Enhancing Dependability of Cloud-based IoT Services through Virtualization

Kashif Sana Dar, Amir Taherkordi and Frank Eliassen
Networks & Distributed Systems Group, Department of Informatics
University of Oslo, Norway
Email: {kashifd, amirhost, frank}@ifi.uio.no

***Abstract*—As Internet of Things (IoT) technology moves forward, more and more IoT provided services are being pushed toward clouds. Since the operation of IoT services runs the risk of failures due to lossy communication links and error prone nature of physical objects, cloud providers (offering such services) should provide suitable platforms supporting two desired service dependability features—i.e., reliability and availability. This issue has so far been addressed for specific application scenarios and often at the network layer. In this paper, we therefore aim at proposing a generic, model-based approach for enhancing these two important features at the application layer of cloud-based IoT systems. Following the principle of *dependability by design*, we build a framework based on the concept of *virtualized IoT services*, promising a service abstraction model to efficiently and simultaneously meet the dependability requirements of multiple cloud-based IoT applications. The proposed virtualization approach supports a variety of different dependability patterns and implements them according to the demands of the target application. We implemented the virtualization framework using the SicthSense cloud platform with satisfactory evaluation results on dependability metrics, such as *maximum availability* and the *probability of failure on demand*.**

I. Introduction

The Internet of Things (IoT) is rapidly being proposed for scenarios where various smart *things* or *objects*—such as Radio-Frequency IDentification (RFID) tags, sensors, mobile phones, etc.—are supposed to interact with each other through unique addressing schemes in a pervasive fashion. Being tightly integrated with the physical world, a degree of unreliability and uncertainty is introduced by things into end-user IoT applications, in addition to the fact that wireless links are lossy, unstable and sensitive to environmental conditions [1], [2]. The massive scale deployments of IoT devices make the dependability concern more critical due to external (e.g., water infiltration) or internal natural processes (e.g., power transient) that cause physical deteriorations [3], [4], [5], resulting in questionable quality of user experience and quality of information. We target both reliability and availability features of dependability for IoT devices. These features become more important especially when IoT devices (henceforth called sensors as well) are shared among several application scenarios since they are being employed to construct a plethora of smart-* applications.

For instance, consider a smart home that has a security system with perimeter intrusion sensors (e.g., occupancy sensor) on doors and windows, and motion detection sensors in some indoor areas. Additionally, a home typically has smoke sensors that, in case of fire, can inform the fire-station via the *fire alarm* application. It contains a thermostat for *indoor temperature sensing*, connected to a *cooling* or *heating* system. Now, consider all of these sensors are accessible from a common interface (e.g., a private cloud [6]) and are shared among all applications for home automation. For instance, the perimeter intrusion and motion sensors could be used in an *activity tracking* application. These sensors along with the thermostat could be used in a home *energy control* application. Similarly, the *activity tracking* could also be used for an *assisted living* application (Fig.1). Moreover, by sharing existing sensors, adding additional application specific sensors to such an existing infrastructure is much cheaper than adding an entire sensor suite for each new application.

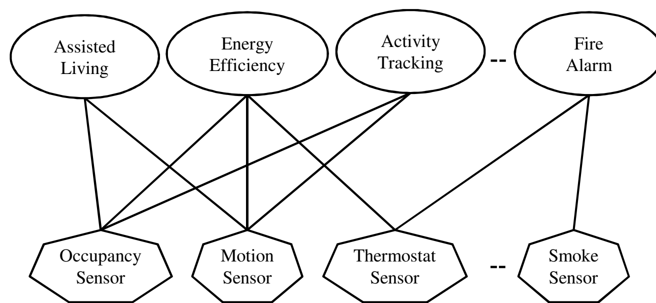

Fig. 1. Different application scenarios.

To enable this vision, one main problem is that each application will have its own sensing and dependability requirements. For instance, the sensors' data accuracy and availability may have more strict dependability requirements in an *assisted living* application than in an *activity tracking* application. These devices are therefore required to reliably convey the monitoring information and remain available for a maximum time period as per consumer application demands. Failure in delivering the sensed values in a reliable and timely fashion may result in high costs, insufficient user satisfaction, and even physical hazards to people or things [7]. Consequently, as the range of applications extends to even more safety-critical systems [8], additional dependability requirements may be introduced to the system.

To cope with this problem, we need a rigorous solution

that allows applications to specify their sensing and associated dependability requirements, transforms them into enactable software artefacts, and coordinates between applications and IoT devices to fulfil the required sensing functionality along with their expected trustworthiness. Towards this goal, we envisage the following desirable dependability properties for IoT applications:

- How to specify application-specific dependability requirements in a more declarative way so that they can be autonomously processed.
- How to provide a trustworthy mechanism on top of error-prone devices to fulfil a certain level of dependability.

