Review

Shared Sensor Networks Fundamentals, Challenges, Opportunities, Virtualization Techniques, Comparative Analysis, Novel Architecture and Taxonomy

Nahla S. Abdel Azeem ¹, Ibrahim Tarrad ², Anar Abdel Hady ^{3,4}, M. I. Youssef ² and Sherine M. Abd El-kader ^{3,*}

¹ Information Technology Center, Electronics Research Institute (ERI), El Tahrir st, El Dokki, Giza 12622, Egypt; nahla@eri.sci.eg

² Electrical Engineering Department, Al-Azhar University, Naser City, Cairo 11651, Egypt; tarradif@gmail.com (I.T.); mohiyosof@yahoo.com (M.I.Y.)

³ Computers & Systems Department, Electronics Research Institute (ERI), El Tahrir st, El Dokki, Giza 12622, Egypt; anar_Abdelhady@eri.sci.eg

⁴ Department of Computer Science & Engineering, School of Engineering and Applied Science, Washington University in St. Louis, St. Louis, MO 63130, 1045, USA; anar@wustl.edu

* Correspondence: nahla@eri.sci.eg

Received: 19 March 2019; Accepted: 7 May 2019; Published: 15 May 2019

Abstract: The rapid growth of today's technological world has led us to connecting every electronic device worldwide together, which guides us towards the Internet of Things (IoT). Gathering the produced information based on a very tiny sensing devices under the umbrella of Wireless Sensor Networks (WSNs). The nature of these networks suffers from missing sharing among them in both hardware and software, which causes redundancy and more budget to be used. Thus, the appearance of Shared Sensor Networks (SSNs) provides a real modern revolution in it. Where it targets making a real change in its nature from domain specific networks to concurrent running domain networks. That happens by merging it with the technology of virtualization that enables the sharing feature over different levels of its hardware and software to provide the optimal utilization of the deployed infrastructure with a reduced cost. This article is concerned with surveying the idea of SSNs, the difference between it and the traditional WSNs, the requirements for its construction, challenges facing it, and the opportunities that are provided by it, then describing our proposed architectures. As a result of using virtualization technology as a basic block in building SSNs, using different types of virtualization will produce different types of SSNs that will give different usages to it. This article proposes a novel approach of taxonomy for SSNs that is based on the used virtualization techniques, and it describes the needs and usages of each one. It presents a wide array of previously proposed solutions comparing them to each other and a brief description of the issues addressed by each category of that taxonomy. Additionally, the shared sensor architecture and shared network architecture were depicted. Finally, some of its applications in some daily life fields are listed.

Keywords: IoT; shared sensor networks; virtualized wireless sensor networks (VSNs), sensor middleware

1. Introduction

The revolution in modern communication produced an innovative technology, the Internet of Things (IoT) [1–3], which can connect any physical objects all over the world and exchange data among them. These objects can be sensed and controlled through its infrastructure, which allows users to

own all services across the world through small electronic devices. This technology is radically based on Wireless Sensor Networks (WSNs), because it provides users with the ability to interact with their environment and react to real-world events. It consists of a large number of tiny sensor nodes that are randomly deployed over a specified environment, to track a certain phenomenon and give the administrator the ability to instrument, observe, and react to it based on different types of routing protocols [4,5]. These traditional WSNs are usually deployed to serve a certain application [6–12] or offer certain services, thus they are considered to be domain specific and task oriented networks. This leads to the redundant deployment of WSN infrastructure when we have to apply multiple applications over the same environment. A new emerging technology has been adopted to overcome this drawback; virtualization technology, it handles the abstraction of the deployed infrastructure from the offered services over it using different virtualization techniques to achieve different goals, as will be described later. By applying virtualization techniques over WSNs, it confers it the ability to be shared among multiple concurrent applications; this way, we will achieve better infrastructure utilization, these new networks are called Shared Sensor Networks (SSNs). There are many benefits for this emerging networks: First, as previously mentioned, it is a commercial solution that allows for multiple applications to share the same network. Second, the application developers will be isolated from the infrastructure technical details. Third, multiple vendors can work together under its umbrella. Fourth, a new type of electronic clouds will appear named cloud of sensors. Fifth, leasing infrastructure will be available.

The idea of SSNs was surveyed in some previous papers, from different aspects, like in [13], where the authors considered the global meaning for each terminal of it without any actual technicalities. In [14], the authors provide readers a profound state of the art for SSNs, but they did not propose taxonomy for it. While, in [15], the authors provided a systematic review for it with a taxonomy based on the used mechanisms only, though they neglected the hybrid solutions for SSNs.

This paper was decided to introduce SSNs from a basic perspective that is based on the used virtualization techniques, because, without virtualization, there will never be a SSN. Accompanying this taxonomy, we support readers by the terminologies that are presented and the appropriate diagrams to visually demonstrate architectures for SSNs. Thus, this article is concerned with describing these promising networks that will open a wide area for the tiny constrained devices to play a major rule in the fields of IoT, Cloud of Sensors, and Big Data. Therefore, it mainly targets the following:

- Giving the reader a global vision of SSNs.
- Clarifying the main differences between SSNs and traditional WSNs.
- Designing a novel taxonomy for SSNs.
- Designing a novel architecture to identify how the sensor node should be in the shared environment.
- Designing a novel architecture to identify how the whole network should be to serve the shared environment.
- Supporting readers with a comparative analysis between the available solutions along with the main differences, objectives and services supported by each type to help decision makers decide which type of SSNs is suitable for which situation.

The rest of the article is organized, as follows. Section 2 provides a general overview of SSNs. Section 3 provides the novel taxonomy for SSNs that organize the types of SSNs that are based on the used virtualization techniques. Section 4 provides a brief opportunities for SSNs. Section 5 provides some applications for it. Moreover, tables to summarize the surveyed work for better comparisons in the identified criteria are provided. Section 4 presents a number of opportunities that are provided by SSNs. Section 5 presents a number of application that are based on SSNs. Section 6 provides some of the real life projects in SSNs. Subsequently, finally, a conclusion is given.

2. A General Overview on SSNs

SSNs are the new paradigm of WSNs, which gives it the shifting start from an application oriented platform to a shared system infrastructure platform. Virtualization technology is the basic stone technology that provides this facility. Where it targets the conversion of the actual hardware into multiple logical units, to make it available for concurrently sharing among multiple applications and services. By applying it over WSNs, it becomes able to work as multiple logical sensor networks, called Virtualized Sensor Networks (VSNs). These virtualized networks can be offered to the end user, who will be abstracted from the detailed infrastructure, and directly use it for the required applications. Thus, the SSNs can be defined as a shard heterogeneous system, which uses virtualization techniques to federate a group of WSNs or to participate it. That is to allow for multiple concurrent applications to share one hardware infrastructure provided by one or more vendors, operating systems, or communication protocols, etc.

2.1. Fundamental Difference between WSNs and SSNs

As mentioned above, the SSNs are an emerging paradigm of traditional WSNs; therefore, this section will provide the fundamental differences between them, by doing so, we can obtain a better understanding of the specific distinguish between them.

- **Components:** WSNs are composed of a large number of distributed sensor nodes over a certain environment of interest, while the SSNs are composed of a number of heterogeneous WSNs from different environments.
- **Ownership:** For traditional WSNs, the ownership of the hardware infrastructure and the service that is offered for the users is the same authority; on the other hand, the hardware infrastructure of SSNs and its offered services are two different authorities.
- **Platform Dependency:** Same sensors' hardware, same operating system, same communication protocols, and every sensor node are preferred to be from the same model, all of the previous points are basic principles in designing a traditional WSN. Thus, if the sensor hardware is expired, it cannot be replaced by a newer version, or a matchable hardware from another vendor. Accordingly, traditional WSNs must use the same platform, thus they are considered homogeneous systems. On the other hand, the SSNs are heterogeneous systems, because they can operate with sensors of different hardware, different Operating Systems (OSs), and different protocols.
- **Virtualization Technology:** The onset of WSNs is interested in improving certain specifications as cost reduction, resource optimization, standardization, scalability, etc., but neglected the number of applications that will be served by the deployed infrastructure, thus traditional WSNs do support it. However, in SSNs virtualization is the foundation stone for it, which depends on the separation between the deployed infrastructure and the offered services over it. Where it converts the deployed infrastructure into multiple logical units, so the users can access it as if he/she were directly using the physical resource.
- **Code Modularization:** This is concerned with the written code of the application and the code of the operating system and communication protocols. Since, traditional WSNs only handles one application per network, the code is installed in the sensor node before the network deployment. On other hand, SSNs serve multiple applications on the same infrastructure, thus there are two separate types of written codes: one to control and manage the infrastructure owned by the service provider and the second is to control and manage the applications served by that infrastructure owned by the application developers.
- **Number of Applications:** As mentioned above, WSNs were not concerned with the number of applied application over it, thus it was usually deployed to only serve one dedicated application, but the SSNs were proposed to overcome this drawback; therefore, they concurrently serve multiple applications over the same infrastructure.

- **Connectivity:** Sensor nodes in traditional WSNs use physical wireless communications links, but the VSNs in SSNs use logical communication links between them.
- **Resource Sharing:** All of the nodes in traditional WSNs are dedicated to serve one application, so, when there is a need to apply an additional application, additional nodes have to be added to serve the new application, but in SSNs there is no need to deploy additional nodes, because it already uses the virtualization technique to allow for resource sharing over nodes.
- **Redundant Infrastructure:** Infrastructure is dedicated for only one application in traditional WSNs, it has to be duplicated if there is another application needed to be served over the same environment. SSNs solved this problem, which is able to share the network resources among multiple concurrent applications.
- **Infrastructure/service:** Infrastructure and service are considered to be one entity in traditional WSNs, but they are totally separated in SSNs using the concept of virtualization.

These differences are surveyed in Table 1:

Table 1. The main differences between Wireless Sensor Networks (WSNs) and Shared Sensor Networks (SSNs).

Criteria	WSN	SSN
Component	Sensor nodes	VSNs
Ownership	Single Authority	Multiple Authorities
Platform Dependency	Homogeneous platform	Heterogeneous platform
Virtualization	Not support	Support
Code Modularization	Single Code	Multiple Codes
No. of Apps	Single	Multiple
Connectivity	Physical	Logical
Resource Sharing	No	Yes
Redundancy	Yes	No
Infrastructure/Service	One identity	Two identities

2.2. Fundamental Requirements for Building SSNs

There are many solutions that are provided by virtualization to convert the application oriented WSNs into the multi-concurrent applications SSNs; these solutions must support the following requirements [14,16] to achieve this goal:

- **Coexistence:** the physical infrastructure of SSNs should have the ability to understand the hypervisors, which gives it the ability to convert the actual resources to virtual ones. In SSNs there are two layers of coexistence. The first one is over the sensor node itself and it is called Low Level Virtualization (LLV); it ensures that the sensor nodes can support the concurrent execution of multiple applications. The second one is over the whole network and called High Level Virtualization (HLV); it is concerned with the ability of sensor nodes to dynamically form groups to perform the isolated and transparent execution of application tasks in such a way that each group belongs to a different application.
- **Heterogeneity:** it should be platform-independent that allows for multiple independent sensor vendors to coexist in the same physical infrastructure. Additionally, it should be able to hold various protocols and algorithms implemented by different SVNPs, and different Quality of Service (QoS) requirements, topologies, security levels, service types, and etc.
- **Resource Discovery:** there must be a resource discovery mechanism to discover the available resources for allocating them to build a real shared environment. Thus, they could be managed and reserved for applications.

- Resource Sharing: it should be able to share all network resources (wireless radio, hardware, and software) over the network based on the rights given to the end user.
- Isolation: it must ensure that the coexisting VSNs are isolated from each other to guarantee privacy, security, and fault- tolerance, thus any configuration changes, topology changes, customization, or faults over one VSN do not affect the others.
- Flexibility: service providers should have enough authorities over the assigned region for them inside the SSN to change different network aspects, such as network topology, routing, control protocols, and forwarding functionalities, which are independent of the underlying physical network and also without affecting the other service providers.
- Scalability: infrastructure providers should have a scalable physical network, which allows for modifications, such as adding VSNs to support the expected increment in coexisting VSNs, without affecting the given performance.
- Manageability: due to the separation process between the infrastructure provider and the service provider, the manageability to the granted resources must be given to the service provider, to allow for him to manage, configure, and allocate VSNs, e.g., routing tables, virtual resource scheduling, admission, and even modifying protocols, etc.
- Programmability: a service provider needs programmability to implement customized protocols and deploy diverse services, according to the end user application requirements under secure programming paradigms with a considerable level of flexibility.
- Mobility: it should support the internal mobility between virtual resources in the same infrastructure and the external mobility between multiple service providers. Thus, it should have solid mobility management solutions for achieving these goals.
- Legacy Support: is the compatibility with the older versions of hardware and software, in SSNs, this compatibility can theoretically be obtained by grouping them in a separate VSN.
- System Recovery: in the case of system failure or errors in the logical representation of the hosted VSNs, how could SSNs recover the affected VSNs? Additionally, how much time is required to retrieve its original status? It should have mechanisms for resource and service discovery, resource allocation, and task allocation and scheduling.
- Support for application/service priority: apps/services should have prioritized execution mechanisms.

2.3. Challenges behind Design of SSNs

Despite the numerous opportunities for SSNs, its spreading is still constrained by several challenges [16,17]; this section will briefly describe some of these challenges, as follows:

- Isolation: is a basic issue in SSNs as it allows the logical separation of VSNs, while they coexist in the same physical infrastructure. It ensures that the technical changes occurring in one VSN (configuration, customization or topology change) do not affect other VSNs.
- Interfacing: is the connectivity description among different entities overall the SSN as the interfaces between different infrastructure providers under the same SSN, which is necessary in information sharing among them and interfaces between infrastructure provider and service provider required before creating the VSNs and to express their requirements of resources to serve end users. Additionally, the interface between the service provider and the end users is needed to explain the communication between them. Thus, the interfaces should consider communication delays and reliability among diverse entities and should also be carefully designed to become compatible with different kinds of requirements.
- Constrained sensor resources: as known sensor nodes are resource constrained devices, this problem affects the design of virtualization solutions.

- **Embedded VSN:** refers to the static and dynamic allocation of virtual sensors and it links the physical sensors and paths between them. This issue is a big challenge due to the constrained sensors resources. Add it to the previous point (it not related to it)
- **Resource discovery:** due to the large number of resources over SSNs, which are distributed over multiple infrastructure providers, there will be a real need from infrastructure providers to discover the active and passive resources in their physical network, so that may reserve some resources for its own use and to be able to share reachability information among them to establish links (interfaces) between them.
- **Resource Allocation:** the SSNs resource allocation schemes are responsible for deciding how to get an optimal placement of a VSN over the physical sensors' network. The users may require resources that are placed in multiple VSNs, in this case, the service provider must assign them to that user while hiding the details from him, and this issue needs a strong management mechanism.
- **Resource Scheduling:** due to the large number of shared resources over the SSN, the infrastructure providers need a strong mechanism to schedule them for the reservation process of the hosted applications.
- **Naming and Addressing:** as described above, the SSN is a heterogeneous system, which hosts different technologies, shares different resources and offers different services which are concurrently served by it; thus, the physical and logical network representation should adopt a strong addressing scheme that identifies these diverse system identities.
- **Security:** a SSN is based on abstracting the real network infrastructure into multiple virtual sensor networks to provide concurrent services over them to a large number of users, thus SSNs must provide strong authenticity mechanisms to give each network entity (IP, SP, app. End user, etc.) enough privileges to execute its dedicated tasks without affecting the privacy of other entities.
- **SSNs Marketing:** since SSN is an emerging paradigm of sensor technology, its economic model is still an open issue, because it contains multiple entities that share it, starting with the infrastructure provider, which will sell the physical network until the service provider will sell the service to the end user, each entity will share the market model in its own responsibility.
- **Controlling Shared Resources usage:** infrastructure providers should provide an accessing mechanism to control access to its physical network, that is to improve the security and guarantee that the reserved resources to the service provider will not exceed the physical structure. Additionally, the service provider needs to provide an access control mechanism to its resources to ensure user's privacy and security. Resource sharing is mentioned before (it described in the fundamental requirements here it described as one of the challenges).
- **Network Management:** it is concerned with managing a virtualized environment is a complex issue, as the virtual network may span over multiple physical networks from different infrastructure providers. Or, it may participate in one physical network from one infrastructure provider. The infrastructure provider may offer an infrastructure for multiple service providers, or multiple infrastructure providers may offer an infrastructure for one service provider, also the aggregation or dissolution of virtualized networks between different network entities and service providers may need to change resources requests according to the user's requirements. Moreover, it has to manage the collected information from diverse devices of different partners, while saving its privacy and removing conflicts and redundancy from it, on another level it has to manage user mobility from one physical location to another. All of that and other many cases may be in concern. Thus, there will be a real need for a scalable and flexible management system.
- **Mobility Management:** providing mobility to the hosted application in a virtualized environment is a complex issue for several issues; the user location management to provide network communication to users by tracking their locations, the service handoff management to provide the service delivery to the user, who may also need location updates in a different VSN by being provided virtual mobility between VSNs.

2.4. SSNs Architecture

The infrastructure of the early stage of traditional WSNs was owned by the publisher, who can dedicate it to a certain application or service. Nowadays, due to the increasing demand on technological applications, it became necessary to separate between the WSN's infrastructure and the offered services. That is to get better utilization of the infrastructure and improve the ability to serve multiple applications. Thus, it became a real need to federate WSNs that belong to different companies and treat them as one Large Federate Sensor Network (LFSN), which is considered to be a multiservice system [15]. Therefore, there are two co-operating partners to implement this issue, the Sensor Infrastructure Provider (SInP) and Sensor Virtualization Network Service Provider (SVNSP) [13], one to federate the deployed sensors' infrastructure from different owners to the service providers, and the other to offer the sensing service to the users, respectively. Thus, the LFSN environment can adjoin a number of heterogeneous sensor networks.

