, let us define a parameter called β to denote the ratio (in percentage) of the virtualizable sensors to the total number of sensors. Accordingly, $\beta = 30$ specifies that 30% of the sensors are non-virtualizable, which can only be used in the physical mode without the capability to create virtual sensors on top of them. Figure 4.7 depicts the total energy consumption vs. β. We observe from the figure that the energy consumption of both algorithms reduces as β increases. This was expected because of two main reasons. First, when there are more virtualizable sensors deployed, the possibility of having a larger number of tasks to be executed increases, thus leading to higher energy consumption. Second, virtualizing the sensors comes with an additional virtualization overhead energy, which may affect the total energy consumption.

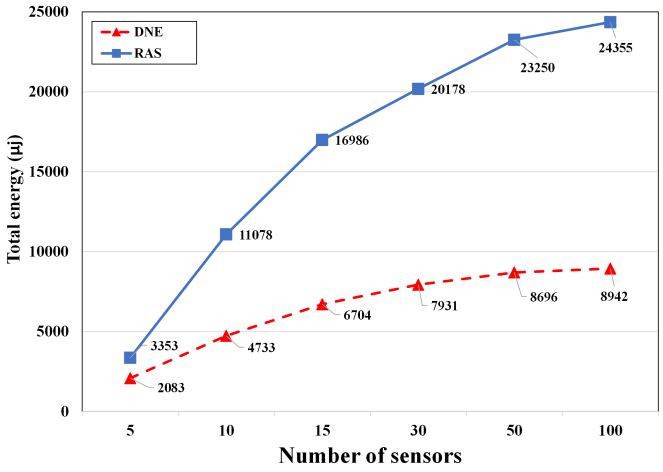


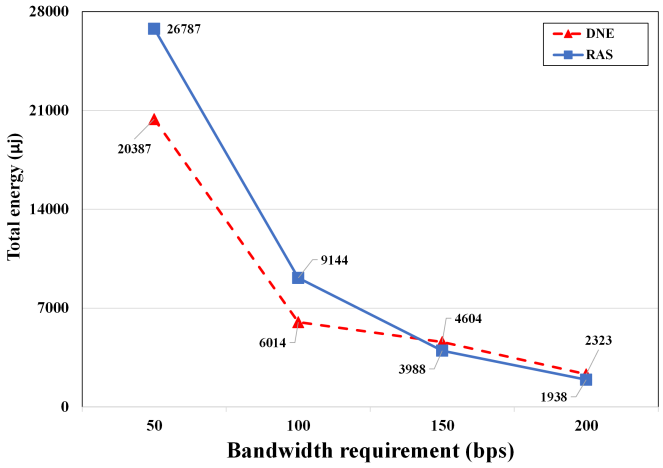
Figure 4.8: Total energy consumption vs. number of sensor nodes (with 500 tasks).

C) Energy Consumption for Different Application Network Parameters

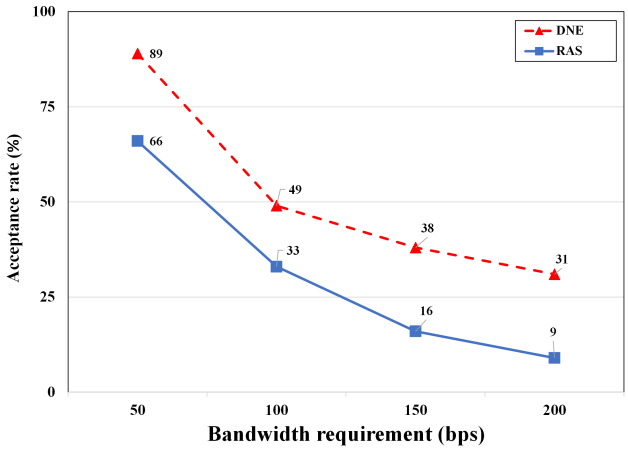
In the following, we assess the energy performance of the proposed DNE heuristic for different network parameters, including the number of sensors, bandwidth, and delay. Figure 4.8 depicts the total energy consumption vs. number of sensor nodes. We observe from the figure that when there are 5 sensor nodes, both proposed DNE heuristic and the RAS algorithm have a very small energy consumption. This happens because the number of sensors (i.e., 5) is too small compared to the number of tasks (i.e, 500). Clearly, the proposed DNE heuristic outperforms the RAS algorithm for any given number of sensors.

Next, we evaluate the impacts of bandwidth requirements on energy performance. Figure 4.9(a) depicts the total energy vs. bandwidth requirement. We observe from Fig. 4.9(a) that the total energy consumption of both DNE and RAS algorithms decrease as the bandwidth requirement increases. This is because increasing the bandwidth requirement leads to a decreased acceptance rate, which is shown in Fig. 4.9(b). According to Fig. 4.9(a), our proposed DNE heuristic outperforms the RAS algorithm in terms of total energy consumptin for bandwidth requirements < 150 bps. Even though the total energy consumption of the DNE heuristic becomes slightly more than the RAS algorithm for bandwidth requirements > 150 bps, its acceptance rate is significantly more than the RAS algorithm, see Fig. 4.9(b). According to Fig. 4.9(b), the DNE heuristic outperforms the RAS algorithm for any given bandwidth requirement.

Next, we examine the impact of delay requirement on the energy performance in Figure 4.10. According to Fig. 4.10(a), the total energy increases for both algorithms for an increasing delay requirement. The reason is that when the delay requirement increases, more requests are accepted, which require more energy. Importantly, we observe that the proposed DNE heuristic outperform the RAS algorithm in terms of total energy consumption for delay requirements > 120 ms. Relaxing the delay requirement leads to an increase of the acceptance rate in both algorithms under consideration, as shown in Fig. 4.10(b), but it is clear that the proposed DNE heuristic outperforms the RAS



(a)



(b)

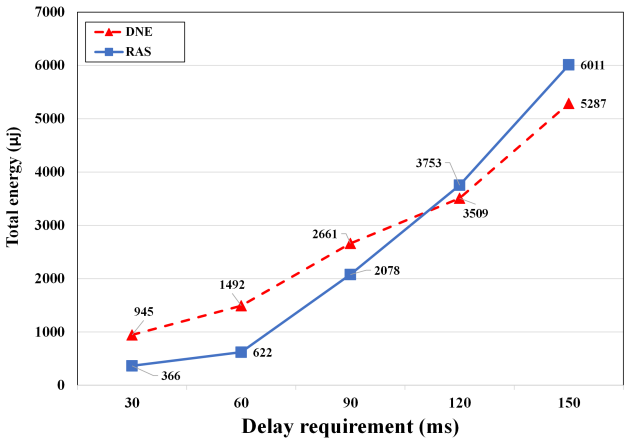
Figure 4.9: (a) Total energy consumption vs. bandwidth requirement and (b) acceptance rate vs. bandwidth requirement.

algorithm in terms of acceptance rate for any given delay requirement.

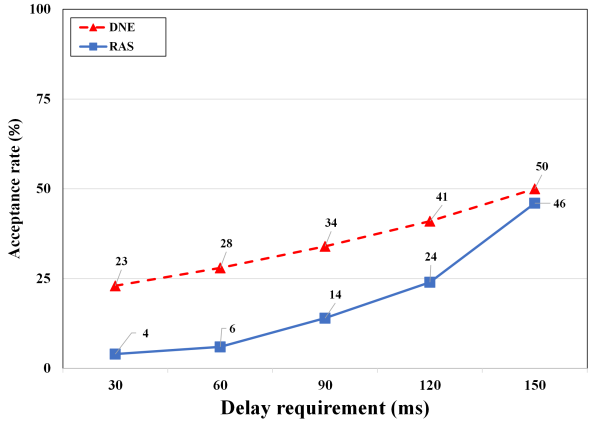
D) Execution Time

In the small-scale scenario, where there is a very small number of sensors/tasks, both the proposed DNE heuristic and the RAS algorithm perform quite similarly without much difference. More specifically, the proposed DNE heuristic and the RAS algorithm solve the problem in 31 ms and 36 ms, respectively, both outperforming the CPLEX solver, which solves the problem in 1300 ms. Although the RAS algorithm performs 5 ms faster than the proposed DNE heuristic, its achieved energy consumption is notably higher, as explained earlier in Figs. 4.3-4.7.

Finally, the execution time for the large-scale scenario comprises a denser network with thousands of tasks. Figure 4.11 depicts the execution time vs. the total number of tasks for both the proposed DNE heuristic and the RAS



(a)



(b)

Figure 4.10: (a) Total energy consumption vs. delay requirement and (b) acceptance rate vs. delay requirement.

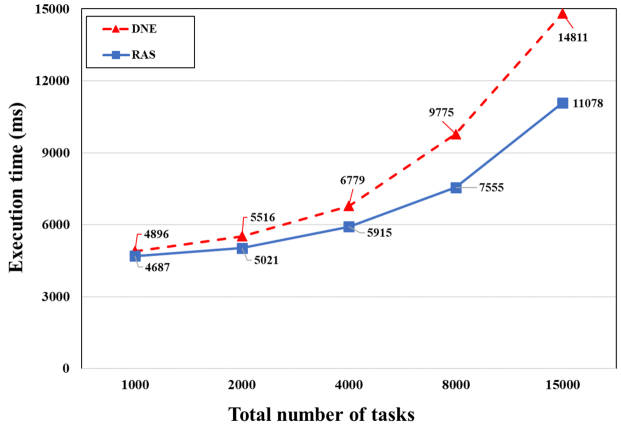


Figure 4.11: Execution time vs. total number of tasks (large-scale scenario).

algorithms. We observe from the figure that the RAS algorithm has a faster execution time compared to the proposed DNE heuristic, although both algorithms perform close to each other when the number of tasks is 1000 and 2000. The execution time for both algorithms increases gradually as the number of tasks increases. Clearly, the RAS algorithm solves the problem faster especially when the number of tasks is large (>8000 tasks). This demonstrates that the computational complexity of our proposed DNE heuristic is slightly larger than that of the RAS algorithm, as explained earlier in Section 4.3.3. Interestingly, the difference between the execution time of the proposed DNE heuristic and the RAS algorithm does not exceed 25%, which happens when the number of tasks is as high as 15000. Nevertheless, we note that the slight increase of the proposed DNE heuristic is outweighed by its beneficial impacts in terms of energy consumption in both small- and large-scale scenarios.

4.5. Conclusions

In this chapter, the problem of dynamic virtual network embedding in virtualized wireless sensor networks by considering both node- and network-level virtualization. We modeled the problem as an ILP to allocate the pool of physical and virtual sensors to the requested applications at the minimum energy consumption. We considered the energy consumption of transmission (forwarding), CPU, wireless communication links, and virtual sensor instantiation overhead as part of the total energy consumption. Furthermore, we defined the SLA constraints, including meeting the given latency and bandwidth requirements. Also, we proposed our heuristic to solve the problem. We conducted extensive simulations to evaluate our proposed heuristic against the optimal (CPLEX) solution and also a solution from the literature (RAS). The evaluations were conducted over both small- and large-scale scenarios. Our results indicate that the proposed DNE heuristic can achieve a near-optimal solution in terms of energy consumption while respecting the given SLA requirements.

Chapter 5

Energy Efficient Distributed Task Assignment in Virtualized Internet of Things [1]

5.1. Introduction

Concerning the relevant constraints of the WSNs, many of the proposed solutions run in a centralized manner at a resource-rich node outside of the WSN. The main advantage is that it does not induce any additional energy consumption on the sensor nodes, especially given that it does not involve any communication and processing inside the WSN. However, the downside is that there might be no resource-rich node available outside of the network in scenarios such as disaster management. This limitation can be partially overcome by having a fixed node carried by a relief vehicle [120] for discovering and connecting to the working base stations. Nevertheless, having a distributed mechanism in WSNs remains a challenge. Unlike the centralized algorithm, the distributed algorithm needs to be executed inside the WSN over the sensor nodes, eventually requiring the WSN network and interacting modules dedicated to each sensor. Hence, designing a distributed mechanism to assign the sensing tasks to the physical and virtual sensors while satisfying the required QoS (i.e., E2E latency, bandwidth, and task deadlines) becomes critical. This challenge gets more complicated considering that the WSN/IoT nodes are resource-constrained devices with limited available energy.