IoT dependability has become one hot topic of research [9], [10] in recent years since today's approaches do not guarantee dependability in building IoT systems [7], [11]. Moreover, the broader picture of design of robust IoT applications is not clear yet [12]. Dependability modelling and analysis techniques [13] help understand the dynamics of IoT environments but are more restricted to quantify the reliability in given network conditions. The techniques that actually aim to enhance the dependability concentrate only on a specific problem e.g., end-to-end data reliability [14], adaptive routing [15], and improving reliability of multi-hop WSN protocols [16], etc. However, a generic approach is still required to lay the scientific foundation for an IoT system that works dependably and is resilient against variety of different failure types.

To address the above challenge, we exploit the concept of *virtualized IoT services* i.e., the sensed data acquired by a set of physical devices can be collected and processed according to an application-provided virtualization function and dependability requirements. Using *virtual services* (henceforth also called services), the programmer focuses on the application logic, rather than on low level implementation details and rigorously integrating the dependability requirements. It should be noted that IoT device virtualization is not a novel approach [17], [18], [19]. Whereas the common goal of reported works has been to enable the *sharing* of sensor-provided services among multiple applications, our virtual service design concept is distinguished from others with respect to its special consideration to the dependability needs of IoT applications. In addition to sharing IoT services, our virtualization approach supports a variety of different dependability patterns and implements them according to the demands of the target application. Existing virtual services in our system can also be reused in subsequent new applications if they depend on equivalent sensing functionality with the same level of dependability.

To demonstrate the feasibility of our proposed framework, we implement both real and simulation based test-beds. Considering both random and systematic failures, our simulation results show the effectiveness of the use of our framework with 98.57% of *maximum availability* under 40% random nodes failure, 0.45% of *probability of failure on demand* under 10% of nodes failures and 95.5% of net *producibility* of the system under 10% of a partial network failure.

In the rest of this paper, we first present the concepts of dependability and virtualization techniques for IoT systems and discuss our contribution in the context of existing literature produced during the recent years. Section III describes the design of our proposed framework with dedicated sub-sections for each design aspect of the framework. In Section IV, we describe the real test-bed implementation using Tmote Sky nodes followed by the detailed simulation-based results presented in Section V. Finally, in Section VI we discuss the achieved evaluation results and make the concluding remarks.

II. Background and Related Work

Dependability is an integrating concept that incorporates the reliability, availability, integrity, maintainability, safety, and security attributes of the system [20]. In this paper, we consider only reliability and availability in the context of cloud-based IoT services, henceforth, *using the term dependability collectively for both of these features*. According to ANSI[1], software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment. The availability is the amount of time a system is actually operating as the percentage of total time it should be operating. A highly available system would disable the malfunctioning portion and continue operating at a reduced capacity. We follow these definitions and aim to enhance the capability of IoT systems by masking failures in their critical services. This is often achieved through redundancy [9] as the redundant system components are often treated as replacement for primary components when they are failed.

On the other side, virtualization, during the past few years, has evolved as an efficient technique that allows the abstraction of actual physical computing resources into logical units, enabling their efficient usage by multiple independent users. In the context of IoT, virtualization has become an established technique to share sensor resources over the network [19], [18].

Below, we elaborate on the most recent work carried out for both dependability and virtualization in the context of Wireless Sensor Networks (WSN) in general and IoT in particular.

A. Dependability

A wide range of strategies toward dependability for low powered resource-scarce devices include dependability planning [21], [22], modelling and analysis [9], [3], [4], [23], and enhancement techniques [13]. The description of each is as follows.

1) Dependability Planning: Dependability planning deals with the problem of finding a well-balanced level of dependability during the application design. A set of system level guidelines are provided in [10] to identify the reliability requirements for IoT applications to ensure reachability to all the sensors and replacement of faulty nodes. For the safety critical embedded systems, [21] investigates the best engineering practices (design patterns) and discusses

[1]American National Standards Institute

variety of different software and hardware-based redundancy techniques. Similarly, [24] proposes a mathematical model equipped with a mix of redundancy providing components considering either active and/or passive redundancy. However, we believe that the proposed model is not verified for real IoT scenarios.

2) Dependability Modelling and Analysis: The most common dependability modelling techniques include Fault Tree Analysis (FTA), the Reliability Block Diagrams (RBDs), and reliability graphs. In [5], the fault trees are dynamically generated out of present network topology to detect network faults, which is further extended in [9] through the Markov Chain based redundancy model. A counter-part approach for above is the DRBD [4] that explains how a WSN can be depicted through the DRBDs under different failure conditions and different network topologies. Similarly, in [3], authors exploit reliability graphs and represent a typical WSN through an undirected graph. The above techniques are more suitable to detect network failures and assess network reliability, instead the modelling scope we target in this paper is to *enhance* the dependability of IoT systems through redundancy patterns.