The architecture of the emerging SSNs paradigm is based on separating the sensor infrastructure from the offered service by it, that is to reduce the required establishment cost, manageability, maintenance, programmability, etc., thus, as can be seen from Figure 1, the SSNs architecture consists of four different layers that will be described briefly, as follows:

Figure 1. SSNs Architecture.

Sensor Infrastructure Provider (SInP): it is the responsible partner for federating and managing the WSNs physical infrastructure, it is able to offer the Sensing Infrastructure as a Service (SIaaS) to a multiple of service providers, it is sensing resources that contain sensors from different vendors (companies), among sensors there are Sensor Gateways Routers (SGR) acting as the sink node, all of them have sufficient power supply, memory, processing, etc. Being connected to high speed wireless networks, it could be able to host different Virtual Sensor Gateways Routers (VSGR), which are offered when the customer requests a special service. SInP can define itself by the type of services that it could provide, and offer the SIaaS through a programmable interface to different SVNSP. As can be seen in the Figure 1, large scale federate sensor networks containing different types of sensors are created by multiple co-operated companies to form the infrastructure.

Sensor Virtualization Network Service Provider (SVNSP): it is the partner who can lease the infrastructure resources from one or more SInP; it uses the virtualization technology to share the leased infrastructure resources and deploys it as a VSN to the end users. As can be seen in Figure 1, the SVNSP offer different VSNs to the end user.

Virtual Sensor Network (VSN): it is the logical representation of the required sensing environment offered to the end user by the SVNSP.

Application Level User (ALU): it is the end user application interface to transact the user queries and allow him/her the ability to instrument, observe, and react to events and phenomena that are required for one or more applications.

2.4.1. SSNs Node Architecture

Although there were not enough proposals that describe the sensor node structure in a shared environment, except in [18], which described it from the aspect of protocol stack. Thus we decided to tackle this track, as shown in Figure 2 shows the proposal for main layers of a SSN's sensor node architecture, the lower layer consists of the physical sensor resources, such as a Central Processing Unit (CPU), Radio Frequency (RF) module, memory module, and power module. Layer 2 contains the available sensor operating system.

Figure 2. Proposed SSN Node Architecture.

The hypervisor layer allows managing, abstracting, virtualizing, and customizing sensing resources that the sensor could provide. It consists of virtualization softwares, like Virtual Machine (VM) or MiddleWare (MW), and the virtualization manager, like service, resource discovery managers and energy and security managers. The virtualization software deals with abstracting the hardware resources details from the running application and the virtualization manager is responsible for managing the services and resources that are provided by the sensor node.

Each one of virtualization techniques uses different programming models and different task execution models to achieve concurrent applications over the same sensor node. Initially, we will provide a brief explanation for them. Figure 3 displays the main models for both task execution and the programming models used in node virtualization.

Figure 3. Task Execution and programming Models for the sensor virtualization.

Firstly, the programming models are event-driven, thread based and Mobile agent, as in Figure 3. In the case of the event-driven programming model, the main program in the OS waits for the occurrence of the designed event threshold, which triggers the matching event-handle function that is registered at the OS scheduler, and when an I/O event stops the program, the event-handle function returns the control to the main program without context switching. However, in the case of the thread based programming model, each program contains multiple threads, and when a thread is blocked by any I/O device, it will suspend and need context switching to execute other threads. Therefore, the thread based model is more complex in programming and more difficult to be implement in sensor nodes than the event-driven model. The mobile agent programming model was presented in [19], where it was considered as a special process that can autonomously migrate across nodes by one of two ways, the first one is strong where the migration process transfers both the code and state, allowing for the agent to resume execution at the destination. The second is weak where the migration process only transfers the code, which is to support the diversity of the applications' requirements.

As for task execution models, there are two ways, either sequential or parallel execution. The sequential tasks model involves a consecutive and ordered execution of tasks one after another, while the Parallel tasks model involves the concurrent or simultaneous execution of tasks or threads at the same time. Although sequential execution has a simple implementation, it is considered to be a weak form of virtualization, because tasks have to wait in a queue, but in parallel execution the tasks are executed in time-slices by rapidly switching the context from one task to another, which will give faster task execution and there is no need to wait for the finishing large tasks, but it needs more complex programming codes.

2.4.2. SSNs Network Architecture

Multiple ideas were previously presented in this issue to provide different solutions in different fields, which, infrastructure-wise, are based on the idea of virtualization, like Cloud of Sensors (CoS), IoT, and SSNs. One of the early proposals in this field was [17], where the main HLV idea handles the separation between the sensor service provider and the owner of the sensor infrastructure, it demonstrated challenges and opportunities without any technical details regarding how to build the that environment. The platform of SenShare [20] proposed an overlay based solution that allows for multiple concurrent applications to run over the shared WSN, which is by grouping sensors that are separately associated with each application in a virtual way. One of the best proposals in this filed was a multilayer WSN virtualization architecture that is presented in [21,22], where it proposed an architecture of four layers, with the support of two data and signaling paths, also with five interfaces to define data and signaling priorities. The first layer is the physical layer, which represents the physical deployed sensors, followed by a virtual sensor layer that provides a logical sensor representation, the third layer is a virtual sensor access layer that provides a unified interface for the next layer, finally the upper layer is the overlay layer that represents a separate overlay for each hosted application, this architecture was implemented over a fire monitoring scenario to obtain some measurements that were presented to study the overlay creation delay, sensor sending delay and the delay of fire alert notifications from the deployed sensors. The results show that most of the delays occurred due to LAN setting. This architecture was extended in [23] to give it the ability to interact with cloud systems, the authors also implement a prototype for this extension and they found that cloud computing eases the service addition and management over virtual networks, but many mechanisms are still needed. The concept of fog computing was introduced in [24] due to the non-scalability of the previous solution, where the proposed architecture is based on the master and slave layers, the first one being responsible for control functions and the second being responsible for flow and resource managements using virtual gateways, but there were not any maintained simulation or implementation for this work. In the field of CoS many solutions were presented to provide the sensor virtualization that is required as a base for building the sensor cloud as in [25], where a three layered architecture was presented: sensor centric, middleware, and client centric. The sensor centric layer represents the physical network deployment, the middleware layer represents the virtual sensors and the required management for them, and finally the client centric layer that represents the multiple user interface. Some studies are concerned with the network hypervisor itself, like in [26], it studied it as a corner stone for the technology of CoS, and proposed an architecture that describes its functionality in managing, abstracting, and virtualizing the resources of the deployed infrastructure, under three main units, which are: the virtualization unit, abstraction unit, and node manager unit. One of the most recent ideas for HLV was SoftWater architecture, which introduces the virtualization technology in the field of underwater communication [27]. It provided the network virtualization based on data plane and control plane. Data plane is an open, programmable, and virtualized network forwarding infrastructure, while the control plane aids the tools for network management and application arrangement. Virtualization SoftWater uses two hypervisors: the first one is the network hypervisor that manages resource allocation and management, while the second is wireless hypervisor that handles resource scheduling and policies management.

According to the previous proposals for the concept of sharing different applications by a single hardware solution, Figure 4 shows our SSN's proposed architecture. It consists of the traditional WSN infrastructure layer that is concerned with sensor deployment and in the maintenance and sensing of the desired phenomena. It is followed by the network hypervisor layer which abstracts the infrastructure of the lower layer using an abstraction manager, and then deals with it as a global resource. The hypervisor uses multiple managers, such as infrastructure manager, which contains information about the whole infrastructure, (sensor, geographical area, hardware types,...), resource discovery, scheduling, and allocation managers to discover and allocate the available resources, and then schedule them to be dedicated to a certain VSN for a certain duration, which is controlled by a resource hosting manager that contains information regarding hosted applications and users. VSN manager is concerned with the creation and removal of VSN hosted by the network. A security manager is responsible for user authentication and privileges on the network. Finally, spectrum manager manages access to the available spectrum, organizes them based on the requirements of served VSNs, and then provides a controlled isolation between the produced VSNs [28]. Thus, the network hypervisor provides global management and monitoring for the SSN through infrastructure, spectrum, and RF managers. Simply, the network hypervisor can be defined as a software that is responsible for separation between the deployed infrastructure and the offered services over the network, which is done by converting the deployed infrastructure into multiple logical units, called VSNs, while providing the isolation between them to keep the security and privacy. By this way, users can access it as if he/she were directly using the physical resource, and get the optimal infrastructure utilization by serving multiple concurrent applications and services. Thus, it handles the deployment of new applications over the network (without interrupting the current ones), information sharing, and data transmission among them, with an optimal routing cost and better energy conservation.

Figure 4. SSN Network proposed Architecture.

3. Novel Taxonomy for SSNs

One of a few previous taxonomies was presented in [29], where it provided a classification of SSNs that is based on the type of used middleware. The authors classified middleware into three main categories to guide users to find a way to select the suitable middleware for his application, as shown in Figure 5.

The first category, named WSN-control middleware, was developed to run over the sensor operating system. The second category, internet sharing middleware, was developed to manage the shared data among internet-based sensors. The third one was WSN and Internet control middleware was developed to provide a complete integration between WSN and internet. However, the used virtualization technique to build the middleware was not declared. Thus, this section will provide a new taxonomy for SSNs based on the used virtualization techniques. Applications from domains, such as smart cities, smart health, smart homes, green computing, and pervasive computing, can

potentially use the SSN virtualization concept for cost effective solutions. New trends, like mobile WSNs, participatory/crowd-based sensing, cloud-based remote sensing, and vehicular networks, can also benefit from this concept. This taxonomy will guide users to find out how the used virtualization technique can change the overall network behavior. That is because the virtualization technology is the foundation stone in creating hypervisors, where it is used to provide an execution environment that hides from the running applications the fact that they are operating in a shared infrastructure, thus this section will focus on classifying in details its different categories to get the prospective models of SSNs. See Figure 6.

Figure 5. Taxonomy of Middleware in SSNs.

Figure 6. Taxonomy of virtualization in SSNs.

3.1. Low Level Virtualization

This level of virtualization is applied over the sensor node itself, thus it may be also called node level virtualization, which is concerned with abstracting the node's hardware to make it shared among multiple concurrent applications. Thus, sensor becomes a multipurpose device. Therefore, LLV can be defined as software that gives the sensor node the possibility to share its hardware between multiple applications concurrently without interrupting each other. This level has three main branches: operating system, virtual machine, and middleware, see Figure 7. A brief discussion regarding the well-known software in addition to a comparative analysis between them will be presented in Table 2.

Figure 7. Low Level Virtualization (LLV) of SSNs.

3.1.1. LLV Based on Operating System Virtualization Solutions

Multitasking OSs take the responsibility of virtualization technology as part of its tasks, it may use one or both of the two available programming models: event-driven and thread based. This section will provide a list of examples of them briefly.

- TinyOS [30]:** is an embedded, component-based operating system, it is tiny because it uses fewer than 400 bytes, it supports an event driven programming model. TinyOS consists of a set of reusable system components along with a task scheduler, which represents a specific set of services, which are specified by interfaces, and each interface defines how the component directly interacts with other components, also the component is considered as an independent computational entity. Thus, it allows for the application developers to develop custom components according to the applied applications and links them to TinyOS's library, these components are written in nesC, which is a dialect of C language optimized for the memory limits of sensor networks, and it has three main abstractions commands, events and tasks. Commands are calls to perform a certain service, events are generated as a reaction to the service implementation and the tasks are produced as functions posted to TinyOS scheduler from commands or events. It may not necessarily be the most viable for SSNs, because it is mainly event-driven. TinyOS features are presented in [29], like reducing the code size by 60%, the timer component causes reduction in CPU utilization by 38%, and the interrupt and task switching also take less time when compared to SenSmart [30]. However, due to the actual added value of this system to the field of SSNs the following OSs evaluated their performance, referring to it as [31,32].
- SenSmart [31]:** is one of the most recent multitasking OSs for the resource constrained sensor platform, as implemented on MICA2/MICAz motes. It is based on the event-driven programming model, and thus follows a sense-and-send workflow model. This further supports its use in SSNs. The executable image of multiple application programs are written at the base station using nesC, where the compiler creates the binary code and memory usage information, and then the rewriter converts them into a normalized code program, which can cooperate with the kernel then links them together with the kernel runtime, and then publish it to the sensor node, as shown in Figure 8.

Figure 8. SenSmart Application development process.

Table 2. Characteristics Summary of SSNs' LLV.

Solution	Solution Type	Program-Ming Model	Source Type	Task Execution	Program-Ming Language	OS & Apps	Virtualizati-On Layer	Sensor Reprogramming	Real Time Support	IoT Support	Scheduling Mechanism	Port-Ability	On the Fly Reprograming
SenSmart 2013	OS	Event Driven	-	Sequential	nesC	Tightly coupled	OS	All System	x	-	Hybrid	x	√
RIOT 2013		Thread Based	Open source	Developer-based	Diverse C/ C++ based	Loosely coupled	OS	-	√	√	Priority-Based	√	-
SenSpire 2011		Hybrid	-	Thread-sequential	CSpire/C	Tightly coupled	OS	All System	x	-	Priority-Based	x	-
MANTIS 2005		Thread Based	Open source	Simultaneous	C	Loosely coupled	OS	Inject updates	x	-	Priority-Based	√	√
LiteOS 2008		Hybrid	Open source	Simultaneous	C	Loosely coupled	OS	Inject updates	x	-	Hybrid	√	√
PAVENET 2012		Thread Based	-	Sequential	C	Tightly coupled	OS	All System	√	-	Priority-Based	x	-
Contiki 2004		Hybrid	Open source	Simultaneous	C	Loosely coupled	OS	Inject updates	x	√	FIFO	√	√
TinyOS 2005	VM	Event Driven	-	Sequential	nesC	Tightly coupled	OS	All System	x	-	FIFO	√	-
Maté 2002		Event Driven	-	Sequential	TinyScript	Tightly coupled	OS + VM	Inject updates	X	-	-	-	-
VMSTAR 2005		Thread oriented.	-	Hybrid	Java	Tightly coupled	OS + VM	Inject updates	x	-	-	√	-
Squawk 2006		Thread-oriented.	License	-	Java	Loosely coupled	VM	Inject updates	x	-	Interrupt-Based	√	-
EVM 2013		-	-	Virtual Task	FORTH	Tightly coupled	OS + VM	All System	√	√	-	-	-
Darjeeling 2009		Thread-oriented.	open source	Simultaneous	Java	Tightly coupled	OS + VM	Inject updates	√	-	Round-Robin	√	
Impala 2003		Event Driven	-	Sequential	-	Loosely coupled	MW	Inject updates	x	-	-	x	√
Agilla 2009	MW	Agent based	-	Sequential	Low-Level (asemply)	Tightly coupled	OS + VM	Inject updates	x	-	Round-Robin	-	√
TinyLime		Agent based	open source	-	Java	Tightly coupled	OS + MW	Inject updates	x	-	-	-	√
TinySOA		-	-	-	Any language	Loosely coupled	OS	-	√	-	-	-	-
MyHealth Assistant		Event Driven	open source		-	Loosely coupled	MW	-	√	√	-	-	-

The application tasks are scheduled with preemption, and they take a dedicated memory region and time slice, as shown in Figure 9. During the execution process, tasks are stored in logical memory addresses, which are mapped into physical memory addresses at the runtime, and when the task is finished, its space becomes free for reclaiming, and when a new task is added for the running tasks, the context is compressed, saved, and then restored from a circular buffer.

Figure 9. SenSmart physical memory organization process.

SenSmart is evaluated for the system operation overheads, application benchmark, and tasks schedule performance. System operation overheads show acceptable values, especially for main processes as context writing, restoring, and switching, but for the application benchmarks SenSmart recorded more CPU usage in comparison to TinyOS. For task's schedule, some delays were recorded in case of running concurrent applications.

- **RIOT [32]:** is the most recent trail to find a flexible open source OS for the heterogeneous, resource constrained hardware in the IoT, which provides an easy development through User friendly API based on ANSI C/C++ languages to run in parallel, real-time multithreading, full TCP/IP stack using 6LoWPAN. Thus, application tasks are independently encoded of hardware and software in order to run them on different devices. This is a key feature that is required for SSNs. The architecture of RIOT is based on microkernel that only requires 1.5 kB of RAM and 5 kB of ROM to run the basic application, it also can operate on 8-bit, 16-bit, and full 32-bit processors. Therefore, it considered a special OS that can serve the diversity of hardware in IoT. To minimize the memory usage RIOT design is based on modular approach [33], and runs the system services and user applications as independent threads. RIOT provides an energy efficient scheduler, which switches to idle thread that enters the system into sleep mode, whenever the tasks are finished to save the overall system energy, only interrupts will end this mode, this scheduler is designed to reduce the context of switching between threads. The kernel operations are divided into small functions, to give the system the ability to work under very low clock speed and facilitate the real-time multithreading. Until now, there is no performance study or comparisons with other OSs.
- **MANTIS [34]:** it is a lightweight, thread-based, and energy efficient operating system, which supports parallel task execution. MANTIS architecture consists of kernel, scheduler, system Application Programming Interface (API), command server, and network stack, as shown in Figure 10.

Figure 10. MANTIS Architecture.