The rest of this chapter is organized as follows. First, it presents the system model, followed by the description of the game definition and the problem formulations. Then, it discusses the designed heuristic which considers a

[1] This chapter is based on a paper under revise for possible publication at IEEE TNSM: [5] V. M. Raee, A. Ebrahimzadeh, Z. Mlika, and R. Glitho, "Energy efficient distributed task assignment for virtualized internet of things," under revise for submission to IEEE Transactions on Network and Service Management (TNSM), 2023.

virtualized IoT network and the required QoS parameters. After that, it presents the simulation parameters and settings followed by the validation results. We will conclude this chapter at the end.

5.2. System Model and Game Formulation

In this section, we present our system model that covers the problem definition, followed by the game definition and formulation. We develop our formulation by applying non-cooperative game theory and defining the players, actions, and utility function that are explained in-depth technically in Section 5.2.1.

5.2.1 System Model

We consider an IoT network with node-level virtualization capabilities enabled, i.e., a WSN infrastructure deployed with both virtualizable and non-virtualizable sensor nodes. The physical sensors with virtualization capabilities allow the creation of virtual sensors as per application sensing task request, having multiple applications' sensing tasks be executed on top of the same physical sensor concurrently. It is important to note that applications' sensing tasks (e.g., temperature, humidity) require reading from a sensor node [105]. This is different from computing tasks, where additional processing may be required by the IoT nodes prior to transmission of the results. Virtualizable sensors may operate in either physical or virtual mode. If a virtualizable sensor node operates in the physical mode, its virtualization functionality is disabled. Yet, the non-virtualizable physical sensors (i.e., sensors with no virtualization capabilities) execute a single task at a time. The system includes a set of time slots. Application requests arrive in batches at the beginning of each time slot. Applications (e.g., fire contour) send the sensing tasks to the IoT network to be executed in a distributed manner. The objective is to assign the requested sensing tasks to the available IoT infrastructure network at minimal energy consumption. Once a sensing task is assigned to a sensor node (physical or virtual), it will be executed immediately.

Let $\mathcal{T} = \{t_1, t_2, ..., t_k\}$ denote the set of time slots, where each time slot $t_k \in \mathcal{T}$ corresponds to the task assignment time frame at t_k. The sensing task requests are generated by applications randomly and then served by the physical and virtual sensors at the end of each time slot t_k. Let α_k be the size of the sensing task request's set at time slot t_k. We model the task assignment in vIoT infrastructure as a non-cooperative game with the given bandwidth, E2E latency, and task deadline constraints. The main motivation behind applying game theory is that it allows for modeling the interactions between the sensor nodes mathematically. Accordingly, based on those interactions, the sensor nodes may do decision-making to optimize their performance (i.e., energy consumption in our study) considering the network utility function. In addition, there may exist conflicts between the sensor nodes in executing the application sensing tasks. This becomes more critical when dealing with a network with limited energy, capacity and computing resources and having application sensing tasks with stringent bandwidth and latency requirements. Therefore, a game-based

modeling can help design distributed solutions where players make decisions that are made interdependently from each other. First, we define the game by introducing the following aspects: (i) players, (ii) actions, (iii) vIoT infrastructure network model, and (iv) application sensing task model. Then, we define the utility function of the game.

5.2.2 Game Definition

A) Players

Each sensor is considered a player. The players are divided into two disjoint subsets: the set of physical players (i.e., non-virtualizable sensors) and the set of virtual players (i.e., virtualizable sensors). We denote the set of players by $\mathcal{P} = \mathcal{P}_p \cup \mathcal{P}_q$, where $\mathcal{P}_p$ and $\mathcal{P}_q$ represent the set of physical and virtual players, respectively.

B) Actions

An action profile $\mathbf{a} = (\mathbf{a}_{p_i}, \mathbf{a}_{p_{-i}})$, is a tuple where action $\mathbf{a}_{p_i}$ is the action of player p_i and $\mathbf{a}_{p_{-i}}$ is the actions of all other players. Let us assume that $\mathcal{F}$ is the set of application sensing tasks. The action set for player p_i is $\mathcal{A}_{p_i} = 2^{\mathcal{F}}$ (i.e., all subsets of the tasks in $\mathcal{F}$). If the player $p_i \in \mathcal{P}_p$, then for each $\mathbf{a}_{p_i} \in \mathcal{A}_{p_i}$, we have $|\mathbf{a}_{p_i}| \leq 1$ (i.e., non-virtualizable sensors can choose either one task or none). When necessary, we add a superscript t_k to each action $\mathbf{a}$ to denote that the action is taken at time slot t_k.

C) vIoT Networking Infrastructure Model

Let $\mathcal{G} = (\mathcal{P}, \mathcal{E})$ be a weighted undirected graph representing the vIoT networking infrastructure, where $\mathcal{G} = \mathcal{G}_p \cup \mathcal{G}_q$. The players represent the physical and virtual sensors. Accordingly, $\mathcal{G}_p = (\mathcal{P}_p, \mathcal{E}_p)$ and $\mathcal{G}_q = (\mathcal{P}_q, \mathcal{E}_q)$ represent the weighted undirected graphs of physical and the virtual players, respectively. Let $\mathcal{P}_p$ and $\mathcal{E}_p$ denote the physical players and their associated physical links. Also, the virtual players instantiated on top of the physical players, and their logical connections to the physical players are denoted by $\mathcal{P}_q$ and $\mathcal{E}_q$, respectively. Accordingly, $\mathcal{P} = \mathcal{P}_p \cup \mathcal{P}_q = \{p_1, \ldots, p_M\}$ is the set of players, with each player $p \in \mathcal{P}$ representing a physical or a virtual player instantiated on top of the physical player with x_{p_i} and y_{p_i} representing the player's coordination in the grid. Set $\mathcal{E} = \{e_1, \ldots, e_{M'}\}$ represents the set of edges between the players, where $\mathcal{E} = \mathcal{E}_p \cup \mathcal{E}_q$. Edge $(p_i, p_{i'}) \in \mathcal{E} \mid p_i \neq p_{i'}$, represents the communication link between players p_i and $p_{i'}$. Also, let $D_g(p_i, p_{i'})$ and $B_g(p_i, p_{i'})$ represent the latency and bandwidth associated with edge $(p_i, p_{i'})$, respectively.

D) Application Sensing Task Model

Let $\mathcal{F} = \{f_1, \ldots, f_{f'}\}$ be the set of application sensing tasks, which needs to be assigned. We let the weighted directed graph $\mathcal{H} = (\mathcal{N}, \mathcal{L})$ represent a given sensing task. $\mathcal{N}$ is the set of tasks defined as $\mathcal{N} = \{n_1, \ldots, n_R\}$ and $\mathcal{L} =$

Table 5.1: General notations of the problem.

Notation	Definition
$\mathcal{G} = (\mathcal{P}, \mathcal{E})$	vIoT infrastructure network graph with players $\mathcal{P}$ and edges $\mathcal{E}$ linking them
$\mathcal{G}_p = (\mathcal{P}_p, \mathcal{E}_p)$	vIoT infrastructure network graph with physical players $\mathcal{V}_p$ and edges $\mathcal{E}_p$ linking them
$\mathcal{G}_q = (\mathcal{P}_q, \mathcal{E}_q)$	vIoT infrastructure network graph with virtual players $\mathcal{V}_q$ and logical links $\mathcal{E}_q$ linking them to physical players which they instantiated over
$p \in \mathcal{P}$	A player (physical/virtual) in $\mathcal{P}$
x_{p_i}, y_{p_i}	Player p_i coordination in a grid
$E_{t_k}^{p_i}$	Current available energy of player p_i at t_k
$(p_i, p_{i'}) \in \mathcal{E}$	An edge in $\mathcal{E}$
$D_g(p_i, p_{i'})$	Latency associated with edge $(p_i, p_{i'})$
$B_g(p_i, p_{i'})$	Bandwidth associated with edge $(p_i, p_{i'})$
$\mathcal{H} = (\mathcal{N}, \mathcal{L})$	Application sensing task graph with tasks $\mathcal{N}$ and logical links $\mathcal{L}$ linking them
$n \in \mathcal{N}$	A task in $\mathcal{N}$
$(n_j, n_{j'}) \in \mathcal{L}$	A logical link in $\mathcal{L}$
x'_{n_j}, y'_{n_j}	Task n_j coordination in a grid
$D_h(n_j, n_{j'})$	Latency associated with logical link $(n_j, n_{j'})$
$B_h(n_j, n_{j'})$	Bandwidth associated with logical link $(n_j, n_{j'})$
$arr_{n_j}^{t_k}$	Task n_j arrival time
$dlt_{n_j}^{t_k}$	Task n_j deadline
$\mathcal{T}$	Total number of time slots
t_k	Task assignment frame time
α_k	Size of each group of sensing task request
C_{p_i}	Capacity of player p_i
$E_{p_i}^{tr}$	Forwarding energy consumption of player p_i
$E_{p_i}^{CPU}$	CPU energy consumption of player p_i
E^{ov}	Virtual player creation energy consumption
$E_{p_i}^{l}$	Wireless link energy consumption
$D_{p_i}^{tr}$	Transmission delay
D^{ov}	Virtual player creation delay
$\mathcal{F}$	Set of application sensing tasks

$\{l_1, \ldots, l_{R'}\}$ denotes the logical links between tasks n_j and $n_{j'}$ in the application set, where $(n_j, n_{j'}) \in \mathcal{L} \mid n_j \neq n_{j'}$. Let x'_{n_j}, y'_{n_j}, and $arr_{n_j}^{t_k}$ denote the requested sensing task's coordination in the grid and its arrival time, respectively. The given QoS requirements include latency $D_h(n_j, n_{j'})$, bandwidth $B_h(n_j, n_{j'})$, and task deadline $dlt_{n_j}^{t_k}$.

Tables 5.1 and 5.2 summarize the problem main notations and problem inputs, respectively.

5.2.3 Energy Model and Constraints

In the following, we formulate the energy model to use in the game formulation. Based on that, we define the utility function and associated constraints for the dynamic distributed vIoT task assignment problem using the non-cooperative game model. Accordingly, the player selects its strategy to minimize the overall energy while respecting the delay and bandwidth constraints.

Table 5.2: Problem inputs.

Input	Definition
$U_{p_i}^{t_k}$	Overall energy consumption of task assignment at time t_k
E_{p_i,t_k}^{P}	Energy consumption of physical player p_i at time t_k
E_{p_i,t_k}^{V}	Energy consumption of virtual player p_i at time t_k
$E_{t_k}^{comm}$	Energy consumption of wireless communication at time t_k
D_{p_i,t_k}^{P}	Delay of physical player p_i at time t_k
D_{p_i,t_k}^{V}	Delay of virtual player p_i at time t_k

A) Energy Model

The overall energy consumption is achieved by three components:

- **Physical Player Energy** (E_{p_i,t_k}^{P})**:** It represents the energy consumption of executing task $n_j \in \mathcal{N}$ on physical player $p_i \in \mathcal{P}_p$ at time t_k. When player p_i chooses action $\mathbf{a}_{p_i}^{t_k}$ at time slot t_k, its energy consumption is calculated as:

$$E_{p_i,t_k}^{P}(\mathbf{a}_{p_i}^{t_k}) = \sum_{n_j \in \mathbf{a}_{p_i}^{t_k}} (E_{p_i}^{tr} + E_{p_i}^{\text{CPU}}), \tag{5.1}$$

and the energy consumption of all physical players is given by:

$$E_{t_k}^{P}(\mathbf{a}^{t_k}) = \sum_{p_i \in \mathcal{P}} E_{p_i,t_k}^{P}(\mathbf{a}_{p_i}^{t_k}), \tag{5.2}$$

where $E_{p_i}^{tr}$ and $E_{p_i}^{\text{CPU}}$ are transmission and CPU energy consumption of player p_i, respectively.