3) Dependability Enhancement: The current dependability enhancement approaches target reliable data transfer e.g., network coding [14], multipath data sending [15], and reliable sensor data fusion [25]. Similarly, authors in [26] demonstrate how to make communication protocols more robust against temperature fluctuations. In [12], the same authors explain how to perform an automatic protocol configuration to increase dependability of IoT applications.

An alternative category for enhancement lies in fault-injection experiments [27], [28]. These approaches aim at accelerating the occurrence of faults for the purpose of assessing the effectiveness of built-in detection and recovery mechanisms. Normally, to inject such faults, simulation-based techniques are used during the development.

B. Virtualization for Smart Devices

In embedded systems, a common trend is to introduce in-node or network level virtualized environments that provide an integrated access of these devices. The node level virtualization allows multiple applications to run their tasks on a single node in a sequential and/or concurrent fashion e.g., implementation of a tiny virtual machine [29]. In network level virtualization, a virtual sensor network is formed by a subset of actual WSN's nodes dedicated to one application at a given time. In this category, two main approaches are based on *service orientation* and *aggregation* function.

Within service based techniques, [17] presents a high level architecture and implementation details of the IoT-VN (Virtual Networks) for both constrained and non-constrained devices. Similarly, [19] proposes a novel architecture based on: i) service enablers (provide actual IoT services), ii) service providers (guarantee their availability), and iii) end users. They ensure connectivity through a gateway node which plays a major role in providing network-level virtualization and maintains several registries to create and manage virtual networks. However, sensor nodes willing to be part of the virtual network must be aware of functionalities offered by the gateway node.

Within aggregation based techniques, [30] defines virtual sensors with the help of aggregation function and design a middleware that provides high level Application Programming Interfaces (APIs) for programming these virtual sensors. Similarly, [31] defines a virtual sensor as the function of input, output, and configuration parameters. They further define the composition of high level virtual services from the existing ones by introducing virtual service *manager* and a *buffer* to store sensor data which is further accessible by other applications.

C. Discussion on Related Work

There are many recent efforts that have investigated both dependability and virtualization functionality within IoT environments. One major limitation of works on dependability is that the underlying design is quite proprietary to a single application scenario [16], [31], [3] and has not been thought in a broader sense to target general purpose dependability features that can be configured to the specific needs of each IoT application. Furthermore, dependability modelling and analysis [32], [3], [23], [21] techniques consider static arrangement of different system components failures (e.g., battery depletion, communication link, sensor hardware or software) and evaluate the overall system reliability by investigating individual components' probability of being functional at a given time in given circumstances. While these efforts provide abstractions by mathematically formalizing the concepts of reliability within IoT, they still lack practical dependability issues of actual deployment problems and therefore the effectiveness of fault tolerant mechanisms for real dynamics of the network is still needed to be investigated. Also, dependability enhancement techniques only focus on specific issues e.g., reliable data transfer [14], [33], communication links redundancy [15], effect of the external environment on IoT communication [26], and simulation-based fault-injection experiments [27], [28] to test dependability. We extend the latter techniques by introducing reliability features at the very initial stage of IoT applications' design and facilitate to dynamically enact those requirements through our proposed framework.

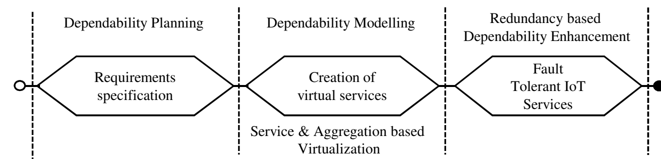

Fig. 2. Our contribution in the context of existing literature.

Figure 2 depicts our contribution landscape in the context of existing literature. Believing that dependability should be an important design consideration, we exploit the dependability planning techniques in [21] to design application's

requirements. In order to fulfil these requirements, we adopt a model-based approach [3], [23] and exploit both service-oriented [17], [18], [19] and aggregation based [34], [30] virtualization techniques. To realize this model, we create a virtualized environment that weaves the application dependability demands with available dependability patterns, based on component redundancy. We then provide support for invocation of IoT services through our previously developed framework [35], and process the acquired data using the specified virtualization function.