The code size for all of them occupies less than 500 byte of the RAM and 14 KB of the flash, which extends the resources to serve more application threads, it is considered to be a portable operating system across multiple platforms, because its programs can be tested on PC and then ported to the sensor node. Both the kernel and application threads are written in C and they support user remote sensor management. MANTIS presents threads in two levels, system level threads which are responsible for representing the kernel scheduler and underlying hardware as APIs for the higher user level threads, each thread is assigned a priority to allow for the interleaving of tasks in parallel execution; if there is a long thread, then it will be divided into shorter threads to avoid delays. MANTIS OS provides some important features, like prototyping environment, which integrates the virtual environment with the real deployment network, where the code of the application is evaluated on the virtual environment and then deployed on the physical one. Another feature is dynamic reprogramming, which provides a remote reflashing of the sensor's OS, or reprogramming of a certain thread, or changing an attribute within a thread, or debugging a running thread through the command server, which is used for remote debugging of sensor nodes that can be considered as a client part of that server. Debugging process occurs through a remote shell feature, where any user can login to the required sensor node and modify the required threads. The performance evaluation of MANTIS is not studied in [34], but an analysis of its performance was compared with TinyOS in [35], it monitored the main performance parameters, which are memory usage, event processing, and energy usage. TinyOS gives lower memory use than MANTIS, while MANTIS provides better event processing than TinyOS, finally TinyOS is more power efficient than MANTIS.

- **PAVENET [36]:** is a hard real-time operating system for WSNs with enabling preemption. To tackle the preemption overheads and stack/memory conflict management problems, it provides three main functions: a hard real-time task scheduler, a best-effort task scheduler, and a wireless communication stack. Hard Real-Time Task Scheduler is provided for real-time tasks (radio management, sensor sampling, and media access control), which have a high task priority and preempt lower priority tasks, this causes scheduling/switching overheads, to limit their effect the PAVENET OS exploit functions on PIC18 (the only working microchip with this OS), which automatically saves the task context when switching occurs. These functions are called dynamic priority levels and a fast return stack. Additionally, it uses the Best-Effort Task Scheduler that adopts hop-by-hop routing, delay writing to flash memory, and replying to sensor data query, it also uses cooperative task switching to eliminate stack/memory conflict management problems. Wireless Communication Stack consists of physical, MAC, network, socket, and application layers to reduce conflict management occurs between these communication layers, the PAVENET OS hides these exclusive controls and provides a buffer management mechanism, called pbuf, which is shared between these communication layers to easily to handle data flow. Thus, users do not need to consider conflict management, and they can easily develop various communication protocols according to the application demands. The performance evaluation of PAVENET is compared to TinyOS, where the results indicate that the hard real-time tasks at 100 Hz sampling are realized in PAVENET much more precisely than TinyOS, the memory usage in PAVENET is more than TinyOS for sample applications, while the execution times of sample applications is comparable to TinyOS; also, the programming complexity was studied and it was found that the code lines of PAVENET are shorter than that of TinyOS, which is because PAVENET can complement a sequence of tasks in one task. Finally, the task switching overheads were compared with two different OSs, where it was found that it was five times less than MANTIS and was comparable to TinyOS.
- **SenSpire [37]:** It is a hybrid programming model OS, which adopts a multilayer abstraction approach in order to develop networked applications, which support both event driven and thread based programming models using a CSpire object-oriented language to program the application's

tasks, while the kernel tasks are programmed in C. Regarding SSNs, SenSpire ensures that tasks can be programmed as events or as threads, see Figure 11.

Figure 11. SenSpire Hybrid Programming Model.

The higher execution priority is for event tasks. Therefore, the system tasks are written as events that are predictable and easier to maintain, while the application tasks are written as threads with different levels of priorities. The **SenSpire** design is based on three principles, as follows:

- **Predictability:** it uses two-phase interrupt servicing and predictable thread synchronization to guarantee that sensor nodes respond to control messages.
- **Flexibility:** it is achieved by providing a hybrid model for both event-driven programming and multithread programming, to ensure that the sensor nodes remain available for data forwarding when needed.
- **Efficiency:** the OS implementation handles the resource constraints of sensor nodes, it uses stack sharing to reduce memory consumption, and also enables energy efficient network reprogramming in the modular design.
- **SenSpire** network abstraction is provided using a new three layer network stack (radio, resource and sensornet), the radio layer implements device specific MAC protocols, the resource layer virtualizes the radio hardware, scheduling concurrent tasks, and sensornet layer implements the neighborhood management and link estimations, which can be shared among different upper-layer network routing protocols. **SenSpire** is implemented on the Mica2, MicaZ, and TelosB motes to obtain its performance evaluation when compared to MANTIS [34] and TinyOS [30]. The results studied the interrupt latency, which is less than TinyOS, latency of bottom halves experiment is compared to TinyOS, hybrid programming model flexibility that was provided by SenSpire, preemptive kernel, scheduling hierarchy, and scheduler overhead, which has more delay when compared to MANTIS, overall system efficiency, which shows less CPU, and memory utilization when compared to MANTIS, while keeping normalized to TinyOS, and reprogramming efficiency, at which the program code size is less than both TinOS and MANTIS. The results show that SenSpire OS ensures predictable system performance, provides a flexible hybrid model for application programming, and is efficient in resource utilization.
- **LiteOS** [38]: is an open source, Unix-like operating system that fits on memory-constrained sensor nodes. It gives users the ability to operate WSNs, like operating UNIX, which will be easier for people with adequate UNIX background. It provides (1) a hierarchical file system and a command shell interface for user interaction using UNIX-like commands working wirelessly; (2) kernel support for dynamic loading and native execution of multithreaded applications; and, (3) online debugging, dynamic memory, and file system assisted communication stacks. The LiteOS design architecture consists of three subsystems: LiteShell, LiteFS, and the kernel, as shown in Figure 12.

Figure 12. LiteOS Architecture.

LiteShell subsystem is a command shell interface that runs on the base station PC side, it is responsible for interacting with sensor nodes only when a user is present to execute file commands, process operation commands, debugging commands, environment commands, and device commands. The LiteFS subsystem contains the description of the hierarchical file system, its architecture is divided into three modules: RAM, EEPROM, and Flash memory, and RAM is concerned about keeping the opened files and their allocation and data information in both of EEPROM and flash, EEPROM handles the information of the hierarchical directories, finally Flash is responsible for storing the actual data. Although the kernel subsystem is based on thread based programming, it allows for user applications to handle events using a callback mechanism. Kernel also implements hybrid scheduling mechanisms priority-based and round-robin, and it supports dynamic loading and un-loading of user applications. The performance evaluation for LiteShell studies the average command response time that is sent by it for different commands, and the average delay of common network commands. Subsequently, the performance evaluation of LiteOS is compared with TinyOS, which proves that the length of the source code for one program written by LiteOS is less than it when written by TinyOS, while it consumes more memory than TinyOS; which is due to the multi-threading support.

- Contiki [39]:** is a highly portable, multi-tasking open source operating system, and the one of the most popular systems for WSNs, which leads it towards the Internet of Things technology. Where it is able to connect tiny low-cost, low-power microcontrollers to the Internet, with its main features are wireless dynamic loading and the unloading of a code at run time using over-the-air protocol and the possibility of multi-threading on the top of an event driven kernel. Thus, Contiki is highly applicable in SSNs, because it supports multiple applications that are typically independent of the OS and can invariably run on top of it. Applications are written in C language and can be ported to many hardware platforms and updated/installed without reinstalling the entire OS. Additionally, it does not provide a hardware abstraction layer. Thus, communication between processes (applications or services) and hardware is direct and it always goes through it. The Contiki architecture consists of kernel, libraries, program loader, and a set of applications, as shown in Figure 13.

Figure 13. Contiki Architecture.

Kernel is based on protothreads (the concept that shares the features of thread-based and event-driven approaches to attain a low memory overhead). There are three core libraries: core program libraries, loadable program libraries, and specific service libraries; all of them are connected to programs if an application explicitly uses it. The program loader uses the run-time relocation function to load programs into the system, which uses the loader to test the available memory space, if enough of the program will be loaded, otherwise it will be aborted, then the loader will call the program's initialization function. Processes can only be executed through the event handler and poll handler functions. Before actual deployment, Contiki offers a network simulator, called Cooja. The performance evaluation was not studied in the original paper of Contiki OS [39], but a comparison for it with TinyOS was presented in [40], which was concerned with resource usage where the memory usage and task execution time for Contiki programs were found to be higher than them in TinyOS, also the flexibility of both was considered, and proved that the Contiki was more flexible than TinyOS; that is due to the dynamic reprogramming of sensor nodes that only updates the changes, while it updates the whole application in TinyOS.

3.1.2. LLV Based on Virtual Machine Solutions

SSNs offer diverse Virtual Machine (VM) solutions to abstract the sensor node heterogeneity (diverse HW and diverse OSs) from the application developers, and to offer one controlling interface for developing and controlling the required applications; this makes the sensor node platform independent by offering separate execution environments for each application over the operating system. Thus, the VM is responsible for separating the sensor HW platform from the hosted applications by creating a separate virtual environment for each application over the operating system. Therefore, this section will focus on briefly listing them:

- **Maté** [41]: is a tiny event-driven VM built on TinyOS, which is developed to enable energy efficient code propagation for reprogrammable sensor networks to serve different applications over the same network. Therefore, it may not necessarily be the most viable for SSNs, because it is mainly event-driven, offers sequential code execution under a stack of byte code interpreter, and only three concurrent applications can be run on the sensor node. To achieve optimal energy utilization, it divides the program codes into capsules, and each capsule contains 24 instructions with one byte long attached to its version to inform the sensor whether it needs to install this code or not. These capsules are propagated through the network using a distribution scheme among the sensors, where each sensor propagates it to its neighbors. When the instructions reach the destination, it starts to execute it in sequence, until the halt instruction is reached. The instructions are saved in two different stacks, one that is called operand stack to store normal instructions and the other is called return address stack to store the control instructions, which control the program flow. Implementing an adhoc routing protocol over TinyOS tested the performance evaluation of Maté, and the instruction rate, CPU utilization, and infection rates of the network were measured. The results proved that the instruction rate overhead from Maté is 10,000 times more than native TinyOS. While CPU utilization was increased by 33 cycles more than native TinyOS during simple operations, like (and-rand-send-sense) in the worst case. Finally, the infection rate of Maté was only 120 s to update all of the sensors over the network.
- **VMSTAR** [42]: is a software framework that is based on java for building a runtime environment for WSN applications. Developers use it to develop required applications in java language through a rich programming interface; the resulting application can be portable to diverse of hardware platforms. It supports hybrid programming models with sequential and parallel task execution. Regarding SSNs, VMSTAR does not support the simultaneous use of multi-thread application tasks; instead, it only supports single-threaded Java applications. The framework consists of three components: the first one is BOTS component language, which defines the components with specified attribute values, the connection among them, and the conditions that trigger its selection. The second is the composition tool that analyzes the written component and

determines the relations between them to satisfy specific constraints. The third is the updating mechanism, which is responsible for the incremental software updates after deployment, thus it updates the sensor software by accurate changes only. Notice that two models handle the hybrid programming models, the first one is called Select and it defines the sequential thread execution for simple applications by registering the interested events handlers for each application, which execute when the event occurs. While the second is called Action Listener and it handles complex applications by defining both of them, in this model, the applications do not register for events, but it creates an application specific handling list through invoking event handlers. The application codes are stored at the base station, which are also used for orchestrating deployment and update processes. The performance evaluation of VMSTAR started with studying its relative tradeoffs of the threaded interpreter with ROMized classes and SRAM; the results proved that the threaded interpreter improved by 15–20% more than ROMized, while it improved by 60% more than SRAM. Presenting comparisons between VMSTAR followed this and both of TinyOS and Maté OSs; the comparison showed that the performance of VMSTAR is better than Maté, while it is worse than TinyOS, in memory utilization, CPU overheads, and packet delivery ratio.

- Squawk [43]:** is a Java micro edition virtual machine that is designed for embedded systems and small devices. It can run without an operating system, thus it can be directly installed on the sensor hardware. For SSNs, Squawk adopts a different approach when compared to other solutions. First, it provides an application isolation mechanism, which enables multiple application tasks to be treated as Java objects. Thus, applications can have multiple threads, which are managed by the Java Virtual Machine (JVM). Secondly, it contains multiple facilities, like resource management, an authentication mechanism, interrupt handling, network functions to support running of concurrent applications to be represented and treated as java objects, etc. Thirdly, it provides a wireless API for application developers to load, unload, stop, and migrate applications on Sun Small Programmable Object Technology (SunSpot). It uses multiple Java features, like garbage collection, exception handling, pointer safety, and thread library. Mostly, it is written in Java in compliance with J2ME CLDC [44]. As it was designed for memory constrained devices, it implements a split VM architecture, which perform the loading of the class file on a desktop machine to create the representation file using suite creator that converts Java bytecodes to Squawk bytecodes, and then deploys and executes it into the SunSpots. These files are smaller in size than the usual Java class files. Squawk implements green threads to emulate multi-threaded environments; they are managed, executed, and scheduled in the user space. The performance evaluation of Squawk focuses on the memory utilization (RAM & flash) and it shows that Squawk utilizes less memory when compared to the Kilobyte Virtual Machine/Connected Limited Device Configuration (KVM/CLDC) [45]. However, by using different sets of ARM platforms with different CPU and memory sizes, KVM gets better results than the Squawk. Finally, the file size of concurrent applications in Squawk is lower than the file size of the java class files and the JAR files by 37%.
- Embedded Virtual Machine (EVM) [46]:** is the distributed runtime system that aims to abstract multiple connected physical nodes into a virtual component that appears to the users as one logical entity, thus the users do not care about topology changes and they will be able to use same network control algorithm. It seamlessly operates over less reliable wireless networks with topological changes. EVM provides new capabilities as sensor fault tolerance and runtime optimization of resource consumption. The design flow of EVM consists of three layers: Platform Dependent, which contains the EVM design; Platform Independent, which contains Domain-specific EVM Description; and, Design Time, which contains the Control System Specification. These layers enable control engineers to design wireless control tasks with centralized in-network processing in a manner that is both highly platform/protocol independent and extensible to different control domains. The performance evaluation of the EVM was illustrated on several practical case studies in industrial automation while using the FischerTechnik model factory consisting of 22 sensors

and actuators. The EVM's adaptation to unplanned link failures while keeping the system's response similar to that in the initial topology was studied.

- **Darjeeling** [47]: is the first open source Virtual Machine that is based on java for resource constrained devices (2–10 KB of RAM), It is considered for SSNs, because it is a completely thread based programming model that can be ported in three different platforms with three different operating systems. Darjeeling provides a group of features, like light-weight threads, dynamic memory management, and exception handling, while targeting to optimize utilization of the constrained resource. It uses an offline tool, called infuser, to convert the java bytecode into custom bytecode and links between them through loadable modules, called infusions. Thus, by repeating that procedure, multiple applications can run concurrently. The performance evaluation of Darjeeling was run using three platforms with different microcontrollers (ATmega128, MSP430), operating systems (TinyOS, Contiki, LiteOS [38]), and radios (CC1000, CC2420, nRF905). This study proved that Darjeeling is slower than native C by 50%, while the static linking was reduced by 55–83%, and the memory efficiency designed for a 16-bit architecture stack space was reduced by 27% for a real-world environmental monitoring application.

3.1.3. LLVM Based on Middleware Solutions

SSNs middleware are designed to overcome the heterogeneity among the different types of sensors to facilitate the concurrent application development and operation. This section will briefly discuss the middleware approaches, which provide hardware abstraction and hide the heterogeneity of sensor nodes in LLVM.

- **Impala** [48]: is a lightweight runtime middleware for resource constrained systems as WSNs, it was a part of ZebraNet [49] project that aims to provide an improved tracking technology with efficient energy utilization and to improve system reliability, performance, and energy-efficiency. It provides a good programming interface and it supports multiple applications, but only allows one application to run at a time based on event-based programming model. It aims to enable applications' modularity, adaptivity, and repairability in WSNs. It also allows for dynamically applying software updates to the sensors and to the running system. It provides an interface for on-the-fly applications adaptation in order to improve the performance, energy-efficiency, and reliability of the software system. The impala system architecture consists of two layers: the upper one contains all the application protocols and programs for ZebraNet project, which are responsible for gathering environmental information, and then routing it to a centralized base station, while the lower layer contains three agents: (1) Application Adapter, which adapts the application protocols to different runtime conditions to improve performance, energy-efficiency, and robustness based on application and system parameters that are defined to represent the runtime states; (2) Application Updater, which receives and propagates software updates and then installs them on the node; and, (3) the Event Filter, which captures and dispatches events to the above system units and initiates chains of processing. Additionally, Impala has five types of events: timer event, packet event, send done event, data event, and device event, if multiple events arrive, they are processed in a sequential way. The performance of Impala was evaluated while using modern mobile computing devices (HP/Compaq iPAQ Pocket PC handhelds) to test its overhead in real sensor networks. It measures its overhead in event delivery and processing. The results show that Impala events occur less frequently than application events. The time required for routing is reduced by 15% less than applications not using it, while the protocol switching cost is reduced by twice per day for a sensor node.
- **Agilla** [19]: is the first mobile agent middleware for self-adaptive applications that operate in resource-constrained WSNs platforms, it was implemented on top of TinyOS. Its programming model is unique based on agent migration. The Base station represents the application codes as mobile agents and injects them into the network, which can migrate themselves to the sensor nodes that are only relevant to that application, by this way the communication cost is reduced by

moving the processing code to the data, rather than gathering it to a central node. Mobile agents have a stack space, a heap, and three registers, which are used to store the ID of the agent, the program code, and the condition code. Every agent, including the clones, has a unique ID. Agilla's model, Figure 14, was designed to provide an adaptive behavior in SSNs, where each node can support different mobile agents. A mobile agent can be migrated from one node to another with one hop at a time, it may be migrated as a clone where a copy of it is transferred and executed at the new node while keeping running on the source node, or it may be migrated as a move where a copy of it is transferred and executed at the new node while no longer being run on the source node. The sensor nodes can support up to four agents at the same time. Thus, Agilla can support the execution of concurrent applications.