- **Virtual Player Energy** (E_{p_i,t_k}^{V})**:** It accounts for the energy consumption of executing task $n_j \in \mathcal{N}$ on the virtual player $p_i \in \mathcal{P}_q$ instantiated over the physical player at time t_k, considering the energy consumption of transmission, CPU, and virtualization overhead. When player $p_i \in \mathcal{P}_q$ chooses action $\mathbf{a}_{p_i}^{t_k}$, then we have two cases: either the player has chosen to be virtualized (i.e., $|\mathbf{a}_{p_i}^{t_k}| > 1$) or not (i.e., $|\mathbf{a}_{p_i}^{t_k}| \leq 1$). Depending on the action chosen by the virtual player at time t_k, the energy consumption is obtained as follows:

$$E_{p_i,t_k}^{V}(\mathbf{a}_{p_i}^{t_k}) = \mathbb{1}_{\{|\mathbf{a}_{p_i}^{t_k}|>1\}}\Big[E_{p_i}^{tr} + \sum_{n_j \in \mathbf{a}_{p_i}^{t_k}} (E^{ov} + E_{p_i}^{\text{CPU}})\Big] + \mathbb{1}_{\{|\mathbf{a}_{p_i}^{t_k}|\leq 1\}}\Big[\sum_{n_j \in \mathbf{a}_{p_i}^{t_k}} (E_{p_i}^{tr} + E_{p_i}^{\text{CPU}})\Big], \tag{5.3}$$

and the energy consumption of all virtual players is given by:

$$E_{t_k}^{V}(\mathbf{a}^{t_k}) = \sum_{p_i \in \mathcal{P}} E_{p_i,t_k}^{V}(\mathbf{a}_{p_i}^{t_k}), \tag{5.4}$$

where, E^{ov} is virtualization energy overhead.

- **Energy Consumption of Wireless Communication Links (**$E_{t_k}^{comm}$**):** It includes the energy consumption of

the wireless links of the infrastructure network by mapping the logical communication link $(n_j, n_{j'}) \in \mathcal{L}$ to the wireless link $(p_i, p_{i'}) \in \mathcal{E}$ at time t_k:

$$E^{comm}_{t_k}(\mathbf{a}^{t_k}) = \sum_{(p_i,p_{i'})\in\mathcal{E}} \sum_{\substack{(n_j,n_{j'})\in\mathcal{L} \\ n_j\in\mathbf{a}^{t_k}_{p_i}, n_{j'}\in\mathbf{a}^{t_k}_{p_{i'}}}} E^{l}_{p_i,p_{i'}}, \tag{5.5}$$

where, $E^{l}_{p_i,p_{i'}}$ is the wireless link energy consumption of player p_i.

B) Delay Model

Before describing the problem constraints, in the following, we define the delay parameters. We consider the following two different delay components.

- **Physical Player Delay** $(D^{P}_{p_i,t_k})$**:** It is the delay incurred by executing task $n_j \in \mathcal{N}$ on the physical player $p_i \in \mathcal{P}_p$ at time t_k. Considering that player $p_i \in \mathcal{P}_p$ chooses action $\mathbf{a}^{t_k}_{p_i}$ at time slot t_k, the delay is calculated as:

$$D^{P}_{p_i,t_k}(\mathbf{a}^{t_k}_{p_i}) = \sum_{n_j\in\mathbf{a}^{t_k}_{p_i}} D^{tr}_{p_i}, \tag{5.6}$$

and the delay of all physical players is given by:

$$D^{P}_{t_k}(\mathbf{a}^{t_k}) = \sum_{p_i\in\mathcal{P}} D^{P}_{p_i,t_k}(\mathbf{a}^{t_k}_{p_i}), \tag{5.7}$$

where $D^{tr}_{p_i}$ is the transmission delay of player p_i.

- **Virtual Player Delay** $(D^{V}_{p_i,t_k})$**:** It represents the delay of executing the task $n_j \in \mathcal{N}$ on the virtual player $p_i \in \mathcal{P}_q$ instantiated over the physical player at time t_k. It can be calculated as follows:

$$D^{V}_{p_i,t_k}(\mathbf{a}^{t_k}_{p_i}) = \mathbb{1}_{\{|\mathbf{a}^{t_k}_{p_i}|>1\}}\Big[D^{tr}_{p_i} + \sum_{n_j\in\mathbf{a}^{t_k}_{p_i}} D^{ov}\Big] + \mathbb{1}_{\{|\mathbf{a}^{t_k}_{p_i}|\leq 1\}}\Big[\sum_{n_j\in\mathbf{a}^{t_k}_{p_i}} D^{tr}_{p_i}\Big], \tag{5.8}$$

and the delay of all virtual players is given by:

$$D^{V}_{t_k}(\mathbf{a}^{t_k}) = \sum_{p_i\in\mathcal{P}} D^{V}_{p_i,t_k}(\mathbf{a}^{t_k}_{p_i}), \tag{5.9}$$

where, D^{ov} is virtualization delay overhead.

C) Constraints

Next, we define the constraints of our problem. First, we have to ensure that the players will not take similar actions. In other words, a task must only be executed once:

$$(\mathbf{a}_{p_i}^{t_k}) \cap (\mathbf{a}_{p_{i'}}^{t_k}) = \emptyset, \quad \forall\, p_i, p_{i'} \in \mathcal{P}, t_k \in \mathcal{T}. \tag{5.10}$$

A task should not be assigned to a physical or virtualized player before its arrival time. We enforce this constraint as follows:

$$\mathbf{a}_{p_i}^{t_k} = \emptyset, t_k < arr_{n_j}^{t_k}, \ \ \forall\, p_i \in \mathcal{P}, n_j \in \mathcal{N}, t_k \in \mathcal{T}. \tag{5.11}$$

Furthermore, a virtualized player can support a maximum of C_{v_i} tasks:

$$|\mathbf{a}_{p_i}^{t_k}| \leq C_{p_i}, \ \ \forall\, p_i \in \mathcal{P}, t_k \in \mathcal{T}. \tag{5.12}$$

A task must not be assigned to a player if the player does not have enough available energy as shown in Eq. (5.13) and Eq. (5.14):

$$E_{p_i,t_k}^{P}(\mathbf{a}_{p_i}^{t_k}) \leq E_{t_k}^{p_i}, \ \ \forall\, p_i \in \mathcal{P}, t_k \in \mathcal{T}. \tag{5.13}$$

$$E_{p_i,t_k}^{V}(\mathbf{a}_{p_i}^{t_k}) \leq E_{t_k}^{p_i}, \ \ \forall\, p_i \in \mathcal{P}, t_k \in \mathcal{T}. \tag{5.14}$$

$$E_{t_k}^{p_i} = E_{t_{k-1}}^{p_i} - E_{p_i,t_k}^{P}(\mathbf{a}_{p_i}^{t_k}) - E_{p_i,t_k}^{V}(\mathbf{a}_{p_i}^{t_k}) - E_{t_k}^{comm}(\mathbf{a}_{p_i}^{t_k}). \tag{5.15}$$

where $E_{t_{k-1}}^{p_i}$ denote the available energy and of the player p_i, at time slot t_{k-1}. Also, the tasks should be executed by their defined deadlines, as follows:

$$arr_{n_j}^{t_k} \leq t_k < dlt_{n_j}^{t_k}, \ \ \forall\, p_i \in \mathcal{P}, n_j \in \mathbf{a}_{p_i}^{t_k}, t_k \in \mathcal{T}. \tag{5.16}$$

Finally, we have to ensure that the bandwidth and E2E latency requirements are satisfied. We define $\mathcal{H}' = (\mathcal{N}'_h, \mathcal{L}'_h)$ such that $\mathcal{H}' \subset \mathcal{H}, \therefore \mathcal{N}'_h \subset N$, $\mathcal{L}'_h \subset L$ and $(n_j, n_{j'}) \in \mathcal{L}'_h \mid n_j \in \mathcal{N}'_h$ and $n_{j'} \in \mathcal{N}'_h$. We enforce the bandwidth and latency requirements as follows:

$$\sum_{\substack{(n_j,n_{j'})\in\mathcal{L}'_h \\ n_j\in\mathbf{a}_{p_i}, n_{j'}\in\mathbf{a}_{p_{i'}}}} B_h(n_j, n_{j'}) \leq \min(B_g(p_i, p_{i'})) \quad \forall\, (p_i, p_{i'}) \in \mathcal{E}, t_k \in \mathcal{T}. \tag{5.17}$$

and

$$\sum_{\substack{(n_j,n_{j'})\in\mathcal{L}'_h \\ n_j\in\mathbf{a}_{p_i},n_{j'}\in\mathbf{a}_{p_{i'}}}} D_h(n_j,n_{j'}) \geq D_g(p_i,p_{i'}) \quad \forall\,(p_i,p_{i'})\in\mathcal{E}, t_k\in\mathcal{T}. \tag{5.18}$$

where $D_g(p_i,p_{i'})$ is given by:

$$D_g(p_i,p_{i'}) = D^P_{t_k}(\mathbf{a}^{t_k}) + D^V_{t_k}(\mathbf{a}^{t_k}). \tag{5.19}$$

D) Total Energy and Utility Function

It represents the payoff of each player p_i (common-payoff) which calculates the total energy consumption for player p_i. It is given by the summation of the above-mentioned three energy components as follows:

$$U^{t_k}_{p_i}(\mathbf{a}^{t_k}) = \begin{cases} E^P_{t_k}(\mathbf{a}^{t_k}) + E^V_{t_k}(\mathbf{a}^{t_k}) + E^{comm}_{t_k}(\mathbf{a}^{t_k}), & \text{if no violation,} \\ \lambda, & \text{otherwise, violation.} \end{cases} \tag{5.20}$$

In our problem, we strive for providing the decision to assign the sensing tasks on the vIoT infrastructure network at the lowest energy consumption while respecting the task deadline, end-to-end latency, and bandwidth requirements.

The overall utility function is given as follows:

$$\forall p_i\in\mathcal{P},\ \min U^{t_k}_{p_i}(\mathbf{a}^{t_k}) = \min U^{t_k}_{p_i}(\mathbf{a}^{t_k}_{p_i},\mathbf{a}^{t_k}_{p_{-i}}), \tag{5.21}$$

which aims to minimize the summation of energies consumed by physical players, virtual players, and communication links. In this work, the payoff function depends on the action of each player. If the player p_i violates any of the defined constraints or takes an action a_i which may degrades the $U^{t_k}_{p_i}(\mathbf{a}^{t_k}_{p_i})$, we can associate a large number λ as a penalty to the player p_i. Otherwise, the action a_i for player p_i leads to the payoff given in (5.20), which is a common-payoff function for all other players.

5.2.4 Nash Equilibrium Analysis

In this section we analyze the existence of pure Nash Equilibrium.

Definition 1. *Pure Nash Equilibrium (PNE).* A PNE can be defined as an action profile $\mathbf{a}^* = (\mathbf{a}^*_{p_i},\mathbf{a}^*_{p_{-i}})$ such that for all $\mathbf{a}'_{p_i}$ in the action set of the player p_i, we have $U_{p_i}(\mathbf{a}^*_{p_i},\mathbf{a}^*_{p_{-i}}) \geq U_{p_i}(\mathbf{a}'_{p_i},\mathbf{a}^*_{p_{-i}})$ [121].

Definition 2. *Best Response (BR).* The best response can be defined as an action profile that produces the most favorable outcome for a player, given other players' actions [121].