Using virtualization as a technique to enhance dependability will have several advantages. First, applications will be releaved from having to deal with the complexity caused by service failure since such failures will be managed at the virtualization layer. Second, since failure in a single device has the potential to spread among numerous applications using its services, risk of cascading effects of failure propagation in such scenarios will be mitigated. Failure management increases complexity especially when it is handled at the application level which is translated into higher workloads for developers and managers. Thus, it is wise to provide a common middleware framework that collectively provides sensing functionality along with the desired level of dependability and failure management capabilities.

III. System Design

Before we describe details of the system design, we identify the main design requirements and assumptions made during the design process.

A. Design Requirements

We identify three main requirements which are basically described to achieve a balance between applications' dependability demands and the constraints of underlying IoT resources.

1) *Dependability requirements specification:* IoT-aware applications should be capable to specify what are their dependability requirements associated with each sensing task. These requirements must be described in a *declarative way* so that they can be processed autonomously.
2) *Virtual services creation and management:* Enable a virtual environment consisting of on-the-fly created virtual services that fully conform to not only functional but also dependability requirements specified by the end-user application. This also implies that existing virtual services should be re-usable among other applications.
3) *Reliable services on top of volatile things:* The services, created as a result of above requirement, should form a reliable infrastructure on top of error-prone IoT devices in order to maximize their availability. This requires an insight into existing redundancy based dependability patterns and their potential impact on IoT-aware applications.

We believe that, by fulfilling the above requirements, IoT-aware applications will become loosely coupled with actual IoT services and will remain unaffected under continuous fluctuation of their availability. Below, we first describe the failure types that we consider in this paper..

B. Failure Types

Within typical IoT environments we consider the sensor node level *crash* that causes permanent failure. This may be either due to hardware or software failure. It is important to note that we do not consider the failure in intermediate/receiving node and communication link and assume that node-level failure is *consistent* i.e., node's service failure is observed by all of its users and is *detectable* i.e., when node fails, it changes to a state that allows others to detect its failure and then stops. Finally, to attain dependability we consider *fault tolerance* technique instead of fault prevention, removal, and forecasting.

C. Assumptions

Due to several dependencies of our proposed solution with other sub-systems, we consider the following assumptions to hold when it comes to the integration of our framework with other components.

- Service discovery: The mechanism to discover target IoT devices participating in each sensing task is already in place i.e., for a given task, we already have a list of devices along with their behavioural description[2].
- Service maintenance cost: We do not take into account the service maintenance cost i.e., failure detection, recovery, restart, and resume time.
- Redundancy type: In this paper, we only consider active redundancy in which active components are running in parallel and do not consider passive redundancy. This is due to the fact that we do not handle the additional functionality needed to activate and switch to the standby components when the primary component fails.

D. Requirements Specifications

We follow a similar domain model as presented in [36] to formulate the sensing requirements. Below, we first describe terminologies we use for this purpose.

- *Application Domain:* refers to a particular business area to which an IoT-aware application belongs such as Health care, Logistics, Smart Building, Smart City, and Smart Traffic.
- *Entity of Interest (EoI):* The potential physical object within the real world which needs to be monitored, controlled, or tracked. For example, patients in healthcare, trucks or food items in logistics, rooms in a building application domain, etc.
- *Property of Interest (PoI):* is the desired monitoring property associated with an EoI. For example, location (PoI) can be associated to a vehicle (EoI) to track.
- *Observation Rate:* A PoI associated with an EoI is monitored through different level of frequencies e.g., periodic or event based monitoring.

[2]describe their offered functionality and their exposed interfaces

- *Dependability Level:* represents a certain level of reliability and availability. We define several levels from low to high depending upon the dependability requirements.
- *Virtualization Function:* defines the data processing logic for virtual services. It may be a simple aggregation or a Boolean function.

Now, let A, D, E, and P represent the set of all possible applications, application domains, EoIs, and PoIs respectively. Let, $L = \{l \mid l \in \{low, medium, high\}\}$ be the set of possible dependability levels,
$Q = \{q \mid q \in \{periodic, range, threshold\}\}$ is the set of possible rates of observation, and
$F = \{f \mid f \text{ a virtualization function}\}$ is the set of supported virtualization functions.
Then the set of sensing requirements R for any application can be represented as follows:
$R = \{(d, e, p, l, q, f)$
where $d \in D, e \in E, p \in P, l \in L, q \in Q, f \in F\}$

E. Redundancy Models

Our solution is based on the active redundancy (our assumption) in which redundant nodes are continuously functioning along with the primary node that offers services. Redundancy is a well known approach to cope with dependability problems. Despite the fact that this topic is very less explored in the context of IoT [9], there exist variety of different ways of implementing it for traditional systems. Following, we discuss three models of redundancy and will eventually evaluate them in the context of IoT.