Figure 14. Agilla's model.

Figure 15 displays that Agilla architecture consists of three layers. The upper layer contains the mobile agents, as mentioned above, while the middle layer contains the Agilla middleware, which consists of multiple components that are controlled by the Agilla engine, including the agent code, context, tuplespace, and reaction managers, with each of them providing a different service, where the agent manager provides the agent execution state, the code manager provides a dynamical allocation service of code memory for each agent and fetches instructions as the agent executes, the context manager provides an updated list services of neighboring nodes, the tuple space manager provides the local tuple space services that implements the nonblocking operations (both local and remote), and the reaction manager provides a store of registered reaction services by the local agents and determines when a reaction should fire. The performance of Agilla was evaluated using two physical testbeds, which consisted of 25 nodes with different vendors (Mica2 and TelosB) to measure its feasibility and efficiency. The Agilla programming model was demonstrated while using three real applications (fire detection, tracing, and Monitoring Cargo Containers); the results show that the migration latency linearly scales with the number of hops, where the migration succeeded by 99% for three hops and then decreased as the number of hops increased, although the additional overhead for reliable operations is justified by their resilience to message loss across multiple hops.

Figure 15. Agilla Architecture.

- TinyLime** [50]: is a data-sharing middleware for WSNs that aims to provide a centralized approach for data gathering and aggregation that is based on location dependent applications. This implementation is typically plagued by challenge in weak abstraction support, and a lack of dynamic topology. Thus, it is not suitable for SSNs. Where, instead of gathering data using a central station, it offers multiple mobile stations to gather data from nodes in their proximity. It shares the sensor data using a tuple space interface, which hides memory sharing between the applications and sensor nodes, allowing for information sharing among neighbors and supports application tunable energy utilization by giving the user the ability to turn the logging of values on the sensors on and off. The internal architecture of TinyLime consists of MoteLimeTupleSpace, MoteLimeTuple, MoteLimeTemplate, and the condition and aggregation classes, which are needed by a client application, as can be seen in Figure 16. Where MoteLimeTupleSpace presents the illusion of a single tuple space containing sensor data to the client, which is internally divided into two Lime tuple spaces, one to hold sensor data and the other for communicating requests from the client. A user can create a MoteLimeTemplate to access the allowed applications. The performance of TinyLime was evaluated using the Crossbow MICA2 platform (Cisco,; the results showed that the response times varied from 0.0049 s to 0.20 s, depending on CPU load. With the low CPU usage, the average was approximately 0.008 s and, as the number of motes increased, the response time is decreased.

Figure 16. TinyLime Architecture.

- **TinySOA** [51]: is a service oriented middleware that abstracts the low-level details of WSNs from application programmers and it allows them to access it from their applications by using a simple service-oriented API via the language of their choice to facilitate the way of accessing and controlling WSNs, because TinySOA aims to integrate the internet applications and the WSNs. As can be seen in Figure 17, the TinySOA architecture consists of a node, gateway, registry, and server cooperated to serve internal and external services that are provided by TinySOA.

Figure 17. TinySOA Architecture.

Where the node provides the internal services (service discovery, reading data, controlling actuators, and communicating via a gateway from and to the network), while the gateway acts as a bridge between a SSN and the outside world, then registry stores all information regarding the infrastructure, and finally, the server acts as a web service provider and provides a friendly API, where each separate WSN is presented as a separate web service. The performance of TinySOA was evaluated by two approaches, the first was developing simple applications while using different types of programming languages and different levels of programmers (experts and beginners) to ensure that TinySOA overcomes the overload of being aware of a certain programming language or knowing related issues. The second was developing complex real world applications to test and measure the degree to which TinySOA tools can help application developers.

- **MyHealthAssistant** [52]: is a service-oriented middleware for user-centric health monitoring running on a smart phone-mediated Body Sensor Network for multiple medical applications, it aims to mediate between the heterogeneous sensors and multiple medical applications running on a smart phone. It treats the applications as platform independent tasks, which act as a service to consumers and then implements the required services, like sensing, actuating, etc. by a set of united instructions across different platforms. Figure 18 illustrates the architecture of MyHealthAssistant.

Figure 18. MyHealthAssistant Architecture.

The ambient sensors communicate with Android Remote Services via Wi-Fi, while the body sensors communicate via Bluetooth protocol. A sensor module handles sensor communication and

creates a corresponding sensor event, which is forwarded to the message handler, which injects the received event to broadcast channels to inform subscribed applications about new sensor readings. Subsequently, the event composer translates the incoming events into general situations on which the system has to react, and it creates a corresponding derived event, it also checks the accuracy of data and prevents the inaccurate data. The system monitor provides the overall system status and detection measurements for critical situations, such as a low battery level. Overall liveliness of the smart phone needs to be monitored, thus a periodical heartbeat message containing the phone status information are sent to a server, which allows the remote detection of a crashed phone or a bad connection to the network carrier. The performance of MyHealthAssistant MW was evaluated by three different healthcare applications that were deployed with it, the performance metrics are energy consumption, overhead that is caused for the smart phone, and processing time under real-world circumstances. The results shows that, as the energy consumption of MyHealthAssistant increased due to the increasing of message workload, the Android system energy consumption also increases by 30% for the additional communication tasks. The delivery ratio decreased lower than 99.9% when the number of workload increased by 12event/Sec. As the event size increased the CPU utilization was not affected, though MyHealthAssistant decides which channel an event has to be sent only based on its type and it does not inspect the payload, the delivery ratio slightly decreases. Additionally, the results proved that the main contribution of MyHealthAssistant MW is running multiple applications while keeping the impact on the system performance steady, where the CPU utilization and message delivery ratio remain steady while the memory usage slightly increases.

3.1.4. An Insight on the Application of LLV

This section will provide us with the result of studying virtualization in LLV, where Table 2 provides a summarization of some characteristics for each virtualization solution.

At the beginning from the study, it is concluded that the preferred programming model in virtualization solutions is the thread-based rather than the event-driven. Although it is more complex, it provides better execution for the concurrent tasks.

As for the OSs, it is concluded that:

The extremely popular OSs are TinyOS, Contiki, and RIOT; they have a good support community. However, TinyOS provides a weak form of virtualization, because it is based on event-driven programming model, and the relation between the OS and the application is tightly coupled. Additionally, its task scheduler is sequential (FIFO based) and, thus, the task execution runs until entire completion. However, Contiki is a better choice, because it provides many innovative features over the last years that makes it a better platform for IOT. Nevertheless, looking for future trends and large-scale deployments, like smart cities, the RIOT is a fresh approach that provides general purpose solutions for divers of IOT devices, because it uses a real-time thread-based programming model. Threads execution is based on its given priority. The application tasks are codes decoupled from hardware and software, which makes it suitable for a wide range of devices.

All other OSs solutions that are covered by this review are less popular than the three described above as SenSmart, SenSpire, MANITS, LiteOS and PAVENET and can be evaluated, as follows:

- SenSmart: To support SSN, it provides application codes programmed and compiled in binary code for each, then combines them together in a single code image and links it to the kernel to be tested and then generate the final code. This strategy of compiling and linking to the kernel means that there is no separation between the OS and application tasks. Therefore, any application updates should update all the sensor software. While its main drawback is the lack of portability, it is the approach that shows how a better hardware design can lead to an efficient sensor OS.
- SenSpire: It supports a hybrid programming model, where system tasks are implemented as event-based tasks, while the application tasks are implemented as thread-based tasks with low priority than the event-based. Thread-based follows the run-to-completion model, which executes

tasks in a sequential (FIFO) manner. Additionally, programming with Cspire is a learning curve for developers. Due to the application being tightly coupled to the OSs, there is a need for updating all sensor software in the case of any application updates. Finally, it has lack of portability.

- MANTIS: This OS is considered to be suitable for SSNs because it provides a complete thread-based system with a multithreading approach that simultaneously runs application tasks without the need to manage low-level details of the stack/memory. Applications threads are coded in C and they are loosely coupled with the OS. However, it has a weak support community.
- LiteOS: It is highly flexible for SSNs, because it uses a programming environment that is based on C and it allows a hybrid programming model that combines the simultaneous execution of application threads and events through a call-back mechanism. The application is loosely coupled with the OS through call gates pointers.
- PAVENET: In order to support SSNs, it supports thread-based programming and the use of C language. It is possible to ensure varying priority levels via the use of programmed multithreaded applications. However, task execution is sequential, because time-slicing is not supported. Additionally, the application is tightly coupled with the OS, thus any application updates will reinstall all sensor software. However, its main drawback is the lack of portability.

For the discussed VMs, the following is concluded:

- Mate: This VM is based on TinyOS to support concurrency, thus it addresses the main drawback of the original TinyOS, which is the sequential task execution. However, Mate provides application updated injection without replacing the OS. Additionally, it provides a simple mechanism to automatically reprogram the SSNs using code capsules. It is considered suitable for simple event-driven SSNs, because it can define events and their outcomes.
- VMSTAR: For SSNs, it does not support the concurrent use of multi-thread application tasks, but it only supports single-threaded Java applications. However, concurrent events can be handled using action listeners. This can be used to identify high priority threads so that expired threads can be released from system resources to cater for other application tasks. Threads can be easily programmed using java. Application implementation is tightly coupled with the OS.
- Squawk: It does not require an OS in order to run; instead, all of its basic requirements are inbuilt. For SSNs, it provides an efficient application isolation mechanism, which allows concurrent applications to be represented as Java objects. Thus, applications can have multiple threads, which are managed by the Java Virtual Machine (JVM). Additionally, updates are injected on the fly.
- EVM: It is highly viable for deployment in LFSN, which is one of main targets of building SSNs. Where it can abstract physical nodes into virtual nodes. Thus, it provides new capabilities as sensor fault tolerance and runtime optimization of resource consumption.
- Darjeeling: Is an open source java based VM that supports thread-based programming models. For application concurrency, it provides an infuser tool to support the LLV.

For the discussed MWs the following is concluded:

- Impala: It is one of early efforts in the field of MW to support application concurrency in WSNs. However, it has some challenges that makes it be unviable in SSNs, like, it does not support data preprocessing, which is an important component of data management. Additionally, it presents some limitations that are related to its inability to perform code management tasks and the unpredictability of agents in the system at runtime.

- TinyLime: Also, it is one of early efforts in the field of MW to support application concurrency in WSNs. Thus, its implementation is plagued by different challenges, such as weak abstraction support and data fusion, to a lack of dynamic topologies.
- TinySOA: Is developed on top of TinyOS, and it utilizes the platform heterogeneity that is provided by it. It has been devised to specially fit the needs for smaller footprint devices that are used in IoT scenarios.
- Agilla: This MW is based on TinyOS in order to provide the simultaneous execution of tasks to be more applicable in SSNs. However, it adopts a low-level assembly-like language and a stack-based programming model, and that makes programs difficult to modify or to build; this difficulty in programming limits its use for SSNs.
- MyHealthAssistant: Is an open source event-driven MW that is designed for phone-based deployment and focuses on the efficient management of wireless sensor data for sharing between multiple healthcare applications.

3.2. High Level Virtualization

This level of virtualization is applied over the whole network to abstract the underlying physical network then offer it as a group of isolated virtual networks to get the best infrastructure utilization and provide more economical solution. Figure 19 shows the two main branches in HLV, which are the network level virtualization and the system level virtualization.

Figure 19. High Level Virtualization (HLV) of SSNs.

3.2.1. HLV Using Network Level Virtualization

This branch of HLV is applied over the whole network, to participate it into smaller areas. While using two types of solutions, the first is Virtual Network/Overlay Sensor Networks solutions and the second is Cluster-based solutions.

- HLV using Virtual Network/Overlay Sensor Networks solutions

These solutions provide network resources' sharing and facilitate the communication among sensors in different administrative domains. Thus, this section will briefly list them:

- **Multi-layer architecture for VSNs [22]:** this article proposed a three layer architecture to create multiple VSNs over the deployed WSN, which will give multiple applications the ability to run concurrently over this SSN. It uses overlay to facilitate the data exchange among the nodes in different VSNs. It consists of three layers, as seen in Figure 20.

Figure 20. Multilayer Architecture for Virtual Sensor Network (VSNs).

The first layer represents the physical WSN that contains heterogeneous sensor nodes, a new generation of them can participate in the overlay, but the older cannot, thus it also contains a Gates-to-Overlay entity to support the earlier sensors to participate in the overlay, the second layer is the virtual sensor layer that abstracts the concurrent application and management tasks implemented over the physical sensor to become virtual sensors, the third layer consists of multiple overlays, and each one uses some virtual sensors as members that are dedicated for a certain application; it also proposes separate paths for data and control messages. This architecture is based on four principles, the first one is caring about the deployment of the new application as an overlay on the top physical network, the second one is multitasking processes on the physical sensor, while the third principle is the overlay operation on sensors that do not support the middleware, which will delegate the operation to other sensors, finally the fourth principle provides two separated paths for sending two types of data between the infrastructure and the overlay applications, where the sensor's data is sent through data path and control data is sent by the signaling path. There is no performance evaluation presented for that proposed architecture.

- **Managed Ecosystems of Networked Objects (MENO) [53]:** This article presents a new concept that aims to create a smart network architecture that collects all of the internet connected objects using the combination of network virtualization technology and clean-slat end-to-end protocol [54] rather than the usual gateway based solutions that provide security, firewalling, protocol translations, and intelligence by gateways at the border of the Internet and the resource constrained networks.

This combination will provide a new open minded platform, that offers open and secure communication methods and services that are able to connect sensors as any other IP-smart object over the internet, that facilitate the application development and innovation. The proposed layered architecture is shown in Figure 21. It consists of four layers, the first one represents the physical network infrastructure that contains a diversity of network technologies, the second is the logical network layer above it that is implemented to connect the cooperated objects, then the third layer presents a design of novel end-to-end protocols that introduces a common language between all connected objects, and the last layer presents the powerful API for services and application development. This concept has no implementation yet, and there is no available performance evaluation.

Figure 21. Layered Managed Ecosystems of Networked Objects (MENO) Concept.

- **IoT-Virtual Networks (VN)** [55]: Internet of Things Virtual Networks is a secured virtual network for both resource constrained and non-constrained objects that need to co-operate under the umbrella of virtual networks, it is based on the idea that was proposed by MENO, it uses end-to-end communication between networked objects. The IoT-VN approach explores more general cases than MENO, as can be seen in Figure 22. The figure is partitioned, as follows:
 - ❖ Partitioning IoT-VN partitions a subset of the available sensors in the sensor network.
 - ❖ Aggregation IoT-VN combines several sensor networks into one big VSN.
 - ❖ Extended IoT-VN includes non-constrained devices. (d) Hybrid IoT-VN combines partitions from two separate sensor networks and extends them with remote non-constrained devices.

Figure 22. Internet of Things Virtual Networks (IoT-VN) generic cases.

The cooperating members of IoT-VN are connected using virtual links, that use an IDRA framework [56] to implement neighbor detection and tunneling mechanisms. To realize network packet processing for non-constrained devices it uses a click modular router [57], a C++ based framework is also used for routing data over the virtual links, they developed the AODV protocol to be extended to include IoT-VN ID header and network header. To study the performance of IoT-VN they built small networks of constrained and non-constrained devices, and run a simple ping application. The results demonstrate that it is feasible to realize IoT-VN approach in including resource-constrained and non-constrained devices in one virtual network. It still requires much more study to find out its actual performance under different and more complex applications.

- **Multi-Agent System (MAS)** [58]: this approach proposes a solution for two main issues in VSN, they are creation and maintenance, and it proposes intelligent hibernations and offline operations for a sensor node when it joins more than one VSN. A Multi-Agent System is resident on individual nodes to facilitate the practical operation of adaptive VSNs, while ensuring optimal energy consumption. MAS is an agent-based solution that is built on top of Java SunSpot devices

and is based on Agent Factory Micro Edition (AFME) library to create multiple agents on nodes (connectivity agent, coverage agent, hibernation broker agent, etc.), as shown in Figure 23. It allows for message exchange between them based on VSN requirements and agents in each sensor node can communicate with their peers on other sensors in the same VSN to optimize the performance of VSN creation and maintenance.

To overcome the drawbacks of earlier layered VSN agent-based solutions with power management, inflexibility, homogeneous timing constraints, and co-operation with hibernating layers, the MAS approach provides a hibernation broker that organizes the sleep periods for each agent and it decides the hibernation process for some or all of the components of a node, depending on the requirements of active VSNs. The performance evaluation of MAS approach not studied in this article.

- **A Resource Efficient Approach for Concurrent Applications [59]:** This article presented two real world applications to explain the concept of VSNs over the SSN, they approached geographically overlapped sensing applications and underground contaminant plume tracking; the first set of applications are for monitoring different events under spread heterogeneous WSN nodes over a large area and the second one is for tracing a dynamic event with a resizable subset of WSN nodes. A design for dynamic VSNs' creation has been proposed based on two main functions; node membership maintenance and maintenance, the first function handles node membership (adding, deleting, joining, leaving, etc.) and then implements the assigned tasks, but the second one handles data routing among the nodes, which may need the help of nonmember nodes to route the gathered data through the SSN to the desired destination.
- **Melete System [60]:** is one of the first trails towards SSN, as the idea of VSN wasn't clear yet, the proposed system implemented the idea of concurrent applications over the same WSN infrastructure, where all of the nodes initially join a global logical group, called the associated group, and each node stores its code all times; however, it added applications to the system during the runtime that required creating a logical group to serve each of them, thus, based on grouping criteria, any node is able to be a member of a dynamic group that was created to serve a certain application, where a node can serve up to five applications concurrently based on its RAM, while the whole system supports up to 16 logical groups. The technique of dynamic grouping is used to support the flexible deployment of concurrent applications at the network level, where each application is deployed and executed on a subset of sensors that were grouped to serve it based on the application requirements and the node's capabilities. Additionally, the technique of Trickle [61] was used to facilitate the code deployment across the whole network. As Trickle supports code movement only through one-hop, thus the developers of Melete added a multi-hop feature to it, using additional node states, as shown in Figure 24.