Theorem 1. *The derived game theory formulation discussed in Sec. 5.2.3 (part C) can be formulated as a distributed ILP problem that admits at least one solution. Hence, the derived game admits at least one Nash Equilibrium.*

Proof. The problem in Sec. 5.2.3 (part C) describes the minimization of overall energy consumption of a set of players in the game while meeting the E2E latency, bandwidth, and task deadline constraints. In addition, each player's decision or strategy is dependent on the strategy/decision taken by the other players in its communication proximity. To prove the theorem, We reformulate the problem as an Integer Linear Programming (ILP) problem to show its convexity and demonstrate the existence of at least one solution in the proposed optimization formulation. Accordingly, we define four decision variables: X_{p_i,n_j,t_k} is equal 1, if the task n_j is assigned to the player p_i on its physical mode and 0, otherwise. Y_{p_i,t_k} equals to 1 specifies that if the player p_i is virtualized at time t_k, and 0, otherwise. Z_{p_i,n_j,t_k} is equal 1, indicates that if the task n_j is assigned to the player p_i on its virtualized mode, and 0, otherwise. Finally, $O^{n_j,n_{j'}}_{p_i,p_{i'},t_k}$ equals to 1 if the players p_i and $p_{i'}$ are linked together, and 0 otherwise. We also define a utility function $U(p_i,t_k)$ for each player p_i that can evaluate its utility in accordance to the objective function. The player's utility is evaluated depending on the player's energy consumption of its physical mode, virtualized mode, and the communication links, that is equivalent to $E^P_{p_i,t_k}$, $E^V_{p_i,t_k}$, and $E^{comm}_{t_k}$, respectively. Therefore, the model can be expressed as a distributed optimization problem as follows:

$$E_{tot} = \min \sum_{p_i \in \mathcal{P}} \sum_{n_j \in \mathcal{N}} \sum_{t_k \in \mathcal{T}} \Big(X_{p_i,n_j,t_k}\, U_{p_i,t_k} + Z_{p_i,n_j,t_k}\, U_{p_i,t_k} \Big) + \sum_{(p_i,p_{i'}) \in \mathcal{E}} \sum_{(n_j,n_{j'}) \in \mathcal{L}} \sum_{t_k \in \mathcal{T}} O^{n_j,n_{j'}}_{p_i,p_{i'},t_k}\, U^{n_j,n_{j'}}_{p_i,p_{i'},t_k}, \tag{5.22}$$

Given that the utility function is expressed as total energy consumption of players and communication links, therefore, the above equation (5.22) can be re-written in terms of energy consumption of players in physical mode, virtualized mode, and the communication links as follows:

$$E_{tot} = \min \sum_{p_i=1}^{\mathcal{M}} \sum_{t_k=1}^{\mathcal{T}} \Bigg(X_{p_i,n_j,t_k} \, E^{P}_{p_i,t_k} + Z_{p_i,n_j,t_k} \, E^{V}_{p_i,t_k} + O^{n_j,n_{j'}}_{p_i,p_{i'},t_k} \, E^{comm}_{t_k} \Bigg),$$

$$\forall \, n_j \in \mathcal{N}, (p_i, p_{i'}) \in \mathcal{E}, (n_j, n_{j'}) \in \mathcal{L}.$$

$$s.t.$$

$$C1 : \sum_{p_i \in \mathcal{P}} X_{p_i,n_j,t_k} + \sum_{p_i \in \mathcal{P}} Z_{p_i,n_j,t_k} = 1, \quad \forall \, n_j \in \mathcal{N}, t_k \in \mathcal{T}.,$$

$$C2 : \sum_{n_j \in \mathcal{N}} X_{p_i,n_j,t_k} \leq 1 - Y_{p_i,t_k}, \quad \forall \, p_i \in \mathcal{P}, t_k \in \mathcal{T}.,$$

$$C3 : \sum_{p_i \in \mathcal{P}} \sum_{t_k \in [0, arr^{t_k}_{n_j})} \Bigg(X_{p_i,n_j,t_k} + Z_{p_i,n_j,t_k} \Bigg) = 0, \quad \forall \, n_j \in \mathcal{N}.,$$

$$C4 : \sum_{p_i \in \mathcal{P}} \sum_{t_k \in [arr^{t_k}_{n_j}, dlt^{t_k}_{n_j}]} \Bigg(X_{p_i,n_j,t_k} + Z_{p_i,n_j,t_k} \Bigg) = 1, \quad \forall \, n_j \in \mathcal{N}.,$$

$$C5 : \sum_{p_i \in \mathcal{P}} Z_{p_i,n_j,t_k} \leq C_{p_i} \, Y_{p_i,t_k}, \quad \forall \, n_j \in \mathcal{N}, t_k \in \mathcal{T}.,$$

$$C6 : X_{p_i,n_j,t_k} \, E^{P}_{p_i,t_k} \leq E^{p_i}_{t_k}, \quad \forall \, p_i \in \mathcal{P}, n_j \in \mathcal{N}, t_k \in \mathcal{T}.,$$

$$C7 : \sum_{n_j \in \mathcal{N}} Z_{p_i,n_j,t_k} \, E^{V}_{p_i,t_k} \leq E^{p_i}_{t_k}, \quad \forall \, p_i \in \mathcal{P}, t_k \in \mathcal{T}.,$$

$$C8 : \sum_{(n_j,n_{j'}) \in \mathcal{L}'_h} O^{n_j,n_{j'}}_{p_i,p_{i'},t_k} \, B_h(n_j, n_{j'}) \leq \min(B_g(p_i, p_{i'})), \quad \forall \, (p_i, p_{i'}) \in \mathcal{E}, t_k \in \mathcal{T}.,$$

$$C9 : \sum_{(n_j,n_{j'}) \in \mathcal{L}'_h} \sum_{(p_i,p_{i'}) \in \mathcal{E}} D_h(n_j, n_{j'}) \geq D_g(p_i, p_{i'}), \quad \forall \, t_k \in \mathcal{T}. \tag{5.23}$$

The constraints (C1) and (C2) ensure that the task is only executed by one player at a time and the player can only execute a task in physical or virtual mode. The constraints (C3) and (C4) ensure that a task is executed after its arrival and before its deadline. Also, the constraints (C5), (C6), and (C7) ensure that the players have sufficient capacity and energy for executing a task. Finally, constraints (C8) and (C9) guarantee the bandwidth and end-to-end latency requirements are satisfied. The problem described in (5.23) discusses the minimization of energy between all players, while meeting the defined SLAs and constraints. Accordingly, we notice that the problem discussed above is a linear model with a linear objective function, that leads to be a convex problem [122].

Theorem 2. *The derived utility function of the proposed game theory formulation is monotone. The monotonicity of the proposed utility function will demonstrate that it continuously decreases/increases (based on the utility function) over the variables, and by evolution of the formulation (additional variables/constraints) the overall network behaviour remains consistent [123].*

Proof. To show the monotonicity of the derived formulation, we have to show its derivative always produces a

non-negative result over the set of feasible solutions of player (p_i). Let us consider the linear objective function defined in (5.23). We therefore have:

$$f(\mathbf{E}_{tot}^{t_k}) = \sum_{t_k=1}^{t_k=\mathcal{T}} \left(X_{p_i,n_j,t_k} E_{p_i,t_k}^{P} + Z_{p_i,n_j,t_k} E_{p_i,t_k}^{V} + O_{p_i,p_{i'},t_k}^{n_j,n_{j'}} E_{t_k}^{comm} \right),$$

while $t_k = 1$,

$$f(\mathbf{E}_{tot}^{t_k}) = X_{p_i,n_j,t_k} E_{p_i,t_k}^{P} + Z_{p_i,n_j,t_k} E_{p_i,t_k}^{V} + O_{p_i,p_{i'},t_k}^{n_j,n_{j'}} E_{t_k}^{comm},$$

$$\frac{df(\mathbf{E}_{tot}^{t_k})}{d(\mathbf{E}_{tot}^{t_k})} = f'(\mathbf{E}_{tot}^{t_k}) = E_{p_i,t_k}^{P} + E_{p_i,t_k}^{V} + E_{t_k}^{comm} > 0,$$

$$\therefore f'(\mathbf{E}_{tot}^{t_k}) > 0 \tag{5.24}$$

It is evident in Eq. (5.24), that the derived function $f(\mathbf{E}_{tot}^{t_k})$ is always non-negative and it is monotone. Therefore, given that the derived non-cooperative game model is both convex and continuous, any solution of the defined utility function f over $(\mathbf{E}_{tot}^{t_k})$ of the player (p_i) is a Pure Nash Equilibrium.

5.3. E2M: Energy Efficient Matching

In this section, we propose our distributed dynamic **E**nergy **E**fficient **M**atching (E2M) heuristic to solve the problem in a computationally efficient manner. Our proposed E2M heuristic aims to minimize the overall energy consumption of executing the tasks over the physical and virtual players over time while meeting the given QoS requirements (i.e., task deadlines, end-to-end latency, and bandwidth requirements). The expected output of the proposed E2M heuristic is the set of action profile $\mathbf{a} = (\mathbf{a}_{p_i}, \mathbf{a}_{p_{-i}})$ performed by players $p_i \in \mathcal{P}$. The pseudo-code of our proposed E2M heuristic is illustrated in Algorithm 5.1.

Initially, the sets of players and their associated capacity and energy sets are defined, followed by the variables linked to the given requirements. As specified in Algorithm 5.1, our proposed heuristic iterates over the set $\mathcal{T}$ of time slots followed by iterations over the set of players and tasks (lines 8 to 10 of Algorithm 5.1). Then, the heuristic applies the *Dense()* function that uses the Euclidean distance algorithm to explore the application sensing tasks in sensing range (proximity) of each player. The heuristic also derives the deadlines of each task and sorts the collected list using the Timsort function [114] (lines 11 to 13). Next, the player determines its action set. The action set is all the possible subsets of the tasks in the sensing range of the player. Once each player realizes its action set, it compares it with the neighboring players, and the player with the largest action set initiates the execution of the tasks using its largest available subset of tasks in its action set. In this case, considering that there is more than one action available for the player, it will decide to be virtualized and execute the tasks over the virtual players. If there are multiple players with

Algorithm 5.1 Energy Efficient Matching (E2M) Heuristic

Input: $\mathcal{P}, \mathcal{E}, \mathcal{N}, \mathcal{L}, \mathcal{T}$
Output: $p_i \in \mathcal{P}, \mathbf{a} = (\mathbf{a}_{p_i}, \mathbf{a}_{p_{-i}})$ /* set of players and their action profile */
C_{p_i}= {cap(p_i) / $p_i \in \mathcal{P}$}; /* player's capacity */
$E_{t_k}^{p_i}$ = {e(p_i) / $p_i \in \mathcal{P}$}; /* player's available energy */
$\mathcal{K}, \mathcal{K}' = \mathcal{KSP}(p_i, p_{i'})$; /* k-shortest simple paths between $(p_i, p_{i'}) \in \mathcal{E}$ */
$\mathcal{D}, \mathcal{D}' = \mathcal{D}_g(p_i, p_{i'})$; /* link latency of $(p_i, p_{i'}) \in \mathcal{E}$ */
$\hat{\mathcal{D}} = \mathcal{D}_h(n_j, n_{j'})$; /* latency requirements of $(n_j, n_{j'}) \in \mathcal{L}$ */
$\mathcal{B}, \mathcal{B}' = \mathcal{B}_g(p_i, p_{i'})$; /* bandwidth of $(p_i, p_{i'}) \in \mathcal{E}$ */
$\hat{\mathcal{B}} = \mathcal{B}_h(n_j, n_{j'})$; /* bandwidth requirement of $(n_j, n_{j'}) \in \mathcal{L}$ */
for $t_k \in \mathcal{T}$ **do**
  for $p \in \mathcal{P}$ **do**
    for $n \in \mathcal{N}$ **do**
      $\hat{\mathcal{F}}$= {Dense(p_i, n_j)}; /* discover the set of tasks */
      $\mathcal{DT} = \{dlt_{n_j}(\hat{\mathcal{F}})\}$; /* set of deadlines of tasks in $\hat{\mathcal{F}}$ */
      $\hat{\mathcal{DT}} = \mathcal{DT}.sort()$;
      while $p_i < \hat{\mathcal{F}}$.size() **do**
        if ((|$\mathbf{a}_{p_i}$|> |$\mathbf{a}_{p_{i'}}$|> 1) **or** ((|$\mathbf{a}_{p_i}$|= |$\mathbf{a}_{p_{i'}}$|> 1) **and** (E($\hat{\mathcal{F}}_{p_i}$) > E($\hat{\mathcal{F}}_{p_{i'}}$)))) **then**
          select player p_i;
          check capacity and energy constraints using Eqs. (5.12) and (5.14)
          check task deadlines $n_j \in (a_{p_i}^{t_k}) \leq dlt_{n_j}^{t_k}$ using Eq. (5.16);
          $(a_{p_i}) = \max((|\mathbf{a}_{p_i}|) \mid |\mathbf{a}_{p_i}| \in \mathcal{A}_{p_i})$;
          update action profile of players p_i and $p_{i'}$ using Eq. (5.10);
          apply Eq. (5.3) and Eq. (5.15) on player p_i;
        else
          select player $p_{i'}$;
          apply Eq. (5.3) and Eq. (5.15) on player $p_{i'}$;
        end if
        if ((|$\mathbf{a}_{p_i}$|$\leq$ 1) **or** ($C_{p_i} == 1$)) **then**
          check capacity and energy constraints using Eqs. (5.12) and (5.13);
          check task deadlines $n_j \in (a_{p_i}^{t_k}) \leq dlt_{n_j}^{t_k}$ using Eq. (5.16);
          $(a_{p_i}) = (|\mathbf{a}_{p_i}|) \mid |\mathbf{a}_{p_i}| \in \mathcal{A}_{p_i})$;
          update action profile of players p_i and $p_{i'}$ using Eq. (5.10)
          apply Eq. (5.1) and Eq. (5.15) on player p_i;
        end if
        $p_i \leftarrow p_i + 1$;
      end while
      while $(n_j, n_{j'}) == 1$ **do**
        explore k-shortest paths between p_i and $p_{i'}$
        if $\mathcal{K}(p_i, p_{i'}) > 1$ **then**
          if (($\mathcal{D}(\mathcal{K}) < \mathcal{D}(\mathcal{K}') < \hat{\mathcal{D}}(\hat{\mathcal{K}})$)) (Eq. (5.18)) **and** (($\mathcal{B}(\mathcal{K}) > \mathcal{B}(\mathcal{K}') > \hat{\mathcal{B}}(\hat{\mathcal{K}})$)) (Eq. (5.17)) **then**
            calculate communication link energy using Eq. (5.5)
            select $\min(E(\mathcal{K}), E(\mathcal{K}'))$
          else
            if $(E(\mathcal{K}) == E(\mathcal{K}'))$ **then**
              select $\min(D(\mathcal{K}), D(\mathcal{K}'))$
            end if
          end if
        end if
      end while
    end for
  end for
  $T_{k+1}.add(n_{j'} \in \mathbf{a}_{p_i})$; /* Task $n_{j'}$ goes to T_{k+1} */
end for
return p, a;