1) Duplex redundancy: The duplex model is used to increase the dependability of the system by providing a replication of the same component (Modular redundancy) to deal with the random faults. The idea of providing an additional component is based on the assumption that it is highly unlikely for two identical components to suffer a random fault simultaneously.

2) Triple Modular Redundancy: This model consists of three identical components which operate in parallel to produce three results that are compared using a voting system to produce a common result. This structure allows the system to operate and provide functionality as long as two components or more have the same result.

3) M-Out-Of-N Redundancy Model: A M-oo-N model works with N identical components which operate in parallel. The M-oo-N redundancy requires that at least M components succeed out of the total N parallel components for the system to succeed. A M-out-of-N voting algorithm is used in the voter component to allow the system operating and providing the required functionality in the presence of random faults without losing the input data.

F. Problem Formalization

We capture the overall picture of IoT by considering different application domains. Therefore, we uniquely represent each domain and allow applications to specify their sensing requirements from one or multiple domains. Within each domain, real world entities (EoIs) are associated with different PoIs along with a specific dependability level, frequency of observation, and a virtualization function to collect and process sensed values.

For example, as envisioned in the example of *smart home* in Section I, the fire alarm application consumes temperature and smoke values and invokes a fire event if corresponding values exceed a particular threshold. This typical functionality can be achieved by fulfilling the following three requirements.

- r_1 - get an average temperature value (out of two devices) from room#1, every 10 seconds (sec) with a medium level of dependability.
- r_2 - get a maximum smoke value (out of three devices) from room#1 every 5 sec with a high level of dependability.
- r_3 - generate a fire alarm if temperature value exceeds 40° and smoke value exceeds 30 with a high level of dependability.

From the above illustration, we see that r_1 and r_2 are similar in a way that they need to directly fetch data from physical devices. Whereas, r_3 requires only to process sensor data gathered as a result of fulfilling r_1, and r_2. Thus the above requirements can be collectively represented as a set of three sub-requirements:

$$R_A = \{r_1, r_2, r_3\}$$

where,
$r_1 = \{\texttt{Smart Home,Room1,Temperature,Medium,10 sec,Average(2)}\}$
$r_2 = \{\texttt{Smart Home,Room1,Smoke,High,5 sec,Maximum(3)}\}$
$r_3 = \{\texttt{Smart Home,Room1,Fire-Alarm,High,}\phi\texttt{,f(}r_1\texttt{,}r_2\texttt{)}\}$

where $\texttt{f(}r_1\texttt{, }r_2\texttt{) = (}|r_1|^3 \texttt{ >= 40}^\circ \texttt{ \&\& } |r_2| \texttt{ >= 30)}$ is a Boolean function that specifies when a fire alarm should be activated. Note that the ϕ value in r_3 points out that r_3 does not have any frequency of observation since it only needs to process data obtained through other requirements. Despite this fact, the high level information generated by it may have its own dependability requirement.

Now, let M represent the supported redundancy models (described previously) and S be the set of physical devices. Then our virtualization problem can be described as follows:

"Create a set of *virtual service(s)* $V = \{(r, s, m) \mid r \in R, s \in S, m \in M\}$ where each $v = (r, s, m)$ conforms to the virtualization requirement r by *using* a suitable redundancy model m that manipulates a set of physical devices s."

By following this, we now describe the detailed system description followed by an algorithm to create virtual services within our system.

G. System Description

A conceptual diagram of our proposed virtualization framework is shown in Figure 3. It mediates between IoT applications and devices belonging to different IoT domains. Each

[3] $|r_i|$ denotes an accumulated result obtained from r_i

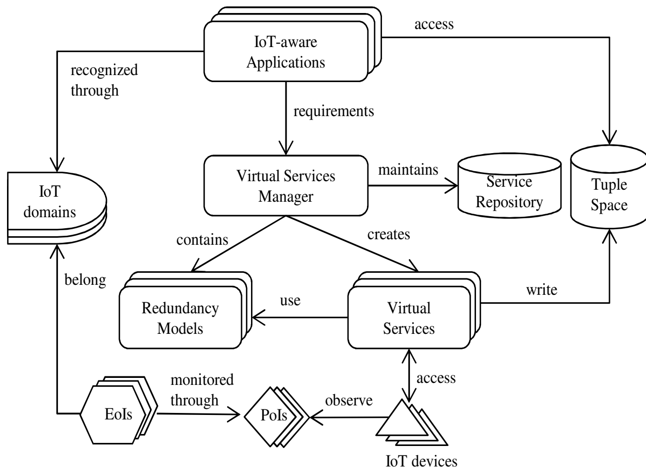

Fig. 3. The conceptual diagram of the proposed virtualization framework.