The performance evaluation study of Melete System was presented both analytically and in simulation, where it was observed that when multiple capsules were simultaneously requested, a forwarding region might be blocked in its discovery stage by existing forwarding regions around it. Also the code caching reduced the number of forwarded requests by 20–30%, and reduced the completion time by 20–50%. Affected requests by the blocking problem were very low. Melete was better than Trickle in both of traffic and time costs. Network traffic, completion time, and number of requests and code chunks increased with the node density.

Figure 23. Multi-Agent System's (MAS's) Node structure.

Figure 24. Sensor state transition in Melete.

- HLV using Cluster Based solutions:

SSNs provide cluster based solutions over the whole network, where the physical network is partitioned into groups/clusters, and each cluster is dedicated to a certain application, at which each sensor node inside the cluster has a certain role, cluster head, or cluster member. This section will briefly list them:

- **Cluster Tree Based Self Organization [62]:** The proposed scheme was presented to facilitate the creation, operation, and maintenance of dynamic VSNs. When a node detects a desired event, it triggers the root of the cluster to start creating a logical dynamic tree for tracking it in a reactive manner. The VSN tree will logically connect the nodes that are interested in that event to ensure the notification detection, by providing unicast, multicast, and broadcast communications among them. In unicast communication, a hierarchical addressing scheme is used, while using a list for both broadcast and multicast communication schemes. In this way, this mechanism is able to group a subset of sensor nodes under a logical tree to serve a certain application. The performance of cluster tree algorithm was evaluated while using a discrete event simulator, with 5000 nodes being randomly placed in a circular region if the radius was 500 m. The event detection was considered under three different scenarios. The results proved that, as the number of sensor nodes monitoring an event increased, a linear increase of hops would occur. During tracking an event, the exchanged unicast messages between the source and destination are not affected by the network size, while the multicast messages are affected by it.
- **Closed-Loop System for Subsurface Contaminant Plume Monitoring [63]:** The developed system aims to develop protocols and software that supported VSN and also a real time integrated subsurface chemical plume monitoring system. Authors introduce the proof-of-concept study, which presents a phenomena-aware clustering algorithm that creates the dynamic VSNs by selecting the nodes closest to the monitored phenomena, allowing the remaining nodes to sleep to save their batteries for other applications. By this algorithm, the data sent is relative to the monitored phenomena, which reduces the total data sent and saves the global energy consumption. However, for an accurate determination of the event location, the authors present another algorithm, called DRAGON, which specifies the triggered event location to start cluster formation and phenomena tracking from that location. The DRAGON algorithm is executed through three phases, as can be seen in Figure 25.

Figure 25. DRAGON's phases.

The first phase is called summary phase, where there is the starting point of DRAGON algorithm, it computes the center and total of mass, and then distributes them to all clusters containing the event. The second and third phases are called Split and Merge, respectively, and they provide stability for the monitored event, if not they will split and merge it. The performance evaluation of closed-loop system was not discussed in that article.

3.2.2. HLV Using System Level Virtualization Solutions

This is the second branch of HLV that was applied over the whole network to controls different authorities of shared resources, it has two roles: the first one is controlling and managing different authorities for different federated WSNs, which are united to create a large federated VSN, and the second one is controlling and managing different authorities for resulting small scale VSNs, which are produced due to participation of a WSNs.

❖ System Level Virtualization based on Federation:

Federated SSNs occur when multiple WSNs from different administrative authorities are united into a single SSN. As shown in Figure 26a, they are usually connected through the internet. It provides a global administrative policy to manage and control the shared resources from different administrative authorities; it faces two major issues, the first is the need for a resource management mechanism that is responsible for describing, discovering, and reserving the allowing shared resources, the second is the need to provide a way for concurrent tasks allocation and scheduling. Finally, it requires authentication, authorization, and accounting (AAA) to control access to the SSN's resources, enforcing policies, auditing usage, and providing the information that is necessary to bill for services. These combined processes are important in effective network management and security. This section will briefly present federated examples of SSNs.

Figure 26. SSNs. (a) Federation (b) participation.

- Global Environment for Network Innovations (GENI):** It is a federated virtual laboratory for networking and distributed systems research and education project. It introduces a novel suite of infrastructure designed to support experimental research in network's science and engineering [64], starting from a new research on networks and distributed systems design to the theoretical underpinnings of network science, network policy and economics, societal values, and the dynamic interactions of the physical and social spheres with communications networks [65,66]. The U.S. National Science Foundation (NSF) funded GENI. It is composed of a broad set of heterogeneous resources, each owned and operated by different entities (called aggregate providers), as can be seen in Figure 27, by joining the GENI federation, these aggregate providers make their resources available to GENI experimenters, while still maintaining a degree of control and trust that these resources will be used in a responsible and secure manner. The federation of testbeds is based on the concept of testbed virtualization and virtual links between them.

Figure 27. GENI's MAP.

GENI Network architecture is shown in Figure 28. It consists of: (1) The control plane, which is used to discover reserve, access, program, and manage GENI's computational and communication resources; (2) The data plane, which is setup depending on the requirements of the experiment. GENI offers some features to users for facilitating the control of the experiment, such as: (1) Slicing the network, this feature allows multiple experiments to share the same physical infrastructure, while keeping isolation among them; (2) Deep programmability, which gives the programmers the ability to determine the packet paths through the experiment; (3) Network federation and stitching, this feature organizes the shared resources among different network providers; (4) GENI wireless networks, which provide a wireless base station that connects the local rack and the rack through the rest of the network; and finally, (5) Connecting campus resources to GENI. GENI proposes an Open Resource Control Architecture (ORCA) framework for federated infrastructure services, it consists of an Aggregate Manager (AM), Slice Manager (SM), and broker. AM is a resource provider that allocates the virtual resources, called sliver to the user, each sliver has a lifecycle, operational states, and it works as a member of one slice, where a slice is an end-to-end user environment, SM refers to the user interface with AM, and broker handles user authority and applies the system policy that is associated to that user. Thus, SM starts the resource request and then broker issues the ticket, which authenticates the user and gives him the associated privileges over the system, and then AM sets up the slivers that are associated to that request.

Figure 28. GENI's Network Architecture.

- WISEBED** [67–69]: is an academic and research project for building a multi-level infrastructure of interconnected testbeds of large scale wireless sensor networks through nine institutes across Europe. The federation of testbeds is based on the concept of testbed virtualization and virtual links between them, pursuing an interdisciplinary approach that integrates the aspects of hardware, software, algorithms, and data. As can be seen in Figure 29, its architecture consists of the partner's testbeds; each testbed comprises a number of sensor nodes, which are managed and controlled by a portal server, this server exposes the functionalities of the associated testbed to users through the Internet by running overlay software that provides WISEBED API-compatible Web services.

Figure 29. WISEBED Architecture.

Portal servers handle authentication, authorization, and accounting (AAA) service, Network Control, Debugging and Configuration management, Data Acquisition and Query Processing. While the interconnectivity between partner's testbeds is also done by a WISEBED API which handles the federation process among them and makes them work as a single virtual large scale unified testbed. These WISEBED API timeline, as shown in Figure 30. Consist of Sensor Network Authentication and Authorization (SNAA) API, Reservation System (RS) API, Wireless Sensor Network (WSN) API, and Controlle3r API. T/he first API provides the user authentication and authorization, the second API handles shared resource reservation, the third API handles the user interactivity with sensor nodes, and the fourth API uses the serial interface to listen for the output generated by nodes.

Figure 30. WISEBED API Timeline.

- FIT IoT-LAB [70–72]:** is the first open access scientific testbed for testing and managing a real large scale open WSN; it has been developed to evaluate the scalable WSN protocols and applications. This testbed consists of 2728 sensor nodes and 117 mobile robots over six sites across France, as shown in Figure 31. Each site offers a different node and hardware structure, but all they directly communicate by a radio interface, or remotely through the internet and they are available for user through the same API. Each node can be configured as a sink that is able to exchange data among other sink nodes in whole lab. The nodes are either static or mobile; therefore, the user can reserve, monitor, and reprogram them, and also can control the motion of mobile nodes. IoT-LAB infrastructure offers reliable access to all nodes, real-time monitoring, and controlling the experiments, security, and data integrity. IoT-LAB hardware infrastructure consists of a set of IoT-LAB nodes, where each node consists of three elements, as shown in Figure 32.

Figure 31. FIT-IoT Map.

Figure 32. IoT-LAB node.

The first one is the Open Node (ON), which is dedicated to the user during experimentation and totally opened to him/her. The second is the Gateway (GW) that handles the connectivity to the global infrastructure to control and monitor the open node. The third one is the Control Node (CN) that works as a passive or active element related to the open node, it handles monitoring of consumption and values of sensor nodes during the experiment. IoT LAB offers three types of sensor nodes, WSN430 node that is considered as legacy node, while M3 node, and A8 node are more recent. IoT LAB software architecture consists of five layers, as shown in Figure 33. Various types of OSs can be run on the open node based on its capabilities, IoT LAB supports the following OSs: FreeRTOS [73], Contiki, TinyOS, Riot, and OpenWSN [74]. Experimental steps started by building nodes' firmware followed by node reservation and monitoring then launching the application and finally ended by analyzing the resulted data.

Figure 33. IoT LAB Software Architecture.

❖ System Level Virtualization based on Participation:

SSNs participation is achieved when one WSN infrastructure participates in multiple VSNs, as shown in Figure 26b; this section will briefly present some examples of it:

- **WSNTB [75]:** is a testbed for heterogeneous WSNs, it consists of a hardware infrastructure and a software framework that is designed to support power consumption, timing measurements, and distributed management. WSNTB hardware infrastructure is physically implemented in a single site and composed of two clusters of WSNs (octopus 1&2); each cluster has its own radio communication way, and its own gateway, as shown in Figure 34.

This infrastructure is accessible for remote users by a software framework, which consists of a services interface layer, a testbed core layer, and a resource access layer; all of these layers are based on a web portal API through the internet to allow users to build their own WSNs for real experiments and give them the facilities to gather the experimental results in an easy way.

WSNTB has been tested using two programs; the first was Distributed adaptive transmission power control algorithm, at which every node determines the appropriate transmission power to its neighbors using two of popular ways, the RSSI (Received Signal Strength Indicator) and LQI (Link Quality Indicator) [76,77]. The results showed that the algorithm caused a reduction of consumed energy by 20–30%. While the second program was Secure Reprogramming Protocols, which has been implemented to ensure the security of the distributed remote code over the sensor network, there were not any presented results for its performance.

- **Cyber-Physical Networking (CPN) [78]:** CPN is a dual testbed that was developed by University of Nebraska-Lincoln. CPN testbed targets that each sensor joins two separate VSNs over a

traditional TCP/IP tunnel using software and hardware tools that supports remote programming, out-of-band monitoring, power management, and real time virtualization. The hardware infrastructure of CPN participates in two separated VSNs, as shown in Figure 35, they were connected by a TCP/IP tunnel, and each sensor was able to set a peer-to-peer communication with any other sensor in the secondary separated VSNs.

The implementation of the CPN hardware infrastructure participates in two sites each of them contains 45 sensor nodes connected to servers using USB-to-UTP converters, this infrastructure is accessible through software with a web portal interface to facilitate the deployment of sensor applications over a dedicated set of sensors. The software components were: (1) Core Databases, which represent the main software component, which contains the locations and detailed information of the testbed and also stores the resulted data from applications; (2) Persistent Device Naming and Mote Type Detection Subsystem, which identifies the event with the relative node and its path; (3) Application Deployment Subsystem, which subsystem presents a GNU make tool for deployment of nesC applications based on TinyOS. (4) Debugging and Data Collection Subsystem: that handles the collection and debugging of data using TinyOS printf library (5) Testbed Management Interface and Data Visualization Tools: it presents the user API and in addition it also provides remote data collection and remote programming. The performance evaluation of CPN was tested using a novel ping application deployed as a proof-of-concept application for peer-to-peer communication in geographically separated VSNs to test the mean delay associated with communication between them. The results show that the inter-network communication latency is approximately 90 ms, it also notes that the increment of local inter-packet interval causes the increments of latency for local communication.

Figure 34. WSNTB Design.

Figure 35. Cyber-Physical Networking (CPN) Infrastructure.

3.2.3. An Insight on the Application of HLV

This area of research is still considered to be recent, thus not much work has been done on it. Therefore, this section will provide us with the conclusion from studying virtualization in HLV, where Table 3 provides a summarization of some characteristics for each virtualization solution.

Table 3. Characteristics Summary of SSNs' HLV Solutions.

Solution	Solution Type	System	IoT Support	Formation Concept	On-the-Fly Deployment
Multi-layer VSN (2013)	VSN Overlay	-	-	Data & control paths	-
IoT-VN (2012)		Embedded Linux for non-constrained devices TinyOS + IDRA for constrained devices	√	Virtual Links	-
MENO (2011)		-	√	End-to-End Communication	-
MAS (2008)		-	-	Multi-Agent System	-
Melete System 2006		TinyOS+ Maté	-	Dynamic grouping	√
Cluster Tree 2009	Cluster Based	-	-	Dynamic Tree based on phenomena tracking	-
Closed-Loop System 2008		-	-	Phenomena-aware Clustering	-
GENI Life Testbed	Federation	-	-	Virtual Links	-
WISEBED Life Testbed		-	-	Virtual Links	-
FIT IoT-LAB Open access Testbed		-	√	Virtual Links	-
WSNTB 2008	Participation	TinyOS+ Java runtime	-	Cluster – gateway Based	-
CPN 2011		-	-	Cluster – gateway Based	√

- Virtual Network/Overlay Sensor Networks solutions:

The importance of these overlay approaches to SSNs is coming from the ability to transmit data between the nodes without the need to build a new infrastructure. Thus, the overlay approach is appealing to achieving the virtualization in SSNs.

- Multi-layer architecture for VSNs: This three-layer architecture is based on overlay network virtualization, which provides scalability in all layers. It does not require the re-deployment of the entire network in the case of new agents. Application tasks are loosely coupled with the agent in the overlay network. Each application is implemented as an overlay; thus, a node can be a member of multiple overlays. Introducing a Gate-to-Overlay entity to support the legacy nodes at sensor layer provides two separate channels for data and signaling, which provides lightweight data exchange protocols between the interface layers. However, the complexity in building the Gate-to-Overlay node is found, since it will consume more energy and support more tasks that are related to the legacy nodes.
- MENO: It is one of first trails to integrate the SSNs with the internet, network virtualization based on virtual links over layer 2 or layer 3 in traditional networks. It still a conceptual work.
- IoT-VN: Network virtualization in this solution is based on the concept of MENO. It presents some implementation details regarding the integration between SSNs and the internet. Thus, it concerned with connection resource-constrained and non-resource constrained devices together and allows end-to-end communication for the deployment of applications and services.

- MAS: An agent-based network virtualization solution is presented in this approach. Although the main target is creation and maintenance of VSN, there are no details regarding VSN formation and its operation.
- A Resource Efficient Approach for Concurrent Applications: Although it presents two real world applications shared the same SSNs, it does not mention any technical details for creating a dynamic or static VSNs.
- Melete System: This system provides a dynamic logical grouping of sensors over the network to serve a certain application. A node can be a member of multiple groups. This system supports up to 16 groups that coexist in SSN, but the sensors only execute one application task at a time.
- Clustered-based solutions:

This solution is preferred when the deployed SSNs targets are dynamic events, like mobile WSNs and vehicular ad-hoc networks.

- Cluster Tree Based Self Organization: This approach presents a dynamic formation VSNs creation that is based on event is detects. Thus, it provides a sequential execution of the desired applications. But it is not clear how it is supported by sensors.
- Closed-Loop System for Subsurface Contaminant Plume Monitoring: This approach is an extension to the previous one that improves the event tracking based on the event location, where the VSNs creation mechanism will select the nodes closest to it to save the energy.

For the discussed System Level, the following is concluded:

In the recent future, this level of virtualization will be applied at the SSNs providers. There are real SSNs examples that apply this level of virtualization. However, all of them are still for research purposes, as GENI networks, FIT IoT-LAB; each of them provide a huge SSN equipment in multiple sites for researches all over the world to implement and test different issues that are related to the field of WSNs. Researchers can create an account on the FIT IoT-LAB easier than GENI. Additionally, he can use a readymade firmware or build his own. While the WISEBED project has simple SSN equipment that is not published for researches.

Regarding SSN participation, the listed approaches are educational test-beds for research purposes. There is no external access for CPN, which is built for studying the behavior of communication between the VSN, while the WSNTS are built for studying the behavior of energy consumption and security issues in VSNs. Additionally, it provides a web portal API for researchers to give them external access to their experiments.

3.3. SSNs Using Hybrid Level Virtualization

SSNs using hybrid level virtualization presents a combination of LLV and HLV to support different areas of networking. This category has three branches, which are middleware/cluster based, middleware/virtual networks (or overlay) based, and virtual machine/dynamic grouping based solutions, as in Figure 36. A brief discussion regarding the well-known software in this area addition to a comparative analysis between them will be presented in Table 4.

Table 4. Characteristics Summary of SSNs's Hybrid Solutions.