the same number of subsets of actions, the player with the highest available energy will be selected to execute the tasks given the available energy, capacity, and task deadline requirements. Accordingly, the other players will update their action set and remove the subset tasks that were already selected for execution. This, as a result, avoids redundant execution of tasks. Next, the heuristic calculates the consumed energy based on the action taken by the player, followed by the updates applied to players' available energy (lines 14 to 21). If any of the constraints defined are not satisfied, the next available player will start executing its action set (lines 23 to 25). If there is a single available action in the action set of the player, it will execute the given task in its physical mode provided that the capacity, energy, and task deadline requirements are all met. Accordingly, the action set of other players will be updated to remove that particular action from their action set. Then, the player's energy will be updated and the consumed energy will be determined. This process will be repeated for all the players and their respective actions (lines 26 to 33). Once the players make their actions over the tasks, the logical communication links between the tasks and their respective allocations over the physical network need to be considered. We use the k-shortest path algorithm, exploring the k paths between the players. If a path does not comply with the end-to-end latency or bandwidth constraints, it will be eliminated. Meanwhile, if there are multiple paths that satisfy both latency and bandwidth requirements, the path with the lowest communication energy will be selected. In case the discovered paths hold equal communication energy consumption, the path with minimum delay will be selected (lines 35 to 43). At this point, given that the tasks in the action set of the players are sorted in ascending order of their deadlines, if there are any tasks in the action set that are left out, they will be moved to the next time-slot for re-consideration for execution as long as their deadline constraints are not violated (line 50). Consequently, the heuristic returns the list of players with their action sets (line 52).

5.4. Performance Evaluation

In this section, we explain our evaluation scenarios and then present the obtained results.

5.4.1 Evaluation Scenarios

We ran extensive simulations on large-scale networks. The system was examined under different values of virtualization overhead, transmission, and link energy consumption values. We considered 1000 sensors in an area of 1000 m $\times$ 1000 m with a total of 20000 tasks. We also tested our system with 500 sensors and various task loads ranging from 1000 to 16000. In all scenarios, we consider the following settings for the sensors: a sensing range of 30 m [52], random initial energy selected from the range $[1.9 - 3.4]$ J [34], and transmission energy and reception energy of 0.017 mJ and 0.031 mJ [34], respectively. The virtualization overhead is set to 0.005 mJ [11], while the energy consumption of CPU and wireless link is set to 0.005 mJ [79] and $[0.007 - 0.021]$ mJ [93], respectively. The required data volume and link bandwidth are set to $[100 - 200]$ bps and $[200 - 400]$ kbps, respectively [48]. The

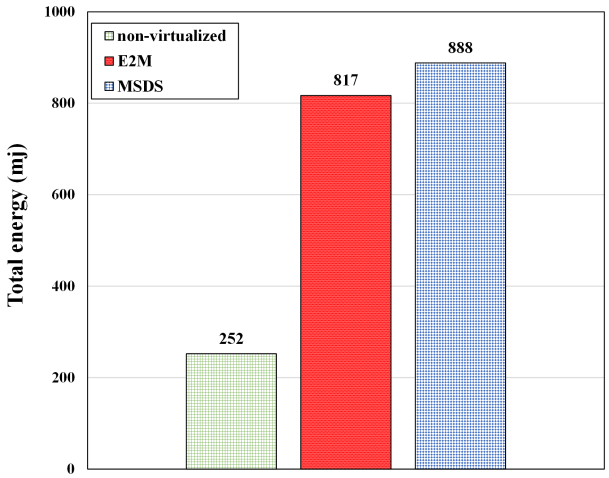


Figure 5.1: Total energy consumption.

transmission delay and virtual sensor creation overhead delay are set to $[0.02 - 0.06]$ s and 0.06 s, respectively [11].

5.4.2 Results

We present the results obtained from our proposed E2M heuristic and compared them with those of the MSDS algorithm [98] and a scenario without any virtualization. The tests were implemented using Python and run on a machine with 64-bit OS Windows 10 Pro, 3.20 GHz Intel ® Corei7-8700 ® CPU, and 16 GB of memory.

A) Energy Consumption

Figure 5.1 illustrates the energy efficiency of the proposed E2M heuristic, MSDS algorithm [98], and non-virtualized solution. We observe from the figure that the non-virtualized solution achieves the smallest energy consumption. This is due to the fact that in a non-virtualized solution, sensors can execute only one sensing task at a time. Thus, fewer number of tasks are executed, thus leading to smaller energy consumption. In addition, the results show that the proposed E2M heuristic outperforms the MSDS algorithm [98] by 8%. This happens because the MSDS algorithm [98] does not take into account the virtualization overhead energy consumption nor does it consider the impacts of routing in its assignment. Figure 5.2 depicts the average energy consumption per task vs. virtualization overhead energy. The results reveal that by increasing the virtualization overhead, the average energy consumption increases, which was expected. We also observe that our proposed E2M heuristic outperforms the MSDS algorithm, showing up to a 24% reduction of energy consumption per task. On the other hand, the energy consumption per task in the non-virtualized solution remains constant for different values of virtualization overhead.

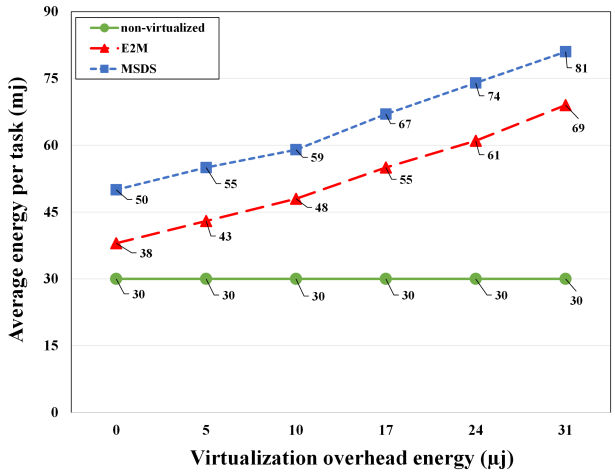


Figure 5.2: Average energy consumption per task vs. virtualization overhead.

B) Admission Rate

Figure 5.3 illustrates the overall admission rate for different assignment schemes under consideration. We observe from the figure that our proposed E2M heuristic outperforms the MSDS algorithm [98] and non-virtualized solution by 14% and 53%, respectively. The main reason is that in our proposed heuristic, the tasks are executed with respect to their deadline requirements. Therefore, if a task cannot be executed upon its arrival, it may be executed in the upcoming time-slots, if the deadline allows. To better see this, we plot the admission rate vs. assignment delay in Fig. 5.4, which reveals more details on the breakdown of the admission rate for different solutions under study. We observe from the figure that even though the MSDS [98] algorithm managed to execute 81% of the tasks at their arriving time-slot, it drops 19% of the tasks. By contrast, our proposed E2M heuristic drops only 5% of the tasks, mainly because it tries to execute as many requests as possible in the subsequent time-slots too. More importantly, the non-virtualized solution has successfully executed 18% but dropped 58% of the tasks.

C) Scalability

Next, we evaluate the scalability performance of our proposed algorithm for a large number of tasks ranging from 1000 to 16000 over 500 sensors. Figure 5.5 illustrates the total energy consumption vs. number of tasks. We observe that the non-virtualized solution achieves the smallest total energy consumption, mainly because it executes only a small portion of the incoming tasks. On the other hand, the proposed E2M heuristic outperforms the MSDS [98] algorithm by up to 34%, which happens when the number of tasks is large. In addition, we observe in Fig. 5.6 that all three solutions under study have a very high admission rate ($\sim$100%) for $<$ 2000 tasks. By increasing the number of tasks to 4000, the admission rate of the MSDS [98] algorithm and the non-virtualized solution starts to drop to 89% and 85%, respectively, while the admission rate of the proposed E2M heuristic remains at $\sim$99%. There is also a sharp

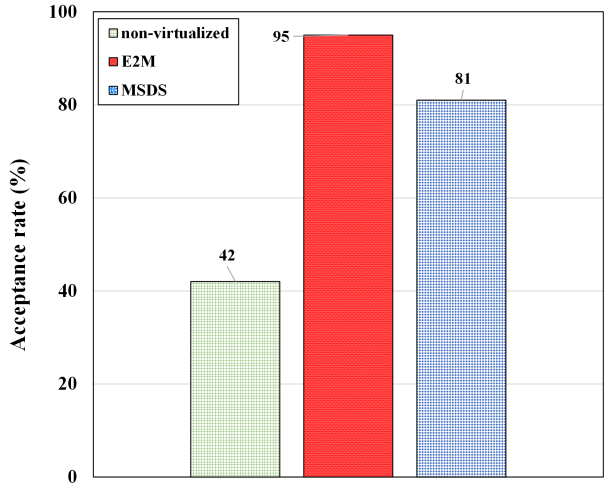


Figure 5.3: Admission rate.

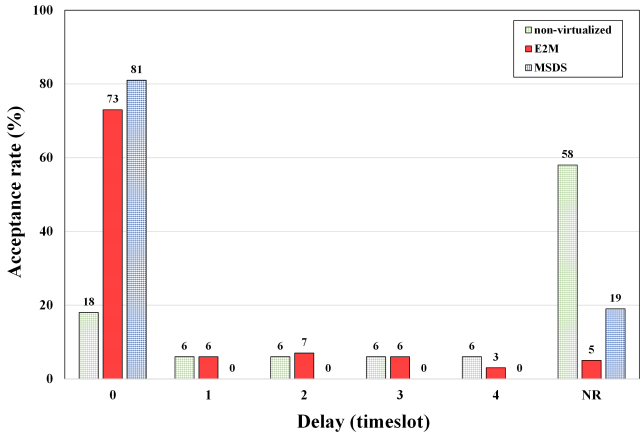


Figure 5.4: Admission rate vs. assignment delay.

decline in the admission rate when the number of tasks increases to 16000. According to Fig. 5.6, the admission rate of the non-virtualized solution and MSDS algorithm is 27% and 31%, respectively, compared to the proposed E2M heuristic with an admission rate of 52%.

D) Virtualizable to non-Virtualizable Ratio

Let β denote the ratio of the number of non-virtualizable sensors to the total number of sensors. Hence, $\beta = 20$ indicates that 20% of the sensors are non-virtualizable, and it is not possible to instantiate virtual sensors on top of them. Figure 5.7 depicts the total energy consumption vs. β. We observe that the energy consumption of both algorithms reduces monotonically as β increases. This is evident because as the number of virtualizable sensors grows larger, a larger number of tasks can be executed, thus leading to higher energy consumption. Besides, virtualization introduces additional virtualization overhead energy, which may have impacts on the total energy consumption. Figure 5.8 depicts

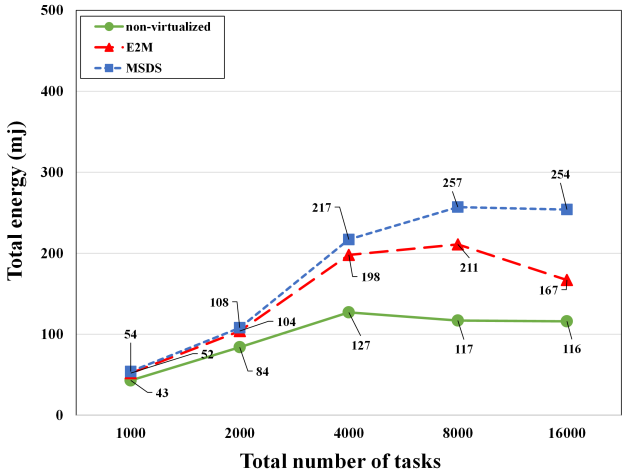


Figure 5.5: Total energy consumption vs. total number of tasks.