IoT device monitors a set of PoIs associated with EoIs and exposes its functionality through a behavioural description represented by a unique document (e.g., Web Service Description Language)[4] WSDL version 2.0. We describe this device level representation in our previous work [35] in which the service on each device is treated as a REST-endpoint[5] and is exposed as a RESTful service. It is worth noting that, in order to invoke services hosted by IoT devices, the same framework [35] is used that offers high-level APIs to invoke IoT services using IP-based communication. We assume that the list of potential devices will be available to the *Virtual Services Manager (VSM)* through an external *discovery mechanism* along with their WSDL files.

On the other hand, the application developer can specify the application *requirements* as described in the previous section. Algorithm 1 describes how the VSM sequentially processes these requirements in order to ensure that for each requirement there is at least one virtual service in the system. For this purpose, the *VSM* first ensures if there already exists a virtual service in the *Service Repository* that fulfils the same requirement - both functional (sensing) and non-functional (dependability). The *Service Repository* provides a catalogue of currently running virtual services within our framework. Each virtual service is recognised through a globally unique identity generated from its associated requirement. Therefore, the VSM performs a local identity-based search operation to ensure de-duplication of the virtual services. In case, a service already exists, the VSM simply adds this to the target service set V. Otherwise, it creates a new virtual service as explained in the Algorithm 1 and creates its entry in the *Service Repository*.

In addition, the class diagram in Figure 4 shows how different functionalities regarding the implementation of relevant redundancy model and virtualization function are encapsulated in different classes. The *Requirement* class aggregates both sensing and data processing logic of the application. The required dependability model is then implemented using the right class derived from the *DependModel* abstract class. Finally, an object of *VirtualService* class is instantiated which uses both the *Requirement* and relevant *DependModel* special classes to enact a virtual service.

[4] http://www.w3.org/TR/wsdl20-primer/
[5] http://rest.elkstein.org/

It is worth noting that within our framework, there are two types of services created by the VSM: *periodic* and *event-based*. The former periodically fetches data from IoT devices and populate the *tuple space* that is accessible by applications through APIs provided by the VSM. The event-based services follow the *push* model by monitoring an event. For example, for r_1 and r_2, from the previous section, the VSM will create a periodic service for each to monitor temperature and smoke values, however, for r_3, it will create an event-based service that will report only when a fire event is detected. This is typically implemented using a callback function.

However, once the virtual service is enacted, it will fetch data from the target IoT services and will populate the *tuple space* according to the assigned observation rate. Currently, the framework only supports pulling data from the devices, however, a device push model will also be implemented in the future.

Having such virtualizion framework will have a number of advantages when it comes to services reusablity and maintenance. The services are easily reused among other applications if they need similar functionality with the same level of dependability. Further, in case a service needs to be replaced with a better alternative, its related old data will be still available from the *tuple space* during the service maintenance task.

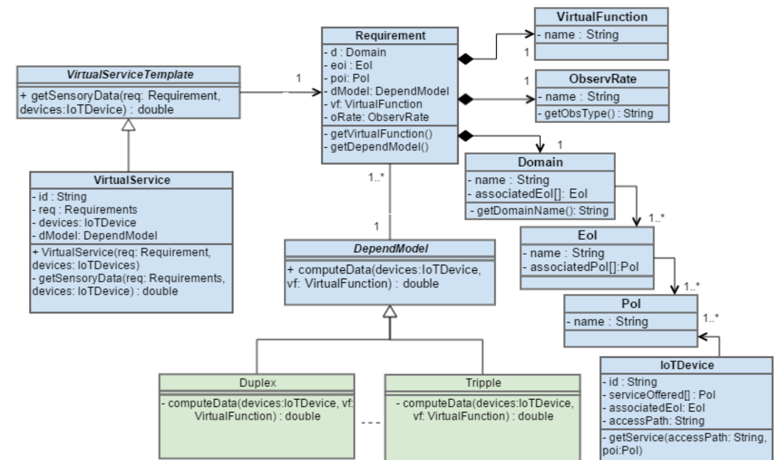

Fig. 4. Class diagram of the virtualization framework.