Solution	Type	LLV	HLV	Programming Model	Programming Language	Task Execution	OS & Application	On the Fly Reprogramming
Sensomax 2013	MW/Cluster	MW Agent-Based	Cluster-Based	Thread-Based	Java	Sequential	Loosely coupled	√
Multi-set Architecture 2010		MW Agent-Based	Cluster-Based	Event-Driven	nesC	Sequential	Loosely coupled	-
SenShare 2012	MW/VN	Multi-tasking OS + MW (HAL)	Overlay based on CTP	Event-Driven	nesC	Simultaneous	Loosely coupled	x
A framework for service provisioning in VSNs 2012		MW	Overlay based on DVNS	-	-	-	-	-
Melete 2006	VM/Dynamic Grouping	VM	Dynamic Grouping	Event-Driven	TinyScript	Simultaneous	Tightly coupled

Figure 36. Hybrid Solutions of SSNs.

3.3.1. Hybrid Virtualization Based on Middleware/Cluster Based Solutions

This branch handles the node level using middleware while handling the high level by clustering sensors into multiple groups. As in:

- Sensomax** [79]: is a middleware solution that supports multiple concurrent applications in SSNs. It is based on two levels of virtualization; the first level is network level at which the network is participated into clusters, and each cluster contains a cluster head and multiple members, each cluster may be dedicated to a single or multiple applications, and treated as a single entity by the application developer, the sensor nodes may play multiple roles where it could be a cluster head for an application and serve another application as a member node. By this way, the applications can be span over multiple clusters, which is by running the agent associated to the application on these clusters, as in Figure 37.

Figure 37. Sensomax Clusters.

The second level of virtualization is the node level at which the applications are coded as agents that are allocated in an execution space and loaded when the associated applications start to run, as in Figure 38. An agent will be loaded to the execution space when running a relevant application triggers it; thus, if multiple applications are running concurrently, Sensomax provides a resource-allocation algorithm to allocate resources among the running agents in the execution space. There are three types of resources; the global resources that represent all shared resources over the network, the local resources that represent the shared resources inside the cluster and system resources represent the statuses of the reserved resources. There are three

types of agents that are used to handle the variety of communication levels: the global agents handle communications among clusters, local agents are responsible for handling the inter-cluster communication, and the system agents handle the reprogramming of sensor nodes on the fly. The performance of Sensomax was evaluated by a network prototype of 12 sensor nodes and by simulation. The results showed that the execution time for an agent is 200 ms if concurrent applications are served by that node, and this time increases as the number of applications increase. The execution time of dynamic update is lower than 100 ms for the same number of applications.

- **Multi-set Architecture for Multi-applications Running on WSNs [80]:** it presents a multi set architectural model for concurrent application running on SSN. The HLV is based on grouping sensor nodes into clusters with cluster heads, which are responsible for propagating the configuration agent that is inherited from the base station to the member nodes, here each cluster serves only one application, therefore the number of hosted application in this type of SSN is equal to the number of created clusters, and thus their execution priorities must be known before the operation starts and this limits the flexibility. While, for LLV that is based on a middleware running on the top of the operating system “TinyOS”, the middleware architecture uses two mobile agents, called Configuration Agent (C-Agent) and Switching Agent (S-Agent), as shown in Figure 39. These agents are used to control the functionality of LLV and NWLV, respectively.

The base station starts the system operation by injecting the C-Agent into the WSN that propagates to the cluster heads to form the clusters. One application is run at a time, thus only one cluster is run at a time while the others are in a sleep mode, and the S-Agent is used to control the wake-up/sleep switching between clusters. The performance evaluation was not presented for this architecture.

Figure 38. Sensomax Architecture.

Figure 39. Middleware layer architecture.

3.3.2. Hybrid Virtualization Based on Middleware and Virtual Networks/Overlay Solutions

This branch also handles the node level using middleware, but handles the high level by virtual network/overlay.

- **SenShare** [20]: is a platform that allows SSNs to be shared among multiple applications. This platform separates between the infrastructure owner and the application developers to offer more flexibility. LLV is based on hardware abstraction layer (HAL) to allow multiple applications to simultaneously operate on each sensor node over a multitasking OS by defining a set of interfaces that facilitate the development of applications without caring about the background hardware, as in Figure 40.

Figure 40. SenShare node Architecture.

Each application in SenShare has its own ID that is attached to its packets to identify them from other running applications where ID is used to manage data and control messages from and to it. Applications also share the HAL that is used as a hardware abstraction layer to allow multiple access to it. Thus, each application has its own virtual hardware that is linked to the TinyOS. Therefore, when an application requests hardware access, the relevant virtual hardware passes that request to the runtime layer between it and the OS to start up the application, it also manages traffic isolation between multiple applications while using the ID attached to the packet at the source, and removes it at the destination. While, at the HAL the overlay middleware groups the scattered nodes associated to a certain application in a single VSN, that's done by using the Collection Tree Protocol (CTP) [81], which is also used for routing data and controlling messages among clusters. The performance of SenShare was evaluated by studying the sampling rate of the sensor node that decreased by 28% when multiple applications shared it as compared to a single running application for the same phenomena. The utilization of the CPU and memory usage linearly increased with the number of added applications. It was also observed that the overlay performs better as the number of scattered nodes increased.

- **A framework for service provisioning in VSNs** [18]: This framework aims to design a middleware between the applications and sensor nodes; it proposes a sensor node architecture that supports the advanced virtualization feature. It achieves the node level virtualization by a middleware that sits on the top of network layer, thus a new configured MAC layer will be instantiated for each application served by the node, where each node contains a node virtualization manager that handles the available resources with the received requests. A VSN core framework that consists of a Dynamic Virtual Network Server (DVNS), a number of instanced VSN Managers, and that may also host some Legacy Gateway Interworking Functions (LGW IF) achieves the network level virtualization. The DVNS is responsible for discovering/registering and publishing VSN services to the end users and initiating service providing. The VSN Manager is responsible for service negotiation, session establishment, and monitoring, while the LGW IF enables the interaction with legacy Wireless Sensor Islands (WSIs), which are the autonomous administrative domain that carry one service or more.

3.3.3. Hybrid Virtualization Based on Virtual Machine/Dynamic Grouping Solution

This branch handles the node level using virtual machine, while the high level is handled by dynamic grouping.

- Melete system [60]: As described above, this system was one of the first trails towards SSNs, where for the node level virtualization a simultaneous execution of hosted applications can be achieved, nodes can serve up to five application concurrently, and each one stores its own code in a dedicated execution space without sharing any variables with each other to ensure that the failure of any one of them will not affect the others. While the network level virtualization is achieved by dynamic grouping of nodes to serve a certain application, where the nodes by default join a global logical group, called the associated group, and then join a logical group dedicated to serve a certain application, the nodes can also be members of multiple logical groups up to five groups.

3.3.4. An Insight on the Application of Hybrid Level Virtualization

The hybrid solutions benefit from both LLV and HLV features, thus it takes much attention in recent running research.

For the discussed Hybrid Level Virtualization, the following is concluded:

- MW/Cluster-based
 - Sensomax: At LLV Java Sun Spot devices are used to exploit their ability to run concurrent application tasks. Each application is implemented as an agent. The base station can assign multiple agents to one node, using one-hop broadcast. For NLV it uses clustering concept, where it divides the network into clusters, each with its own cluster head.
 - Multi-set Architecture for Multi-applications Running on WSN: It uses TinyOS to provide LLV, thus, it inherits its drawbacks. While, for NLV, the network is divided into subsets, each subset groups nodes as a cluster and each has its own cluster head. Each subset is dedicated to one application. When SSNs is deployed, only one application begins its execution, according to the pre-defined sequence. While nodes in other subsets sleep to save energy until their turn comes.
- MW and Virtual link/overlay
 - SenShare: It uses TinyOS to provide LLV, thus, it inherits its drawbacks. Nevertheless, it uses CTP for creating dynamic clusters for HLV.
 - A framework for service providing in VSNs: it provides LLV using MW in each node, while the HLV is achieved by gateway nodes that provide VSNs creation and management.
- MW/Hierarchical clustering
 - Melete system: At LLV, Melete improves on Maté, but, since application tasks have their own data and execution space, only a limited number of application tasks can run concurrently. The programming model is based on the event-driven approach of TinyOS. The application programs are written in TinyScript. While HLV is achieved by a dynamic grouping scheme. Finally, the programmer needs to be aware of many situations that may arise in the network and program the responses, and this approach is not flexible at all.

4. Opportunities Provided by SSNs

A SSN provides a lot of opportunities to the WSN, it provides a new research area that will become a reality, like the internet when its infrastructure has enough support from sensor network providers in the commercial environment. This section will focus on displaying opportunities that are provided by implementing SSNs, which could be, as follows:

- 1- Infrastructure Sharing: virtualization technology provides a great donation for WSNs by allowing for its infrastructure to be shared between multiple concurrent applications and cause its conversion from an application/service specific system to a general purpose physical resources to multiple concurrent (shared) application system in its new look as a SSN. Many benefits are obtained behind this opportunity, like reducing WSN redundancy, as there is no need to build a dedicated WSN each time required to host an application, hardware technical details are abstracted from application developers and users, which increases the security and reduces developments overheads.
- 2- Infrastructure Marketing: one of the greatest benefits of SSNs is the separation between the infrastructure and the offered service using virtualization, which allows for providers to sell the infrastructure to a third party without interrupting the hosted services, and thus opening large business opportunities.
- 3- Infrastructure Diversity: as a result of infrastructure separation from the service based on virtualization, the infrastructure provider becomes able to co-operate with third parties, and offers multiple isolated services over a cost reduced infrastructure.
- 4- Flexible and Scalable Infrastructure: SSNs have the flexibility to add and remove a VSN according to the hosted application requirements, and it also has the ability to generate the co-operation between heterogeneous sensor networks to provide infrastructure scalability.
- 5- Opened Vendor Platform: SSNs is a heterogeneous based system, thus it gives all hardware vendors the same opportunity to work under its umbrella.
- 6- Foundation Stone for multiple emerging Technologies: SSNs is a basic infrastructure for many emerging technologies, like IoT, which allows for objects to be sensed and controlled remotely across existing network infrastructures, creating opportunities for more direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy, and economic benefit. It is also a basic stone in the emerging technology, called Cloud of Sensors (CoS), at which the related physical sensors are grouped together to form a virtual sensor, and then these virtual sensors are grouped together to form a global sensor cloud. This cloud can be used by different concurrent applications; visualization and remote management platforms that leverage to provide excellent data scalability; and, user programmable analysis.
- 7- Define New Business Models: that define the roles of SSNs, such as infrastructure providers and service providers.

Information Sharing: Data collected by sensors can be shared among hosted applications, which give lower energy consumption and overcome redundant data transmission.

5. Applications of SSNs

Traditional WSNs were motivated by military applications, such as as battlefield surveillance, but nowadays its new paradigm (SSNs) spread in different daily life fields, like civilian applications (smart home, surveillance, habitat, structure and environmental monitoring, healthcare, traffic control, etc.), thus the needs for the SSNs increased, and thus this section will present a brief review for its applications.

- 1- Battlefield Monitoring [11,12]
Monitoring a battlefield is a complex mission, because there are different types of targets to be considered as civilians, enemies, soldiers, different military vehicles, important infrastructure, etc., and also in the battlefield there will be different types of deployed sensors that are deployed as sound sensors, camera sensors, temperature sensors, motion detection sensors, etc. Which were implemented from different vendors, thus deploying a SSN is a cost effective and optimal solution to serve different applications, over a united heterogeneous infrastructure.

2- Smart Buildings [6,10,82,83]

Intelligent control for smart building conditions requires a large number of varied applications that are based on the variety of deployed sensors that collaborate to monitor and control the environment to make it more comfortable and more secure by controlling (temperature, humidity, doors and windows lock status, video surveillance, air conditioning, lighting, acoustic signals for noise control, etc.), through local or remote access of the internet and monitoring (building infrastructure health, gas, water, CO, smoke detection, etc.), so, if you are away of your home, you can make it ready for your arrival by remotely controlling internal conditions, also you can control your office remotely and monitor your employees, etc. Therefore, this variety of HW and SW is a fruitful environment for implementing a SSN.

3- Food Transportation [84]

Due to the importance of food transportation process from production sites to consumption sites, while saving the quality of the products during this process, the SSN became the best solution for monitoring food statuses during that process, monitoring and managing different food types with different conditions for each type, and then sending this data to a certain user (food owner).

4- Health and Medical applications [52]

As a result of the continuous increase in population over the world, the need for healthcare applications also increased to provide patients' comfortability and elderly. Thus, the WSN technology provides a separate branch for that purpose, called Wireless Body Sensor Networks (WSBNs), but healthcare applications in that branch are isolated from each other, which causes more sensor equipment requirement and more administrative loads from the medical team, thus the SSN provides an effective enhancement when applied to that branch, because it allows sensors and application information sharing among concurrent applications, this will reduce the number of mounted sensors on the patient, making him more comfortable and reducing the administrative loads over development and medical team. If the patient status needs adding a new application, it can be added without interrupting the working ones, this increases flexibility and reduces the overall cost.

5- Smart Grid [85]

The complexity of electricity services comes from the need to deliver it to the customer at real time; it passes through three stages, starting with energy generation, followed by power transmission and then power distribution. Thus, monitoring and controlling these three stages is a complex process, which requires multiple motoring and controlling applications that must be able to share the gathered information among them, SSN can deal with this scenario, and make the system more reliable and flexible.

6- Agricultural Monitoring [8]

SSN gives a real chance to improve agriculture quality and productivity using real time monitoring for soil elements, such as temperature, humidity, soil condition, sunlight, etc. This provides the farmer the ability to take good decisions regarding optimal plants placement, water level requirements, energy and pesticide usage, etc. This will produce a suitable environment that is based on the diverse conditions that are required for different types of crops.

7- Industrial Monitoring [86]

Attaching a SSN to the industrial process will improve it by providing a diverse of applications that facilitate the system management and control. Offering dynamic data share and analysis for everything in the industrial environment, as energy consumption, machine status, product quality, and alerts for critical equipment parameters (system temperature, electricity, etc.) give an early detection of unstable conditions that cause risk reduction and removal of the related cost as a result. Therefore, it directly affects the manager decisions and makes industrial operations more stable, predictable, better energy consumption, safer, and cost effective.

8- Natural Monitoring [87,88]

Observation of nature to predict disasters as ice avalanches, rock falls, volcanic eruptions, the migration of animals, etc. Is a sensitive issue, as it involves securing human beings, animals, and plants. Thus, a SSN provides diverse applications for that purpose, it offers reusing the old deployed infrastructure in these places and this reduces the overall cost and gives a chance for new applications to be deployed in that places, while sharing the gathered information among them.

6. SSNs Projects

Many projects envisioned the utilization of SSNs by multiple concurrent applications, which tells us the importance of its role towards the IoT revolution. Each project has its own way to treat the challenges that are faced by this type of emerging shared network. Table 5 summarizes the characteristics for presented projects.

1- **FRESNEL [89]**: Federated Secure Sensor Network Laboratory is part of an academic research provided by Oxford internet institute in cooperation with Computer Laboratory (University of Cambridge) and Oxford University Computing Laboratory (ComLab). It focused on building a large scale federated sensor network framework, which supports concurrent applications running on the same infrastructure while also ensuring the reliability of intra-application communication as well as the scalability of distributed management infrastructure. It also maintains orthogonality, privacy, and security. The used sensors sense many environmental phenomena, like temperature, pollution, movement, etc., and the network will be running multiple applications of multiple authorities in the city of deployment. There are many research objectives to achieve the project goals described briefly, as follows:

- Developing a language for service level description and an agreement for this emerging type of shared sensor networks, this language will be the base for resource sharing, application management, and monitoring framework.
- Developing new techniques for dynamic resource allocation are decentralized solutions, these techniques will be developed according to the applications and they will vary according to its needs based on available network resources.
- Identifying the mechanisms for dynamic partitioning of the physical network into application-specific virtual sensor networks. During the phase of dynamic partitioning/repartitioning, these mechanisms must protect different application's nodes against each other, for privacy preservation and the observation of resource allocation boundary issues.
- Developing device distributed and scalable communication protocols, to allow for reliable and efficient communication among sensor applications.
- Developing distributed algorithms for processing concurrent queries with very different quality-of-service requirements for accuracy and delay. These algorithms should be able to give priority to degrading the quality of the query; by this way, they can handle the needs of critical applications.
- Deploying a prototype of federated sensor networks across the campus of the University of Cambridge, which partly integrate with existing sensor networks. This prototype will cover the needs of heterogeneous monitoring, which range from room usage and college security to air quality and traffic monitoring.

2- **Virtual Sensor Networks (VSNs) [63]**: is a project for a self-organizing scheme on the top of a cluster tree in VSNs based on a subsurface chemical plume monitoring system, this project is provided by Computer Networking Research Laboratory by Colorado State University. The project team concentrated on cluster tree random routing, virtual coordinates, and VSN support functions. As VSNs are an emerging form of collaborative WSNs, it can support collaborative, resource

efficient, and multi-purpose WSNs. The VSNs are created by a subset of sensor nodes/users that dynamically vary based on the migration between the available resources and application requirements. This project is useful in three types of applications: the first is Geographically overlapped applications, like monitoring rock slides and animal crossing within a mountainous terrain; the second is logically separated multi-purpose sensor networks, like smart neighborhood systems with multifunctional sensor nodes; and, the third are certain dedicated applications, such as those that enhance the efficiency of a system by tracking a dynamic phenomena, such as subsurface chemical plumes that migrate, split, or merge. Thus, this project aims for resource sharing, which achieves application objectives using a good resource utilization way.