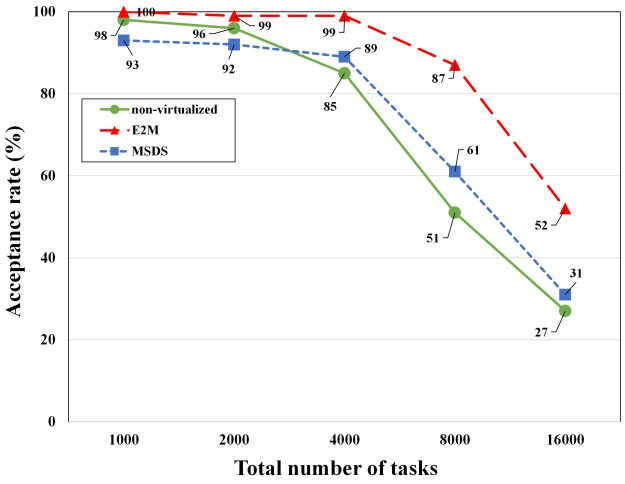


Figure 5.6: Admission rate vs. total number of tasks.

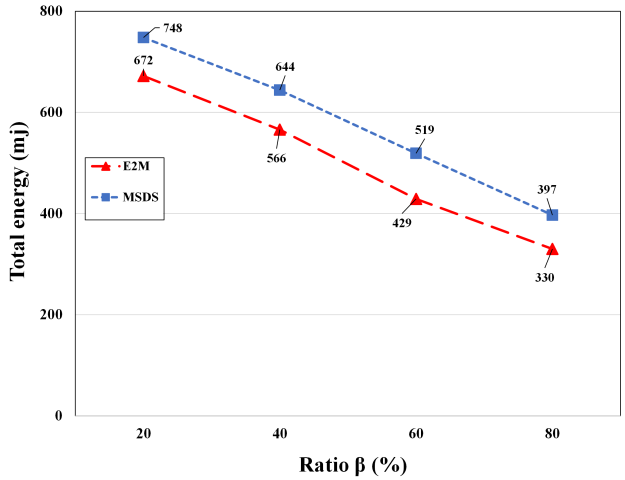


Figure 5.7: Total energy consumption vs. ratio β.

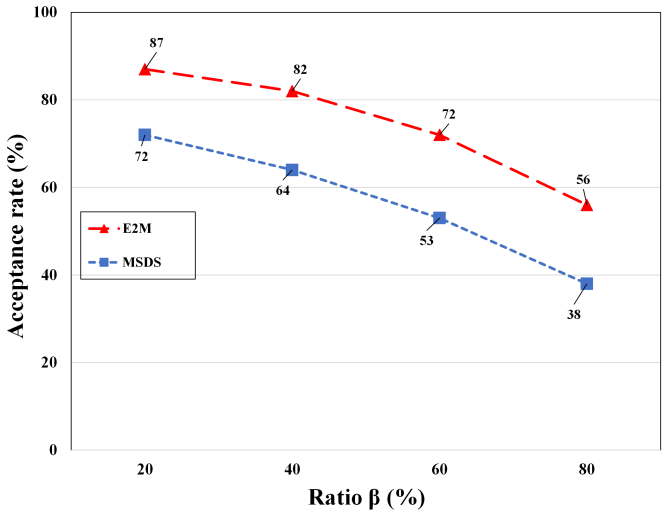


Figure 5.8: Admission rate vs. ratio β.

admission rate vs. ratio β. As shown in the figure, by increasing the ratio β, the admission rate drops in both the proposed E2M heuristic and MSDS algorithm. The main reason is that non-virtualizable sensors cannot execute more than one task at a time. Thus, when the number of non-virtualizable nodes increases, the number of dropped tasks increases as a result. The obtained results reveal that the proposed E2M heuristic outperforms the MSDS algorithm by up to 17% and 32% in terms of energy consumption and admission rate, respectively.

E) Execution Time

Finally, we investigated the execution time and the end-to-end latency performance of the proposed algorithm. We illustrate the execution time (in seconds) vs. the total number of tasks in Fig. 5.9. We observe that by increasing the total number of tasks, the execution time increases for all the solutions under study. When there are 1000 to 4000 tasks, the execution time of the proposed E2M heuristic, MSDS [98] algorithm, and the non-virtualized solution is almost the same with 1 to 3 seconds difference. However, by increasing the number of tasks from 4000 to 16000, the execution time of all three solutions increases. More specifically, the non-virtualized solution has the highest execution time. This is because in a non-virtualized network, a very limited amount of resources is available and tasks need to be selected from a pool of tasks. Therefore, the solution takes a considerable amount of time to select the set of tasks to be executed. On the other hand, the MSDS [98] algorithm has the lowest execution time. It performs 1-2 seconds better than our proposed heuristic when there are 1000-4000 tasks and it performs $\sim$10 seconds faster than our proposed heuristic when there are 16000 tasks. This happens because of two reasons. First, our algorithm seeks to select the path with the lowest energy consumption. Secondly, it prioritizes the tasks and shifts the tasks forward to the next time-slot for execution, if possible, in order to improve the admission rate and energy consumption, as discussed above.

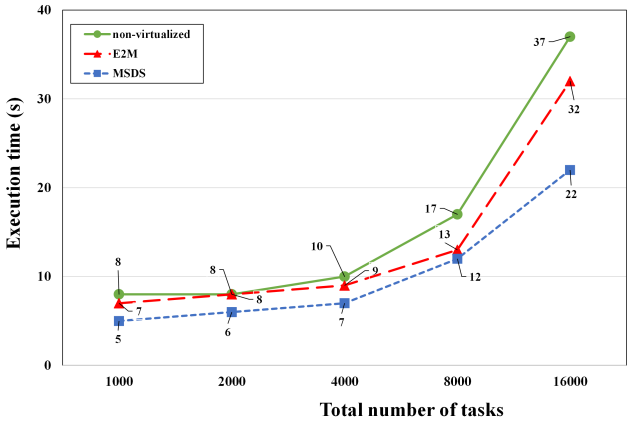


Figure 5.9: Execution time vs. total number of tasks.

5.5. Conclusions

We studied the problem of task assignment for distributed vIoT. We modeled the problem as a non-cooperative game, aiming at minimizing energy consumption while meeting end-to-end latency, bandwidth, and task deadline QoS metrics. We proposed a heuristic to solve the problem and evaluated it in terms of energy consumption, admission rate, scalability, and execution time. The results indicate that the proposed heuristic can achieve smaller energy consumption while increasing the admission rate in different large-scale tests, as compared to the benchmarks. The simulation results indicate that the proposed heuristic can achieve an improvement in the energy consumption and admission rate of up to 24% and 14%, respectively, compared to an existing benchmark.

Chapter 6

Conclusions and Future Works

6.1. Conclusions

The Internet of Things and Wireless Sensor Networks are widely being used in various application domains ranging from smart home, smart city, smart manufacture, to e-healthcare, smart disaster management, etc. Accordingly, based on Cisco's anticipation, it is expected to have over 500 billion connected devices by 2030 [9]. Considering that the WSN nodes are mainly constrained devices with limited available resources in terms of energy, computing, and storage, hence, inefficient resource utilization of these constrained devices results in faster depletion of their energy, making them inaccessible. Virtualization technology is applied to address this issue. However, virtualization challenges energy consumption due to the additional energy consumed by virtualization overhead. In this Ph.D. thesis, we addressed the algorithmic challenges related to the task assignment in virtualized WSNs in chapter 3. We aimed at minimizing the overall energy consumption of the WSN while considering task deadlines as the QoS requirement. It was observed that the proposed solution enhances energy consumption and it has a lower execution time as compared to the optimal solution and the literature.

Several applications such as fire contour require the WSN nodes to interact and collaborate to address an application's request. The second challenge is to address dynamic network embedding in virtualized wireless sensor networks in an energy-efficient manner. Therefore, it is critical to have a mechanism that enables multiple virtual networks to co-exist over the same deployed virtualized WSN substrate network. We addressed this challenge in chapter 4. The objective of the proposed solution was to minimize overall energy consumption. In addition, it considered the E2E latency and bandwidth as SLAs. The effectiveness of the proposed solution was tested over both small- and large-scale scenarios in different tests and the results were compared and analyzed with those obtained from the optimal solution and a related work.

Considering that the WSNs are intrinsically distributed, it is not always feasible to have a resource-rich node

outside of the WSN. Therefore, it is crucial to design a distributed mechanism that can be executed over the resource-constrained nodes inside of the WsN with interacting modules dedicated to each sensor node. We addressed this challenge in chapter 5 by modeling the problem of distributed task assignment in virtualized WSNs and proposing a solution to solve the problem. We aimed at minimizing the overall energy consumption while satisfying the given QoS criteria of E2E latency, bandwidth, and task deadline. The solution was implemented and its results were compared with a work from the literature and a scenario without virtualization. The results reveal that the proposed solution can enhance energy efficiency and admission rate in different tests.

6.2. Future Works

This thesis presented significant contributions towards energy efficiency in virtualized IoT networks by addressing the related challenges in task assignment, network embedding, and distributed task assignment. Yet, there exist several research directions for the future.

6.2.1 Task Assignment in Virtualized WSNs

There are several possible approaches to further continue and improve this work. ***Mobility*** is an interesting aspect to consider. In this work, it is assumed that all of the sensor nodes are stationary. In other words, t is assumed that they are static with no mobility and movement. It would be interesting to investigate the problem considering a mobile network. In addition, ***Mobile Crowdsensing*** is another research aspect that can be considered. By increasing the number of IoT nodes, it is still a challenge to design, model, and implement algorithms to assign sensing tasks to IoT devices in an energy-efficient manner.

6.2.2 Network Embedding in Virtualized WSNs

One interesting research direction to follow for the solution proposed is to take advantage of applications of Edge Computing. While embedding the virtual networks onto the virtualized IoT substrate network, modeling an algorithm that can assist in the decision-making process on which nodes (on the virtualized substrate network or at the edge of the network) to be used will be helpful. This process is heavily dependent on the stringency of the QoS requirements of the applications that request the resources. Furthermore, Resource Preservation is another interesting research direction. Considering that a lot of applications with diverse requirements need to be deployed on the IoT nodes, it will be useful to have a predictive mechanism that can estimate the volume of incoming applications' requests. This will help to solve the problem of resource allocation and resource preservation in virtualized WSNs based on those estimated values obtained by applying machine learning techniques.

6.2.3 Distributed Task Assignment in Virtualized IoT

There are number of ways to contribute to expand and continue the research in distributed task assignments in virtualized IoT. As a future work, we aim to determine the Price of Anarchy (PoA) and Price of Stability (PoS). To find the PNE experimentally, the SAGE Tools [124] can be used while the Gambit Tools [125] can be applied to explore the PoA and the PoS. By analyzing the existence of PNE, it will be possible to determine if the proposed game model has a stable solution. The existence of PNE indicates that all players prefer to follow the pure strategy in which no player achieves a higher payoff by deviating from its strategy given all strategies of other players. On the other hand, the PoA is defined as "the ratio between the worst equilibrium and the optimal solution whereas the PoS is the ratio between the best equilibrium and the optimal solution" [121]. The optimal solution can be obtained by re-formulating the problem as ILP/MILP and solved using optimization engines to get the results. These two ratios can quantify how well a PNE is compared to the optimal solution. In addition to the above, another interesting future work is to design and implement a machine learning techniques such as reinforcement learning (RL) and deep RL that eventually learns the game to solve the problem.

Bibliography

[1] **Raee, Vahid Maleki**, D. Naboulsi, and R. Glitho, "Energy efficient task assignment in virtualized wireless sensor networks," in *Proc. IEEE Symposium on Computers and Communications **(ISCC)***, pp. 00976–00979, 2018.

[2] **Raee, Vahid Maleki**, A. Ebrahimzadeh, R. H. Glitho, and H. Elbiaze, "Ensuring energy efficiency when dynamically assigning tasks in virtualized wireless sensor networks," *IEEE Transactions on Green Communications and Networking **(TGCN)***, vol. 6, no. 1, pp. 613–628, 2021.