IV. IMPLEMENTATION

We implemented the virtualization framework in Java using *template* and *factory* design patterns. The former is used to implement different redundancy models and instantiate one of them depending on the scenario while the latter is used to

Algorithm 1 Creation of Virtual Services

Require:
1: Set of pairs (r, s) where $r \subseteq R$ and $s \subseteq S$
2: Sets of M and F mentioned in Section III-D
Ensure:
3: Set of V

```
4: procedure BUILD–VS
5:     for each r ∈ R do
6:         v ← EXIST-VS(r)
7:         if v ≠ φ then
8:             V ← v
9:         else
10:                ▷ Create a new service from given r and s
11:            V ← CREATE-VS(r, s)
12:        end if
13:    end for
14:    function CREATE-VS(r, s)
15:            ▷ Implement required Redundancy Model (RM)
16:        for each m ∈ M do
17:            if r.l = m then
18:                v ← IMPLEMENT-RM(m, s)
19:                BREAK
20:            end if
21:        end for
22:            ▷ Implement the Virtualization Function (VF)
23:        for each f ∈ F do
24:            if r.f = f then
25:                v ← IMPLEMENT-VF(f)
26:                BREAK
27:            end if
28:        end for
29:            ▷ Get a service type by checking frequency (q)
30:        if r.q ≠ φ then
31:            v ← GET-TYPE(periodic)
32:            v ← SET-PERIOD(r.q)
33:        else
34:            v ← GET-TYPE(eventing)
35:                ▷ Implement call back function for VF (r.f)
36:            v ← IMPLEMENT-CALLBACK(r.f)
37:        end if
38:        return v
39:    end function
40:    function EXISTS-VS(r)
41:        for each service v ∈ V do
42:            if v.r = r then ▷ Find service with similar r.
43:                return v
44:            end if
45:        end for
46:        return φ
47:    end function
48: end procedure
```

create different types of virtual services implementing their own virtualization functions and redundancy models.

To implement a real test-bed, we deploy a small network of Contiki[6]-based Tmote Sky[7] nodes along with a private instance of SicsthSense[8]—a Java-based open cloud platform for the IoT. We chose SicsthSense as our cloud platform because it is compatible and convenient to deploy along with our previously built framework [35] that we used to invoke services hosted by IoT devices. Both of them are implemented in Java and offer REST-based communication.

The Tmote Sky node that we used contains an 8 MHz MSP430 microcontroller, a CC2420 radio chip, 48 KB of programmable flash, 10 KB of RAM, and light, temperature, and humidity sensors. We use a small single-hop network of 6 nodes with static routes. Among these nodes, one node implements the 6LoWPAN[9] border router connected via USB to a Ubuntu computer. For the rest of the nodes, each node implements a single service that provides the current temperature value in the external environment.

SicsthSense enables low power devices to easily store their generated data streams in the cloud. This allows the data streams to be made permanently and globally available to users for visualisation, processing and sharing. These streams are a sequence of scalar numeric values arranged through time. Each stream is a logical grouping of some data point measurements that users are interested in (such as temperature). SicsthSense can be configured to populate these streams by its interaction with external *Resources*. A *resource* is a single source/device that provides information, such as a Tmote Sky node. Each resource may provide multiple different data points upon each interaction with it. For example, a Tmote Sky resource may provide a data payload containing the temperature and light. This payload then has to be processed and split into two different streams for storage. This splitting procedure is performed by *Parsers* that specify which bits of data in a payload are of interest to the user. In short, users have many resources containing *parsers* and time series of *streams*.

Figure 5 shows our implemented test-bed. Consider a typical application specifies a sensing requirement to the Virtualization Framework that requires to create a *periodic-based* virtual service. The VSM processes the requirement and creates a virtual service as explained in the previous section. This virtual service then creates a unique resource into the SicsthSense server and creates one or more data streams associated with that resource. The resource is exposed by SicsthSense through HTTP-based API (Constrained Application Protocol[10] (CoAP) is not supported in its current version). These data streams then act as a tuple space from where data can be accessed by simply making a HTTP GET request. To make these streams persistent, the SicsthSense server stores them into a relational database – MySQL in our case. On the other hand, if

[6]http://www.contiki-os.org/
[7]http://tmote-sky.blogspot.no/
[8]http://presense.sics.se/
[9]http://6lowpan.net/about/
[10]https://tools.ietf.org/html/rfc7252.

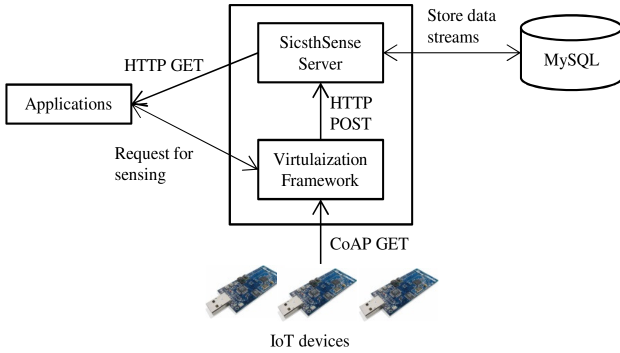

Fig. 5. Real implementation test-bed.