- 3- **Virtualized Sensing Environment (ViSE) [90]**: this project is funded by NSF as part of the GENI [51] initiative, hosted at UMass-Amherst. The project scope is extending WSNs outdoor using a wide-area sensor/actuator network testbed to support slivering and utilizing. A GENI candidate control framework that selects the Open Resource Control Architecture (ORCA) [59] as a platform for federating the environment of the GENI testbed infrastructure services. It contains: (1) Virtualization of a sensor/actuator system, exporting rich actuation capabilities that produce high-bandwidth sensor data, such as Pan-Tilt-Zoom video cameras and steerable weather radars; (2) Integrating sensor virtualization technology with GENI-compliant Software Artifacts, including the use of Shirako software (part of the ORCA project) as the base of the control framework to create a GENI; (3) Operating a testbed linked to GENI that enables researchers and end-users to control a fabric of rich sensors in end-to-end GENI experiments. In the first year, publish your testbed to available GENI users, by the end of second year integrate it into an environment of GENI federated testbeds; and, (4) Preparing the required documentation for testbed users, administrators, and developers. The project finished an initial research-grade implementation of sensor virtualization in Xen that applies the approach to Pan-Tilt-Zoom video cameras. There are two types of users that are targeted by this project's testbed; the first user is the one who wishes to experiment with long distance 802.11b wireless communication, but it is difficult to setup, because they need to access the infrastructure, especially the towers, to provide line-of-sight. The second users are the radar researchers who can leverage the project radar deployment. Notice that the testbed interacts with a remote Clearinghouse run by RENC/Duke to facilitate resource allocation.
- 4- **DVM [91]**: is a Multi-level Software Reconfiguration for Sensor Networks through Dynamically Extensible Virtual Machine projects, which aims at presenting a system that supports the flexible software reconfiguration of sensor networks. The system architecture consists of: (1) an operating system kernel, which allows for dynamica incremental upgrades to the system; and, (2) Binary modules that can be dynamically inserted, updated, or removed unobtrusively. There is a Dynamically extensible Virtual Machine (DVM) on the top of operating system, which can respond to some events and execute application scripts. The DVM has three unique features, the first is that Kernel binary modules can expose interfaces to the VM, these modules are scriptable and are able to dynamically register custom extensions to the VM at run-time. The second is that kernel scriptable binary modules can interact with application scripts and expose their configurable parameters to the VM through a published interface that facilitates the module parameter reconfiguration according to the operating environment. The third is giving users the ability to install and update the scripts, and also to modify at the run-time set of events. Thus, the high-level scripts that are executed by the VM, can efficiently access services that are exported by a module and tune module parameters.
- 5- **PRESTO [92,93]**: Predictive storage architecture for emerging large-scale hierarchical sensor networks. Most recently due to the evolution of WSNs' field, new paradigms as sensor-cloud and sensor-grid appeared, a large amount of data is collected every minute from different applications, like RFDI, weather, habitat monitoring, building monitors, surveillance, remote sensing data, such as radar, etc. ... the generated data needs to be processed, filtered, interpreted, cached,

and archived in order to provide a useful infrastructure for users, thus PRESTO is presented to provide the interactivity of the data streaming approach with the energy efficiency of the direct sensor querying. PRESTO design focuses on five main characteristics briefly described, as follows: (1) **Tiered Design**: because the large scale sensor networks of the future are expected to be multiple tiers, with several tens of untethered sensors per tethered sensor proxy and several tens of sensor proxies per application. (2) **Predictive Storage**: PRESTO makes extensive use of predictive techniques that are a natural fit to the correlated behavior of the physical world. Technology trends in storage are exploited to build an architecture that emphasizes archival at remote sensors and extensive modeling and predictive caching at proxies. (3) **Interactive Use**: PRESTO is designed for rapid responses to users posing ad-hoc queries on the distributed sensor data without incurring the energy costs of the data streaming approach or the losses, latency, and reliability concerns of directly querying remote sensors. (4) **Archival Queries**: PRESTO supports archival queries on data that may be deemed to be an interesting *post-facto*. The ability to query historical data is important in many sensor applications to conduct postmortems of unexpected and unusual events to better understand them in the future. (5) **Single Logical View of Data**: PRESTO aims to provide a single logical view of data distributed across many sensor proxies and numerous remote sensors. Such a view can abstract the user from variability's at many levels; unreliable remote sensor network, spatial and temporal consistency issues in sensor data, as well as bandwidth and connectivity issues in the case of wireless proxies.

- 6- **WebDust** [94]: is a generic and modular application environment for developing applications for WSNs, it gives the application developer the facility to develop a customized environment based on the application requirements, and this will provide a wide range of services for WSNs. WebDust stores the gathered information from WSN in a database to offer extendable statistics and provide users different web-based interfaces to present the gathered information according to the application requirements. The WebDust environment gives nodes and devices the ability to automatically register technical characteristics in heterogeneous hierarchical sensor networks. WebDust has many strengths (1) due to the automatic node registration, it gives the administrator an easy setup and good control for the sensor network. (2) Multiple, heterogeneous WSNs can be treated as a unified VSN. (3) The web-based application interface gives users the ability to access, monitor, query data, etc. in a simultaneous way.
- 7- **CitySense** [95]: it is a large-scale urban sensor network testbed of more than 100 sensor nodes that are powered by streetlights across the city of Cambridge. Each sensor node is connected to its neighbors via a radio link, and it uses ad hoc routing protocols to communicate with each other using network layer (IP), also each node is directly programmed by the end user, where the project team aimed at allowing multiple users and applications to share the CitySense testbed because this project was designed to facilitate a broad range of research projects in large-scale sensor networks. Thus, this testbed is open to researchers, by using a web-based interface that gives them the availability to collect data, experiment with networking protocols, and reprogram nodes based on the experiment requirements. CitySense targets to build an infrastructure to support a broad range of research projects, also to implement a web-based interface for scheduling, programming, and monitoring its platform in an easy way, and the developed software under CitySense will be released with an open source license, the hardware designs will be published with no restrictions on use and the simulation studies can be experimentally validated using CitySense.
- 8- **VITRO** [96]: Virtualized dIstributed plaTfoRms of smart object, this project aims to build a link between heterogeneous WSNs platform and cooperative smart objects. That is by developing the required architectures, algorithms and engineering methods, which will enable the realization of scalable, flexible, adaptive, energy-efficient, and trust-aware VSN platforms. To achieve this goal, VITRO platform provides:

- (1) resource (HW&SW) discovery, allocation, and management of a great collection of heterogeneous smart objects will become more easier;
- (2) the collaboration between inter-objects should be dependable, virtualized, secure and scalable, to ensure energy-efficiency, trust-awareness, and seamless connectivity for good communication in large-scale heterogeneous WSNs; and,
- (3) expanding the network degree by enabling federated collaboration with external homogeneous objects.

VITRO will produce tools that will be packed in a VITRO toolbox that contains middleware, advanced component for core communication and a user-friendly management tool, and it will facilitate the VSN application deployment, configuration, and instant support.

- 9- **Smart Santander [97]**: this project aims at the creation of an experimental test facility for the research and experimentation of architectures, key enabling technologies, services, and applications for the IoT in an urban landscape. The envisioned facility is conceived as an essential instrument to achieve European leadership on key enabling technologies for IoT, and to provide the European research community with a one and only platform of its characteristics, which is suitable for large scale experiments and an evaluation of the IoT concept under real-life conditions. Thus, smart stander facility is a large scale smart city that was deployed in the Spanish city of Santander, and its testbed supports two types of experiments: IoT native experimentation (WSN experiments) and service provision experiments (applications using real-time real-world generated sensor data). Smart Santander offers different services, like integral traffic management, environmental monitoring parks and garden irrigation, participatory sensing, and augmented reality. All of the gathered information is stored in a large database, which will be allowed for concurrent users once they are authenticated through the web-based interface. IoT infrastructure consists of an IoT node, which gathers required sensing information from the environment as (temperature, noise, light, etc.), repeaters: which work as forwarding nodes for the gathered information, thus they are placed in a higher level, such as over street light, and the gateway node that is responsible for gathering the sensed data from sensor nodes and sending it through the internet to the project servers.
- 10- **iCore [98]**: this project is a coherent European approach that targets providing a cognitive management framework and associated functionalities to the IoT, thus it has two main issues, first one is the abstraction of diverse objects heterogeneity and the second is obtaining good reliability to ensure the provision of different applications for concurrent users while obtaining business integrity. This project proposed a cognitive framework that consists of virtual objects (VOs), composite virtual objects (CVOs), and functional blocks, and all are reusable for various applications. VOs are responsible for abstracting the heterogeneity of diverse of real world objects into virtual objects. CVOs are responsible for mixing semantically interoperable VOs and delivering services to associated users under proposed security protocols, which span over all framework levels to ensure the ownership and privacy of data, controlling objects access by using scalable mechanisms for the registration, look-up, discovery, and management of entities and events.

Table 5. Characteristics Summary of SSNs's Projects.

Project	Type	Target	Virtualization Level	Hardware
FRESNEL (2010–2012)	Academic research	Building a large scale federated sensor network framework which support the concurrent applications	LLV & HLV	35 iMote2 nodes (embedded Linux) distributed in an academic building
VSN	Academic research	Cluster tree random routing, virtual coordinates	HLV	-
WISE (2010–2013)	Academic research	Extend WSNs outdoor using GENI framework	LLV	Three High-end nodes acting as gateways and running Linux where deployed near a forested area
DVM 2006	Academic research	Presenting a Multi-level Software system that supports flexible software reconfiguration of sensor networks.	LLV	Mica2 motes with SOS kernel including the DVM core.
PRESTO 2005	Academic research	presented to provide the interactivity of the data streaming approach with the energy efficiency of the direct sensor querying in large-scale sensor networks	LLV & HLV	Telos Motes scale up to 100 Motes per proxy
WebDust	-	Build a modular application environment for developing applications for WSNs	LLV	-
CitySense 2008	Academic research	Build a large-scale, urban sensor network testbed	HLV	More than 100 sensor nodes with embedded Linux powered by streetlight across the city of Cambridge.

7. Conclusions

The idea, fundamentals, motivation, proposed architecture models (global SSN, node, and network architectures), building requirements, and challenges of SSNs has been overviewed throughout this article. The benefits of SSNs over the traditional WSNs. Subsequently, a novel taxonomy is presented based on the type of used virtualization technique. The taxonomy presents three main categories: HLV, LLV, and hybrid. The category of LLV presents the multitasking OSs, VM, and MW solutions. More efforts are required to virtualize the individual components of sensor nodes, such as MAC and routing layers, as the trend moves towards more powerful IP-SSNs. While the second category is the HLV was divided into two branches, which are network and system levels: the first branch presents both Virtual NW/Over-relay Sensor NW and Cluster-based solutions, and the second branch presents the Federation and Participation solutions. The overlay networks can provide an efficient solution, as they are robust and they can work efficiently without changes in the underlying network. Some solutions exist, like those in Sensomax, SenShare, Melete system, etc., but they are still embryonic in nature and do not consider the requirements of multiple applications concurrently utilizing SSNs. Finally the hybrid category presents the Middleware/Hierarchical Clustering, Middleware Virtual NW/Overlay, and Virtual Machine Dynamic Grouping solutions. Each solution in all of the categories was briefly summarized with its performance evaluation and a comparative analysis with other peer solutions; also, a characteristics summary table for each category is presented. Some of the projects using the technology of SSNs are briefly explained. Additionally, some applications in different life aspects that are based on SSNs are presented. Under the umbrella of virtualization, a promising future is behind of SSNs, where this concept opens the ability of being capable of supporting the expected enormous growth of the IoT sensor-based uplink data traffic generated by customer devices in smart cities, smart farming, industrial applications, healthcare monitoring systems, driverless cars navigation, drones/robotics applications, and a wide diverse of user applications.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References and Notes

1. Nurellari, E.; Srivastava, S. A Practical Implementation of an Agriculture Field Monitoring using Wireless Sensor Networks and IoT Enabled. In Proceedings of the 4th IEEE International Symposium on Smart Electronic Systems, Hyderabad, India, 17–19 December 2018.
2. Lin, J.-W.; Chelliah, P.R.; Hsu, M.-C.; Hou, J.-X. Efficient Fault-Tolerant Routing in IoT Wireless Sensor Networks Based on Bipartite-Flow Graph Modeling. *IEEE Access* **2019**, *7*, 14022–14034. [[CrossRef](#)]
3. Balhwan, S.; Gupta, D.; Reddy, S.R.N. Smart Parking—A Wireless Sensor Networks Application Using IoT. In Proceedings of the 2nd International Conference on Communication, Computing and Networking, Singapore, 29–30 March 2019; pp. 217–230.
4. El-Basioni, B.M.M.; El-Kader, S.M.A.; Eissa, H.S.; Zahra, M.M. Clustering in Wireless Sensor Network: A Study on Three Well-known Clustering Protocols. In *Handbook of Research on Progressive Trends in Wireless Communications and Networking*; IGI Global: Hershey, PA, USA, 2014; pp. 340–364.
5. Abo-Elhassab, A.E.; El-Kader, S.M.A.; Elramly, S. Wireless Sensor Networks Localization. In *Routing Protocols and Architectural Solutions for Optimal Wireless Networks and Security*; IGI Global: Hershey, PA, USA, 2017; pp. 204–240.
6. El-Basioni, B.M.M.; El-Kader, S.M.A.; Eissa, H.S. Independent living for persons with disabilities and elderly people using smart home technology. *Int. J. Appl. Innov. Eng. Manag.* **2014**, *3*, 11–28.
7. Nabeel, M.M.; I-Dien, F.M.E.; I-Kader, A.S.E. Intelligent vehicle recognition based on wireless sensor network. *Int. J. Comput. Sci.* **2013**, *10*, 164–174.
8. El-Kader, S.M.A.; El-Basioni, B.M.M. Precision farming solution in Egypt using the wireless sensor network technology. *Egypt. Inform. J.* **2013**, *14*, 221–233. [[CrossRef](#)]
9. El-Basioni, M.M.; El-kader, S.M.A.; Fakhredin, M.A. Smart Home Design using Wireless Sensor. *Netw. Biom. Technol.* **2013**, *1*, 2.
10. El-Basioni, B.M.M.; El-kader, S.M.A.; Abdelmonim, M. Smart home design using wireless sensor network and biometric technologies. *Inf. Technol.* **2013**, *1*, 2.
11. Onur, E.; Ersoy, C.; Deliç, H.; Akarun, L. Surveillance with wireless sensor networks in obstruction: Breach paths as watershed contours. *Comput. Netw.* **2010**, *54*, 428–441. [[CrossRef](#)]
12. Bokareva, T.; Hu, W.; Kanhere, S.; Ristic, B.; Gordon, N.; Bessell, T.; Rutten, M.; Jha, S. Wireless sensor networks for battlefield surveillance. In Proceedings of the Land Warfare Conference 2006, Brisbane, Australia, 24–27 October 2006; pp. 1–8.
13. Islam, M.M.; Hassan, M.M.; Lee, G.-W.; Huh, E.-N. A survey on virtualization of wireless sensor networks. *Sensors* **2012**, *12*, 2175–2207. [[CrossRef](#)] [[PubMed](#)]
14. Khan, I.; Belqasmi, F.; Glitho, R.; Crespi, N.; Morrow, M.; Polakos, P. Wireless sensor network virtualization: A survey. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 553–576. [[CrossRef](#)]
15. Farias, C.M.D.; Li, W.; Delicato, F.C.; Pirmez, L.; Zomaya, A.Y.; Pires, P.F.; de Souza, J.N. A systematic review of shared sensor networks. *ACM Comput. Surv. CSUR* **2016**, *48*, 51. [[CrossRef](#)]
16. Liang, C.; Yu, F.R. Wireless network virtualization: A survey, some research issues and challenges. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 358–380. [[CrossRef](#)]
17. Islam, M.M.; Huh, E.-N. Virtualization in Wireless Sensor Network: Challenges and Opportunities. *J. Netw.* **2012**, *7*, 412. [[CrossRef](#)]
18. Sarakis, L.; Zahariadis, T.; Leligou, H.-C.; Dohler, M. A framework for service provisioning in virtual sensor networks. *EURASIP J. Wirel. Commun. Netw.* **2012**, *2012*, 135. [[CrossRef](#)]
19. Fok, C.-L.; Roman, G.-C.; Lu, C. Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. *ACM Trans. Auton. Adapt. Syst.* **2009**, *4*, 16. [[CrossRef](#)]
20. Leontiadis, I.; Efstratiou, C.; Mascolo, C.; Crowcroft, J. SenShare: Transforming sensor networks into multi-application sensing infrastructures. In Proceedings of the European Conference on Wireless Sensor Networks, Trento, Italy, 15–17 February 2012; pp. 65–81.