[3] **Raee, Vahid Maleki**, A. Ebrahimzadeh, M. Rayani, R. Glitho, M. El Barachi, and F. Belqasmi, "Energy efficient virtual network embedding in virtualized wireless sensor networks," in *Proc. IEEE Consumer Communications & Networking Conference **(CCNC)***, pp. 187–192, 2022.

[4] **Raee, Vahid Maleki**, A. Ebrahimzadeh, R. H. Glitho, M. El Barachi, and F. Belqasmi, "E2dne: Energy efficient dynamic network embedding in virtualized wireless sensor networks," *submitted to IEEE Transactions on Green Communications and Networking **(TGCN)***, 2022 (Under Review).

[5] **Raee, Vahid Maleki**, A. Ebrahimzadeh, Z. Mlika, and R. Glitho, "Energy efficient distributed task assignment for virtualized Internet of Things," *submitted to IEEE Transactions on Network and Service Management **(TNSM)***, 2023 (Under Review).

[6] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless sensor network virtualization: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 553–576, 2015.

[7] M. Series, "Minimum requirements related to technical performance for IMT-2020 radio interface (s)," *Report*, pp. 2410–0, 2017.

[8] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Elsevier Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[9] "Dsi achieves cisco Internet of Things authorization." https://dsitech.com/about/news/2020/dsi-achieves-cisco-internet-of-things-iot-authorization.html, year=2020, note = Accessed: March 31, 2023.

[10] H. Yetgin, K. T. K. Cheung, M. El-Hajjar, and L. H. Hanzo, "A survey of network lifetime maximization techniques in wireless sensor networks," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 828–854, 2017.

[11] M. N. Alam and R. H. Glitho, "An infrastructure as a service for the Internet of Things," in *Proc. IEEE International Conference on Cloud Networking (CloudNet)*, pp. 1–7, 2018.

[12] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless sensor network virtualization: early architecture and research perspectives," *IEEE Network*, vol. 29, no. 3, pp. 104–112, 2015.

[13] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[14] R. Gomes, D. Vieira, Y. Ghamri-Doudane, and M. F. de Castro, "Network slicing for massive machine type communication in IoT-5G scenario," in *Proc. IEEE Vehicular Technology Conference (VTC2021-Spring)*, pp. 1–7, 2021.

[15] "Virtualization." https://www.ibm.com/cloud/learn/virtualization-a-complete-guide, 2019. Accessed: March 31, 2023.

[16] "Advanticsys." https://www.advanticsys.com/shop/industrial-iot-gateways-c-8/. Accessed: March 31, 2023.

[17] "Virtenio." https://www.virtenio.com/en/. Accessed: March 31, 2023.

[18] "Contiki os." http://www.contiki-os.org/. Accessed: March 31, 2023.

[19] A. P. Jayasumana, Q. Han, and T. H. Illangasekare, "Virtual sensor networks-a resource efficient approach for concurrent applications," in *Proc. IEEE International Conference on Information Technology (ITNG)*, pp. 111–115, 2007.

[20] S. Bose and N. Mukherjee, "Sensiaas: A sensor-cloud infrastructure with sensor virtualization," in *Proc. IEEE Conference on Cyber Security and Cloud Computing (CSCloud)*, pp. 232–239, 2016.

[21] S. Bose, N. Mukherjee, and S. Mistry, "Environment monitoring in smart cities using virtual sensors," in *Proc. IEEE Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 399–404, 2016.

[22] D. Rajavel, A. Chakraborty, and S. Misra, "QoS-aware sensor virtualization for provisioning green sensors-as-a-service," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 3, pp. 1128–1137, 2021.

[23] S. Misra, R. Schober, and A. Chakraborty, "Race: Qoi-aware strategic resource allocation for provisioning se-aas," *IEEE Transactions on Services Computing*, vol. 15, no. 3, pp. 1540–1550, 2020.

[24] K. S. Dar, A. Taherkordi, and F. Eliassen, "Enhancing dependability of cloud-based iot services through virtualization," in *Proc. IEEE Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 106–116, 2016.

[25] I. Khan, F. Belqasmi, R. Glitho, and N. Crespi, "A multi-layer architecture for wireless sensor network virtualization," in *Proc. IFIP/IEEE Wireless and Mobile Networking Conference (WMNC)*, pp. 1–4, 2013.

[26] I. Khan, F. Z. Errounda, S. Yangui, R. Glitho, and N. Crespi, "Getting virtualized wireless sensor networks' IaaS ready for PaaS," in *Proc. IEEE Conference on Distributed Computing in Sensor Systems*, pp. 224–229, 2015.

[27] "Waspmote." https://www.libelium.com/iot-products/waspmote/, note = Accessed: March 31, 2023.

[28] A. D. Kshemkalyani and M. Singhal, *Distributed computing: principles, algorithms, and systems*. Cambridge University Press, 2011.

[29] "Introduction to parallel computing tutorial." https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial, note = Accessed: March 31, 2023.

[30] "Concurrency." https://web.mit.edu/6.005/www/fa14/classes/17-concurrency/, note = Accessed: March 31, 2023.

[31] "Deep ocean tsunami detection buoys." http://www.bom.gov.au/tsunami/about/detection_buoys.shtml. Accessed: March 31, 2023.

[32] J.-P. Montagner, K. Juhel, M. Barsuglia, J. P. Ampuero, E. Chassande-Mottin, J. Harms, B. Whiting, P. Bernard, E. Clévédé, and P. Lognonné, "Prompt gravity signal induced by the 2011 Tohoku-Oki earthquake," *Nature Communications*, vol. 7, no. 1, pp. 1–7, 2016.

[33] D. Zeng, P. Li, S. Guo, T. Miyazaki, J. Hu, and Y. Xiang, "Energy minimization in multi-task software-defined sensor networks," *IEEE Transactions on Computers*, vol. 64, no. 11, pp. 3128–3139, 2015.

[34] N. Edalat, W. Xiao, C.-K. Tham, E. Keikha, and L.-L. Ong, "A price-based adaptive task allocation for wireless sensor network," in *Proc. IEEE International Conference on Mobile Adhoc and Sensor Systems*, pp. 888–893, 2009.

[35] N. Edalat, W. Xiao, N. Roy, S. K. Das, and M. Motani, "Combinatorial auction-based task allocation in multi-application wireless sensor networks," in *Proc. IFIP/IEEE International Conference on Embedded and Ubiquitous Computing*, pp. 174–181, 2011.

[36] T. Le, T. J. Norman, and W. Vasconcelos, "Agent-based sensor-mission assignment for tasks sharing assets," in *Proc. Third International Workshop on Agent Technology for Sensor Networks*, 2009.

[37] S. Misra and A. Chakraborty, "QoS-aware dispersed dynamic mapping of virtual sensors in sensor-cloud," *IEEE Transactions on Services Computing*, vol. 14, no. 6, pp. 1970–1980, 2019.

[38] M. Lemos, R. Rabelo, D. Mendes, C. Carvalho, and R. Holanda, "An approach for provisioning virtual sensors in sensor clouds," *Wiley International Journal of Network Management*, vol. 29, no. 2, p. e2062, 2019.

[39] T. Ojha, S. Misra, N. S. Raghuwanshi, and H. Poddar, "DVSP: Dynamic virtual sensor provisioning in sensor–cloud-based Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5265–5272, 2019.

[40] M.-Z. Zhang, L.-M. Wang, and S.-M. Xiong, "Using machine learning methods to provision virtual sensors in sensor-cloud," *Sensors*, vol. 20, no. 7, p. 1836, 2020.

[41] C. M. de Farias, L. Pirmez, F. C. Delicato, W. Li, A. Y. Zomaya, and J. N. de Souza, "A scheduling algorithm for shared sensor and actuator networks," in *Proc. IEEE International Conference on Information Networking (ICOIN)*, pp. 648–653, 2013.

[42] C. Delgado, M. Canales, J. Ortín, J. R. Gállego, A. Redondi, S. Bousnina, and M. Cesana, "Joint application admission control and network slicing in virtual sensor networks," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 28–43, 2017.

[43] A. Mukherjee, P. Goswami, Z. Yan, L. Yang, and J. J. Rodrigues, "ADAI and adaptive PSO-based resource allocation for wireless sensor networks," *IEEE Access*, vol. 7, pp. 131163–131171, 2019.

[44] B. Zhao and X. Zhao, "Deep reinforcement learning resource allocation in wireless sensor networks with energy harvesting and relay," *IEEE Internet of Things Journal*, vol. 9, no. 3, pp. 2330–2345, 2021.

[45] T. L. Porta, C. Petrioli, C. Phillips, and D. Spenza, "Sensor mission assignment in rechargeable wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 10, no. 4, pp. 1–39, 2014.

[46] C. Delgado, M. Canales, J. Ortín, J. R. Gállego, A. Redondi, S. Bousnina, and M. Cesana, "Energy-aware dynamic resource allocation in virtual sensor networks," in *Proc. IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pp. 264–267, 2017.

[47] I. L. Santos, L. Pirmez, F. C. Delicato, G. M. Oliveira, C. M. Farias, S. U. Khan, and A. Y. Zomaya, "Zeus: A resource allocation algorithm for the cloud of sensors," *Elsevier Future Generation Computer Systems*, vol. 92, pp. 564–581, 2019.

[48] Z. Li and A. Zhong, "Resource allocation in wireless powered virtualized sensor networks," *IEEE Access*, vol. 8, pp. 40327–40336, 2020.

[49] A. Hazra, M. Adhikari, T. Amgoth, and S. N. Srirama, "Joint computation offloading and scheduling optimization of IoT applications in fog networks," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 3266–3278, 2020.

[50] Y. Yang and T. Song, "Energy-efficient cooperative caching for information-centric wireless sensor networking," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 846–857, 2021.

[51] D. Pizzocaro, M. P. Johnson, H. Rowaihy, S. Chalmers, A. Preece, A. Bar-Noy, and T. La Porta, "A knapsack approach to sensor-mission assignment with uncertain demands," in *Unmanned/Unattended Sensors and Sensor Networks V*, vol. 7112, p. 711205, International Society for Optics and Photonics, 2008.

[52] H. Rowaihy, M. P. Johnson, O. Liu, A. Bar-Noy, T. Brown, and T. L. Porta, "Sensor-mission assignment in wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 6, no. 4, pp. 1–33, 2010.

[53] C. Delgado, J. R. Gállego, M. Canales, J. Ortín, S. Bousnina, and M. Cesana, "On optimal resource allocation in virtual sensor networks," *Elsevier Ad Hoc Networks*, vol. 50, pp. 23–40, 2016.

[54] L. Chen, D. Wu, and Z. Li, "Multi-task mapping and resource allocation mechanism in software defined sensor networks," in *Proc. IEEE International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 32–37, 2020.

[55] A. Zhong, Z. Li, D. Wu, R. Wang, A. Fedotov, and V. Badenko, "Reward maximization strategy in virtualized wireless sensor networks," in *Proc. IEEE International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 38–43, 2020.

[56] S. Bharti and K. K. Pattanaik, "Task requirement aware pre-processing and scheduling for IoT sensory environments," *Ad Hoc Networks*, vol. 50, pp. 102–114, 2016.

[57] C.-C. Lin, D.-J. Deng, and L.-Y. Lu, "Many-objective sensor selection in IoT systems," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 40–47, 2017.

[58] A. Prasanth, J. A. George, and P. Surendram, "Optimal resource and task scheduling for IoT," in *Proc. IEEE International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pp. 1–4, 2019.

[59] E. A. Khalil, S. Ozdemir, and S. Tosun, "Evolutionary task allocation in Internet of Things-based application domains," *Elsevier Future Generation Computer Systems*, vol. 86, pp. 121–133, 2018.

[60] S. Bousnina, M. Cesana, J. Ortín, C. Delgado, J. R. Gállego, and M. Canales, "A greedy approach for resource allocation in virtual sensor networks," in *Proc. IEEE Wireless Days*, pp. 15–20, 2017.

[61] H. Rowaihy, M. Johnson, A. Bar-Noy, T. Brown, and T. La Porta, "Assigning sensors to competing missions," in *Proc. IEEE Global Telecommunications (GLOBECOM) Conference*, pp. 1–6, 2008.

[62] M. P. Johnson, H. Rowaihy, D. Pizzocaro, A. Bar-Noy, S. Chalmers, T. La Porta, and A. Preece, "Sensor-mission assignment in constrained environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 11, pp. 1692–1705, 2010.

[63] J. F. Botero, X. Hesselbach, M. Duelli, D. Schlosser, A. Fischer, and H. De Meer, "Energy efficient virtual network embedding," *IEEE Communications Letters*, vol. 16, no. 5, pp. 756–759, 2012.

[64] X. Chen, C. Li, and Y. Jiang, "Optimization model and algorithm for energy efficient virtual node embedding," *IEEE Communications Letters*, vol. 19, no. 8, pp. 1327–1330, 2015.

[65] M. Zhu, Q. Sun, S. Zhang, P. Gao, B. Chen, and J. Gu, "Energy-aware virtual optical network embedding in sliceable-transponder-enabled elastic optical networks," *IEEE Access*, vol. 7, pp. 41897–41912, 2019.

[66] Z. Xu, L. Zhuang, S. Tian, M. He, S. Yang, Y. Song, and L. Ma, "Energy-driven virtual network embedding algorithm based on enhanced bacterial foraging optimization," *IEEE Access*, vol. 8, pp. 76069–76081, 2020.

[67] V. Lira, E. Tavares, and M. Oliveira, "An approach for reducing energy consumption in dependable virtual network embedding," in *Proc. IEEE International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9, 2017.

[68] P. Zhang, H. Yao, and Y. Liu, "Virtual network embedding based on computing, network, and storage resource constraints," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3298–3304, 2017.

[69] C. Aguilar-Fuster, M. Zangiabady, J. Zapata-Lara, and J. Rubio-Loyola, "Online virtual network embedding based on virtual links' rate requirements," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1630–1644, 2018.

[70] P. Zhang, X. Pang, G. Kibalya, N. Kumar, S. He, and B. Zhao, "GCMD: Genetic correlation multi-domain virtual network embedding algorithm," *IEEE Access*, vol. 9, pp. 67167–67175, 2021.

[71] C. K. Dehury and P. K. Sahoo, "DYVINE: Fitness-based dynamic virtual network embedding in cloud computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1029–1045, 2019.

[72] S. Zhang, "Reliable virtual network mapping algorithm with network characteristics and associations," *IEEE Access*, vol. 9, pp. 48121–48130, 2021.

[73] I. Ullah, H.-K. Lim, and Y.-H. Han, "Ego network-based virtual network embedding scheme for revenue maximization," in *Proc. IEEE International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pp. 155–160, 2021.

[74] F. Habibi, M. Dolati, A. Khonsari, and M. Ghaderi, "Accelerating virtual network embedding with graph neural networks," in *Proc. IEEE International Conference on Network and Service Management (CNSM)*, pp. 1–9, 2020.

[75] L. Nonde, T. E. Elgorashi, and J. M. Elmirghani, "Cloud virtual network embedding: Profit, power and acceptance," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2015.

[76] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1040–1057, 2020.

[77] P. Zhang, C. Wang, N. Kumar, W. Zhang, and L. Liu, "Dynamic virtual network embedding algorithm based on graph convolution neural network and reinforcement learning," *IEEE Internet of Things Journal*, 2021.

[78] A. A. Nasiri, F. Derakhshan, and S. S. Heydari, "Distributed virtual network embedding for software-defined networks using multiagent systems," *IEEE Access*, vol. 9, pp. 12027–12043, 2021.

[79] H. Q. Al-Shammari, A. Lawey, T. El-Gorashi, and J. M. Elmirghani, "Energy efficient service embedding in IoT networks," in *Proc. IEEE Wireless and Optical Communication Conference (WOCC)*, pp. 1–5, 2018.

[80] H. Q. Al-Shammari, A. Lawey, T. El-Gorashi, and J. M. Elmirghani, "Energy efficient service embedding in IoT over PON," in *Proc. IEEE International Conference on Transparent Optical Networks (ICTON)*, pp. 1–5, 2019.

[81] H. Q. Al-Shammari, A. Q. Lawey, T. E. El-Gorashi, and J. M. Elmirghani, "Resilient service embedding in IoT networks," *IEEE Access*, vol. 8, pp. 123571–123584, 2020.

[82] P. Zhang, X. Pang, Y. Bi, H. Yao, H. Pan, and N. Kumar, "DSCD: Delay sensitive cross-domain virtual network embedding algorithm," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2913–2925, 2020.

[83] H. Afifi and H. Karl, "Reinforcement learning for virtual network embedding in wireless sensor networks," in *Proc. IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 123–128, 2020.

[84] H. Afifi, K. Horbach, and H. Karl, "A genetic algorithm framework for solving wireless virtual network embedding," in *Proc. IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 1–6, 2019.

[85] H. Afifi and H. Karl, "An approximate power control algorithm for a multi-cast wireless virtual network embedding," in *Proc. IEEE/IFIP Wireless and Mobile Networking Conference (WMNC)*, pp. 95–102, 2019.

[86] O. Kaiwartya, A. H. Abdullah, Y. Cao, J. Lloret, S. Kumar, R. R. Shah, M. Prasad, and S. Prakash, "Virtualization in wireless sensor networks: Fault tolerant embedding for Internet of Things," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 571–580, 2017.

[87] V. Cionca, R. Marfievici, R. Katona, and D. Pesch, "JudiShare: Judicious resource allocation for QoS-based services in shared wireless sensor networks," in *Proc. IEEE wireless communications and networking conference (WCNC)*, pp. 1–6, 2018.

[88] R. Katona, V. Cionca, D. O'Shea, and D. Pesch, "Virtual network embedding for wireless sensor networks time-efficient QoS/QoI-aware approach," *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 916–926, 2020.

[89] Y. Li, Z. Zhang, S. Xia, and H.-H. Chen, "A load-balanced re-embedding scheme for wireless network virtualization," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 4, pp. 3761–3772, 2021.

[90] D. T. Nguyen, C. Pham, K. K. Nguyen, and M. Cheriet, "Virtual network function placement in IoT network," in *Proc. IEEE International Wireless Communications & Mobile Computing Conference (IWCMC)*, pp. 1166–1171, 2019.

[91] R. Kouah, A. Alleg, A. Laraba, and T. Ahmed, "Energy-aware placement for IoT-service function chain," in *Proc. IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pp. 1–7, 2018.

[92] A. Leivadeas, M. Falkner, I. Lambadaris, M. Ibnkahla, and G. Kesidis, "Balancing delay and cost in virtual network function placement and chaining," in *Proc. IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pp. 433–440, 2018.

[93] H. Q. Al-Shammari, A. Q. Lawey, T. E. El-Gorashi, and J. M. Elmirghani, "Service embedding in IoT networks," *IEEE Access*, vol. 8, pp. 2948–2962, 2019.

[94] H.-S. Lee and J.-W. Lee, "Resource and task scheduling for SWIPT IoT systems with renewable energy sources," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2729–2748, 2018.

[95] W. Yu, Y. Huang, and A. Garcia-Ortiz, "Optimal task allocation algorithms for energy constrained multihop wireless networks," *IEEE Sensors Journal*, vol. 19, no. 17, pp. 7744–7754, 2019.

[96] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, pp. 2287–2295, 2019.

[97] K. Lin, Y. Li, Q. Zhang, and G. Fortino, "AI-driven collaborative resource allocation for task execution in 6G-enabled massive IoT," *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5264–5273, 2021.

[98] C. Avasalcai, C. Tsigkanos, and S. Dustdar, "Resource management for latency-sensitive IoT applications with satisfiability," *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2982–2993, 2021.

[99] W. Yu, Y. Huang, and A. Garcia-Ortiz, "Distributed optimal on-line task allocation algorithm for wireless sensor networks," *IEEE Sensors Journal*, vol. 18, no. 1, pp. 446–458, 2017.

[100] J. C. SanMiguel and A. Cavallaro, "Cost-aware coalitions for collaborative tracking in resource-constrained camera networks," *IEEE Sensors Journal*, vol. 15, no. 5, pp. 2657–2668, 2014.

[101] S. Wu, X. Peng, and G. Tian, "Decentralized max-min resource allocation for monotonic utility functions," in *PRoc. IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1–6, 2021.

[102] X. Fu, F. R. Yu, J. Wang, Q. Qi, and J. Liao, "Resource allocation for blockchain-enabled distributed network function virtualization (NFV) with mobile edge cloud (MEC)," in *Proc. IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1–6, 2019.

[103] X. Wang, R. Jia, X. Tian, and X. Gan, "Dynamic task assignment in crowdsensing with location awareness and location diversity," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, pp. 2420–2428, 2018.

[104] X. Wang, X. Chen, and W. Wu, "Towards truthful auction mechanisms for task assignment in mobile device clouds," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, pp. 1–9, 2017.

[105] L. Pournajaf, D. A. Garcia-Ulloa, L. Xiong, and V. Sunderam, "Participant privacy in mobile crowd sensing task management: A survey of methods and challenges," *ACM Sigmod Record*, vol. 44, no. 4, pp. 23–34, 2016.

[106] L. Moccia, J.-F. Cordeau, M. F. Monaco, and M. Sammarra, "A column generation heuristic for a dynamic generalized assignment problem," *Elsevier Computers & Operations Research*, vol. 36, no. 9, pp. 2670–2681, 2009.

[107] "Dual-pivot quicksort." https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html#sort(int[]). Accessed: March 31, 2023.

[108] V. Yaroslavskiy, "Dual-pivot quicksort," *Research Disclosure*, 2009.

[109] H. Kim, W.-K. Hong, J. Yoo, and S.-e. Yoo, "Experimental research testbeds for large-scale WSNs: A survey from the architectural perspective," *International Journal of Distributed Sensor Networks*, vol. 11, no. 3, pp. 1–18, 2015.

[110] "IBM ILOG CPLEX optimization studio." https://www.ibm.com/ca-en/products/ilog-cplex-optimization-studio. Accessed: March 31, 2023.

[111] E. Mortazavi, R. Javidan, M. J. Dehghani, and V. Kavoosi, "A robust method for underwater wireless sensor joint localization and synchronization," *Ocean Engineering*, vol. 137, pp. 276–286, 2017.

[112] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem," *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 213–220, 2016.

[113] R. Mosca, "Polynomial algorithms for the maximum stable set problem on particular classes of P5-free graphs," *Information processing letters*, vol. 61, no. 3, pp. 137–143, 1997.

[114] N. Auger, C. Nicaud, and C. Pivoteau, "Merge strategies: from merge sort to timsort," *[Online] (Accessed: March 31, 2023) https:https://hal-upec-upem.archives-ouvertes.fr/hal-01212839v2*, 2015.

[115] D. Yan, X. Yang, and L. Cuthbert, "Regression-based k nearest neighbours for resource allocation in network slicing," in *Proc. IEEE Wireless Telecommunications Symposium (WTS)*, pp. 1–6, 2022.

[116] Y. Han, D. Niyato, C. Leung, C. Miao, and D. I. Kim, "A dynamic resource allocation framework for synchronizing metaverse with iot service and data," in *Proc. IEEE International Conference on Communications (ICC)*, pp. 1196–1201, 2022.

[117] S. Rathinam, R. Sengupta, and S. Darbha, "A resource allocation algorithm for multivehicle systems with non-holonomic constraints," *IEEE Transactions on Automation Science and Engineering*, vol. 4, no. 1, pp. 98–104, 2007.

[118] J. Hershberger, S. Suri, and A. Bhosle, "On the difficulty of some shortest path problems," *ACM Transactions on Algorithms (TALG)*, vol. 3, no. 1, pp. 1–15, 2007.

[119] A. Moschitta and I. Neri, "Power consumption assessment in wireless sensor networks," in *ICT-energy-concepts towards zero-power information and communication technology*, IntechOpen, 2014.

[120] C. Mouradian, N. T. Jahromi, and R. H. Glitho, "NFV and SDN-based distributed IoT gateway for large-scale disaster management," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 4119–4131, 2018.

[121] Z. Mlika, E. Driouch, and W. Ajib, "A fully distributed algorithm for user-base station association in hetnets," *Elsevier Computer Communications*, vol. 105, pp. 66–78, 2017.

[122] D. Bertsekas, *Convex optimization theory*, vol. 1. Athena Scientific, 2009.

[123] A. Zappone, E. Björnson, L. Sanguinetti, and E. Jorswieck, "A framework for globally optimal energy-efficient resource allocation in wireless networks," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3616–3620, 2016.

[124] "Sage tool for game theory." https://doc.sagemath.org/html/en/reference/game_theory/index.html, note = Accessed: March 31, 2023.

[125] "Gambit tools for game theory." http://www.gambit-project.org/, note = Accessed: March 31, 2023.