21. Khan, I.; Belqasmi, F.; Glitho, R.; Crespi, N. A multi-layer architecture for wireless sensor network virtualization. In Proceedings of the 2013 6th Joint IFIP Wireless and Mobile Networking Conference (WMNC), Dubai, United Arab Emirates, 23–25 April 2013; pp. 1–4.
22. Khan, I.; Belqasmi, F.; Glitho, R.; Crespi, N.; Morrow, M.; Polakos, P. Wireless sensor network virtualization: Early architecture and research perspectives. *IEEE Netw.* **2015**, *29*, 104–112. [[CrossRef](#)]
23. Khan, I.; Errounda, F.Z.; Yangui, S.; Glitho, R.; Crespi, N. Getting Virtualized Wireless Sensor Networks' IaaS Ready for PaaS. In Proceedings of the 2015 International Conference on Distributed Computing in Sensor Systems (DCOSS), Fortaleza, Brazil, 10–12 June 2015; pp. 224–229.
24. Microsoft Word—ICACT_Conf_paper-LEE-F.docx—20160258_finalpaper.pdf. Available online: http://icact.org/upload/2016/0258/20160258_finalpaper.pdf (accessed on 4 November 2017).
25. Madria, S.; Kumar, V.; Dalvi, R. Sensor cloud: A cloud of virtual sensors. *IEEE Softw.* **2014**, *31*, 70–77. [[CrossRef](#)]
26. Distefano, S.; Merlino, G.; Puliafito, A.; Vecchio, A. A hypervisor for infrastructure-enabled sensing clouds. In Proceedings of the 2013 IEEE International Conference on Communications Workshops (ICC), Budapest, Hungary, 9–13 June 2013; pp. 1362–1366.
27. Akyildiz, I.F.; Wang, P.; Lin, S.-C. SoftWater: Software-defined networking for next-generation underwater communication systems. *Ad Hoc Netw.* **2016**, *46*, 1–11. [[CrossRef](#)]
28. Blenk, A.; Basta, A.; Reisslein, M.; Kellerer, W. Survey on network virtualization hypervisors for software defined networking. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 655–685. [[CrossRef](#)]
29. Cecílio, J.; Furtado, P. Existing Middleware Solutions for Wireless Sensor Networks. In *Wireless Sensors in Heterogeneous Networked Systems-Computer Communications and Networks*; Springer: Cham, Switzerland, 2016.
30. Levis, P.; Madden, S.; Polastre, J.; Szewczyk, R.; Whitehouse, K.; Woo, A.; Gay, D.; Hill, J.; Welsh, M.; Brewer, E.; et al. Tinyos: An operating system for sensor networks. In *Ambient Intelligence*; Springer: Berlin, Germany, 2005; pp. 115–148.
31. Chu, R.; Gu, L.; Liu, Y.; Li, M.; Lu, X. SenSmart: Adaptive stack management for multitasking sensor networks. *IEEE Trans. Comput.* **2013**, *62*, 137–150. [[CrossRef](#)]
32. Baccelli, E.; Hahm, O.; Gunes, M.; Wahlisch, M.; Schmidt, T.C. RIOT OS: Towards an OS for the Internet of Things. In Proceedings of the 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Turin, Italy, 14–19 April 2013; pp. 79–80.
33. Baccelli, E.; Hahm, O.; Günes, M.; Wählisch, M.; Schmidt, T.C. OS for the IoT-Goals, Challenges, and Solutions. In Proceedings of the Workshop Interdisciplinaire sur la Sécurité Globale (WISG2013), Baden-Baden, Germany, 12–15 May 2013.
34. Bhatti, S.; Carlson, J.; Dai, H.; Deng, J.; Rose, J.; Sheth, A.; Shucker, B.; Gruenwald, C.; Torgerson, A.; Han, R. MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms. *Mob. Netw. Appl.* **2005**, *10*, 563–579. [[CrossRef](#)]
35. Duffy, C.; Roedig, U.; Herbert, J.; Sreenan, C.J. A Performance Analysis of MANTIS and TinyO. *Cork Irel. Tech. Rep.* **2006**, 1–29, University College Cork, Ireland, Technical Report CS-2006-27-11, November 2006.
36. Saruwatari, S.; Suzuki, M.; Morikawa, H. PAVENET OS: A compact hard real-time operating system for precise sampling in wireless sensor networks. *SICE J. Control Meas. Syst. Integr.* **2012**, *5*, 24–33. [[CrossRef](#)]
37. Dong, W.; Chen, C.; Liu, X.; Liu, Y.; Bu, J.; Zheng, K. Senspire OS: A predictable, flexible, and efficient operating system for wireless sensor networks. *IEEE Trans. Comput.* **2011**, *60*, 1788–1801. [[CrossRef](#)]
38. Cao, Q.; Abdelzaher, T.; Stankovic, J.; He, T. The liteos operating system: Towards unix-like abstractions for wireless sensor networks. In Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN'08), St. Louis, MO, USA, 22–24 April 2008; pp. 233–244.
39. Dunkels, A.; Gronvall, B.; Voigt, T. Contiki-a lightweight and flexible operating system for tiny networked sensors. In Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, Tampa, FL, USA, 16–18 November 2004; pp. 455–462.
40. Reusing, T. Comparison of operating systems tinyos and contiki. *Sens. Nodes-Oper. Netw. Appl. (SN)* **2012**, *7*, 7–13.
41. Levis, P.; Culler, D. Maté: A tiny virtual machine for sensor networks. *ACM SIGPLAN Not.* **2002**, *37*, 85–95. [[CrossRef](#)]

42. Koshy, J.; Pandey, R. VMSTAR: Synthesizing scalable runtime environments for sensor networks. In Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, San Diego, CA, USA, 2–4 November 2005; pp. 243–254.
43. Simon, D.; Cifuentes, C.; Cleal, D.; Daniels, J.; White, D. Java™ on the bare metal of wireless sensor devices: The squawk Java virtual machine. In Proceedings of the 2nd International Conference on Virtual Execution Environments, Ottawa, ON, Canada, 14–16 June 2006; pp. 78–88.
44. Muchow, J.W. “Core J2ME technology and MIDP.” Prentice Hall PTR. 2001.
45. KVM: A Small Java Virtual Machine for J2ME. Available online: <https://barrgroup.com/Embedded-Systems/How-To/KVM-J2ME-Java-Virtual-Machine>. (accessed on 18 October 2017).
46. Pajic, M.; Chernoguzov, A.; Mangharam, R. Robust architectures for embedded wireless network control and actuation. *ACM Trans. Embed. Comput. Syst. TECS* **2012**, *11*, 82. [CrossRef]
47. Brouwers, N.; Langendoen, K.; Corke, P. Darjeeling, a feature-rich VM for the resource poor. In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, Berkeley, CA, USA, 4–6 November 2009; pp. 169–182.
48. Liu, T.; Martonosi, M. Impala: A middleware system for managing autonomic, parallel sensor systems. *ACM SIGPLAN Not.* **2003**, *38*, 107–118. [CrossRef]
49. Martonosi, M. Embedded systems in the wild: Zebanet software, hardware, and deployment experiences. *ACM SIGPLAN Not.* **2006**, *41*, 1. [CrossRef]
50. Curino, C.; Giani, M.; Giorgetta, M.; Giusti, A.; Murphy, A.L.; Picco, G.P. Mobile data collection in sensor networks: The TinyLime middleware. *Pervasive Mob. Comput.* **2005**, *1*, 446–469. [CrossRef]
51. Avilés-López, E.; García-Macías, J.A. TinySOA: A service-oriented architecture for wireless sensor networks. *Serv. Oriented Comput. Appl.* **2009**, *3*, 99–108. [CrossRef]
52. Seeger, C.; van Laerhoven, K.; Buchmann, A. MyHealthAssistant: An event-driven middleware for multiple medical applications on a smartphone-mediated body sensor network. *IEEE J. Biomed. Health Inform.* **2015**, *19*, 752–760. [CrossRef] [PubMed]
53. Hoebeke, J.; de Poorter, E.; Bouckaert, S.; Moerman, I.; Demeester, P. Managed ecosystems of networked objects. *Wirel. Pers. Commun.* **2011**, *58*, 125–143. [CrossRef]
54. Feldmann, A. Internet clean-slate design: What and why? *ACM SIGCOMM Comput. Commun. Rev.* **2007**, *37*, 59–64. [CrossRef]
55. Ishaq, I.; Hoebeke, J.; Moerman, I.; Demeester, P. Internet of things virtual networks: Bringing network virtualization to resource-constrained devices. In Proceedings of the 2012 IEEE International Conference on Green Computing and Communications (GreenCom), Besancon, France, 20–23 November 2012; pp. 293–300.
56. De Poorter, E.; Troubleyn, E.; Moerman, I.; Demeester, P. IDRA: A flexible system architecture for next generation wireless sensor networks. *Wirel. Netw.* **2011**, *17*, 1423–1440. [CrossRef]
57. Kohler, E.; Morris, R.; Chen, B.; Jannotti, J.; Kaashoek, M.F. The Click modular router. *ACM Trans. Comput. Syst. TOCS* **2000**, *18*, 263–297. [CrossRef]
58. Tynan, R.; O’Hare, G.M.; O’Grady, M.J.; Muldoon, C. Virtual sensor networks: An embedded agent approach. In Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications, Sydney, NSW, Australia, 10–12 December 2008; pp. 926–932.
59. Jayasumana, A.P.; Han, Q.; Illangasekare, T.H. Virtual sensor networks—a resource efficient approach for concurrent applications. In Proceedings of the Fourth International Conference on Information Technology (ITNG’07), Las Vegas, NV, USA, 2–4 April 2007; pp. 111–115.
60. Yu, Y.; Rittle, L.J.; Bhandari, V.; LeBrun, J.B. Supporting concurrent applications in wireless sensor networks. In Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, Boulder, CO, USA, 31 October–3 November 2006; pp. 139–152.
61. Levis, P.; Patel, N.; Culler, D.; Shenker, S. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation, San Francisco, CA, USA, 29–31 March 2004.
62. Bandara, H.M.N.; Jayasumana, A.P.; Illangasekare, T.H. Cluster tree based self-organization of virtual sensor networks. In Proceedings of the 2008 IEEE Globecom Workshops, New Orleans, LA, USA, 30 November–4 December 2008; pp. 1–6.

63. Han, Q.; Jayasumana, A.P.; Illangaskare, T.; Sakaki, T. A wireless sensor network based closed-loop system for subsurface contaminant plume monitoring. In Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), Miami, FL, USA, 14–18 April 2008; pp. 1–5.
64. What Is GENI? Available online: <https://www.geni.net/about-geni/what-is-geni/#> (accessed on 10 May 2019).
65. Elliott, C. Geni—global environment for network innovations. In Proceedings of the 33rd IEEE Conference on Local Computer Networks, Montreal, QC, Canada, 14–17 October 2008.
66. GENI Maps. Available online: <https://www.geni.net/about-geni/geni-maps/> (accessed on 10 May 2019).
67. Chatzigiannakis, I.; Fischer, S.; Koninis, C.; Mylonas, G.; Pfisterer, D. WISEBED: An open large-scale wireless sensor network testbed. In *Proceedings of the International Conference on Sensor Applications, Experimentation and Logistics*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 68–87.
68. Wireless Sensor Network Testbeds. Available online: <https://cordis.europa.eu/project/rcn/86619/factsheet/en> (accessed on 10 May 2019).
69. Hellbrück, H.; Pagel, M.; Köller, A.; Bimschas, D.; Pfisterer, D.; Fischer, S. Using and operating wireless sensor network testbeds with wisebed. In Proceedings of the 2011 the 10th IFIP Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net), Favignana Island, Sicily, Italy, 12–15 June 2011; pp. 171–178.
70. Adjih, C.; Baccelli, E.; Fleury, E.; Harter, G.; Mitton, N.; Noel, T.; Pissard-Gibollet, R.; Saint-Marcel, F.; Schreiner, G.; Vandaele, J.; et al. FIT IoT-LAB: A large scale open experimental IoT testbed. In Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), Milan, Italy, 14–16 December 2015; pp. 459–464.
71. Fambon, O.; Fleury, E.; Harter, G.; Pissard-Gibollet, R.; Saint-Marcel, F. Fit iot-lab tutorial: Hands-on practice with a very large scale testbed tool for the internet of things. In Proceedings of the 10èmes Journ. Francoph. Mobilité Ubiquité UbiMob2014, Sophia Antipolis, France, 5–6 June 2014.
72. IoT-LAB: A Very Large Scale Open Testbed. Available online: <https://www.iot-lab.info/> (accessed on 10 May 2019).
73. FreeRTOS. Available online: <https://www.freertos.org/> (accessed on 10 May 2019).
74. OpenWSN-Confluence. Available online: <https://openwsn.atlassian.net/wiki/spaces/OW/overview> (accessed on 10 May 2019).
75. Sheu, J.-P.; Chang, C.-J.; Sun, C.-Y.; Hu, W.-K. WSNTB: A testbed for heterogeneous wireless sensor networks. In Proceedings of the 2008 First IEEE International Conference on Ubi-Media Computing, Lanzhou, China, 31 July–1 August 2008; pp. 338–343.
76. Levis, K. RSSI is under appreciated. In Proceedings of the Third Workshop on Embedded Networked Sensors, Cambridge, MA, USA, May 2006; pp. 239–242.
77. Raju, M.; Oliveira, T.; Agrawal, D.P. A practical distance estimator through distributed RSSI/LQI processing—An experimental study. In Proceedings of the 2012 IEEE International Conference on Communications (ICC), Ottawa, ON, Canada, 10–15 June 2012; pp. 6575–6579.
78. Kowalczyk, J.; Vuran, M.C.; Pérez, L.C. A dual-network testbed for wireless sensor applications. In Proceedings of the 2011 IEEE Global Telecommunications Conference (GLOBECOM 2011), Houston, TX, USA, 5–9 December 2011; pp. 1–5.
79. Haghighi, M.; Cliff, D. Multi-agent support for multiple concurrent applications and dynamic data-gathering in wireless sensor networks. In Proceedings of the 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), Taichung, Taiwan, 3–5 July 2013; pp. 320–325.
80. Majeed, A.; Zia, T.A. Multi-set architecture for multi-applications running on wireless sensor networks. In Proceedings of the 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA), Perth, WA, Australia, 20–23 April 2010; pp. 299–304.
81. Gnawali, O.; Fonseca, R.; Jamieson, K.; Moss, D.; Levis, P. Collection tree protocol. In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, Berkeley, CA, USA, 4–6 November 2009; pp. 1–14.
82. Zygiaris, S. Smart city reference model: Assisting planners to conceptualize the building of smart city innovation ecosystems. *J. Knowl. Econ.* **2013**, *4*, 217–231. [[CrossRef](#)]
83. Kim, S.A.; Shin, D.; Choe, Y.; Seibert, T.; Walz, S.P. Integrated energy monitoring and visualization system for Smart Green City development: Designing a spatial information integrated energy monitoring model in the context of massive data management on a web based platform. *Autom. Constr.* **2012**, *22*, 51–59. [[CrossRef](#)]

84. Qi, L.; Xu, M.; Fu, Z.; Mira, T.; Zhang, X. C 2 SLDS: A WSN-based perishable food shelf-life prediction and LSFO strategy decision support system in cold chain logistics. *Food Control* **2014**, *38*, 19–29. [CrossRef]
85. Li, F.; Qiao, W.; Sun, H.; Wan, H.; Wang, J.; Xia, Y.; Xu, Z.; Zhang, P. Smart transmission grid: Vision and framework. *IEEE Trans. Smart Grid* **2010**, *1*, 168–177. [CrossRef]
86. Nkomo, M.; Hancke, G.; Abu-Mahfouz, A.; Sinha, S.; Onumanyi, A. Overlay virtualized wireless sensor networks for application in industrial internet of things: A review. *Sensors* **2018**, *18*, 3215. [CrossRef] [PubMed]
87. Rahman, M.T.; Salaudin, M.; Maharjan, P.; Rasel, M.S.; Cho, H.; Park, J.Y. Natural wind-driven ultra-compact and highly efficient hybridized nanogenerator for self-sustained wireless environmental monitoring system. *Nano Energy* **2019**, *57*, 256–268. [CrossRef]
88. Doshi, S.; Dube, S. Wireless Sensor Network to Monitor River Water Impurity. In Proceedings of the International Conference on Computer Networks and Communication Technologies, Toronto, ON, Canada, 13–14 July 2019; pp. 809–817.
89. Fresnel-CL Wiki. Available online: <http://www.wiki.cl.cam.ac.uk/rowiki/Fresnel?> (accessed on 10 May 2019).
90. Virtualized Sensing Environments (ViSE) and DiCloud. Michael Zink (UMASS) | Research Training Group. Available online: <http://www.gkmm.tu-darmstadt.de/?q=node/871> (accessed on 22 March 2017).
91. Balani, R.; Han, C.-C.; Rengaswamy, R.K.; Tsigkogiannis, I.; Srivastava, M. Multi-level software reconfiguration for sensor networks. In Proceedings of the 6th ACM & IEEE International conference on Embedded Software, Seoul, Korea, 22–25 October 2006; pp. 112–121.
92. Desnoyers, P.; Ganesan, D.; Li, H.; Li, M.; Shenoy, P.J. PRESTO: A Predictive Storage Architecture for Sensor Networks. *HotOS* **2005**, *5*, 12–15.
93. Li, M.; Ganesan, D.; Shenoy, P. PRESTO: Feedback-driven data management in sensor networks. *IEEE ACM Trans. Netw.* **2009**, *17*, 1256–1269.
94. Webdust. Available online: <http://ru1.cti.gr/projects/webdust> (accessed on 10 May 2019).
95. Murty, R.N.; Mainland, G.; Rose, I.; Chowdhury, A.R.; Gosain, A.; Bers, J.; Welsh, M. CitySense: An urban-scale wireless sensor network and testbed. In Proceedings of the 2008 IEEE Conference on Technologies for Homeland Security, Waltham, MA, USA, 12–13 May 2008; pp. 583–588.
96. Navarro, M.; Antonucci, M.; Sarakis, L.; Zahariadis, T. Vitro architecture: Bringing virtualization to wsn world. In Proceedings of the 2011 IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS), Valencia, Spain, 17–22 October 2011; pp. 831–836.
97. Sanchez, L.; Muñoz, L.; Galache, J.A.; Sotres, P.; Santana, J.R.; Gutierrez, V.; Ramdhany, R.; Gluhak, A.; Krco, S.; Theodoridis, E.; et al. SmartSantander: IoT experimentation over a smart city testbed. *Comput. Netw.* **2014**, *61*, 217–238. [CrossRef]
98. Berkers, F.; Roelands, M.; Bomhof, F.; Bachet, T.; van Rijn, M.; Koers, W. Constructing a multi-sided business model for a smart horizontal IoT service platform. In Proceedings of the 2013 17th International Conference on Intelligence in Next Generation Networks (ICIN), Venice, Italy, 15–16 October 2013; pp. 126–132.

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).