VSM needs to create an event-based virtual service, it follows the aforementioned procedure in addition to implementing a separate call-back function for pushing data to relevant application(s).

After successfully implementing the above test-bed, we calculated both the *build time*—time taken to transform an application requirement into an enactable service, and *communication delay*—time taken to make a CoAP request to fetch data from sensors. We noticed that the average build time is 9 millisecond (ms), whereas the average communication delay is 346 ms – average of 10 values. By these values, we observe that the build time is negligible compared to the communication cost and therefore it will not impose any considerable overhead in term of processing time when our framework is used. However, it should be noted that we do not consider further processing delay, i.e., making HTTP GET/POST requests to SicsthSense as, unlike our single machine based implementation, it may vary depending upon the actual scenario implementation.

V. Evaluation

Dependability of a software-based system is hard to measure and even sometimes difficult to define. To quantitatively express the dependability of a software product, the choice of metrics depends upon the type of system to which it applies and the requirements of the application domain. Therefore, we will use following metrics to evaluate our results:

- *Failure Intensity*: describes the percentage of failed nodes out of total number of nodes in the network. A failure intensity of 2% means that 2% nodes of the whole network are dis-functional.
- *Probability of Failure on Demand* (PFD): is defined as the probability that a system will fail when a particular service is requested. It is the number of service failures given the total number of service calls. A PFD of 0.1 means that one out of ten service requests may result in failure.
- *Producibility*: is defined as the ability of a system to produce the desired result, both accurate (ideal value) or near-accurate (acceptable value). For example, the user may specify to accept an old value (which is not expired yet) in case fresh data is not available due to any failure. Thus, producibility is quantified as the probability of system being able to produce the desired result.
- *Availability* (AVAIL): is the probability a system is available for use at a given time. It takes into account the repair time and the restart time for the system. Therefore, it is the percentage of time that a system is available for use, taking into account planned and unplanned downtime. If a system is down for an average of four hours out of 100 hours of operation, its AVAIL is 96%.

TABLE I
Simulation test-bed configuration.

Parameter	Value
Total number of nodes	20000
Network Arrangement	A 40×500 matrix of mesh network
Total number of operations	50000
Redundancy models distribution	Random among all operations
Failure intensity	Up to 100% with 0.5% increase rate
Failure patterns	Random and systematic

Below we describe our simulation-based test-bed description to calculate the above metrics.

A. Test-bed Description

Table I describes our Java-based simulation test-bed that we used in all our experiments. The scale of network size is carefully chosen by studying the existing large-scale IoT applications, e.g., [37] that uses a well estimated number of sensor devices for a smart city application. This project rolled out over 20000 devices (such as sensors, repeaters, gateways and mobile handsets) in Santander and the surrounding area enabling various applications in areas such as public transport and mobility, traffic, pollution, waste management, noise control, etc. However, the choice of total number of operations to be performed on a given network varies from one application scenario to another. Therefore, we selected a reasonable amount of operations so that we can at least access each node in the selected network. Finally, we used both *random* and *systematic* (common cause) failure patterns.

B. Random Failures

We injected random failures in the network right from 0.5% to 100% failed nodes with the increase rate of 0.5% and used the redundancy models described in Section III-E (3-oo-5 in the MooN case) and randomly distributed them among all operations. Figure 6 shows the simulated network and illustrates how different types of operations are executed by the virtual services created within the Virtualization Framework. With this set-up, we calculate the overall availability of the system as follows.

1) Availability: We evaluated the number of recovered operations out of total number of operations performed at a given time with a given failure intensity. We define an operation as a single sensing request e.g., getting value of current temperature. Within our experiments, we define the following operations.

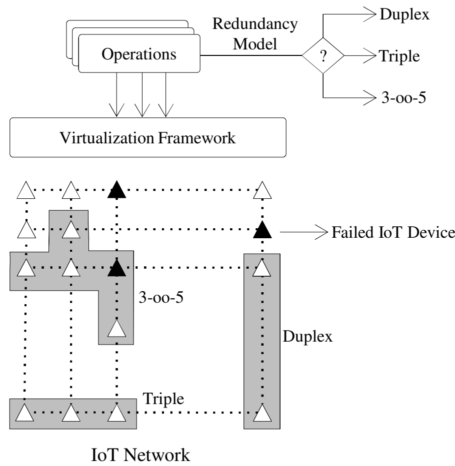

Fig. 6. Test-bed scenario for random and systematic failures.

- Fail Safe: yields desired result without facing any failure in primary nodes. The number of fail safe operations remains same with or without using our framework.
- Recovered: Despite failure in primary nodes, it yields the desired result due to recovery performed by redundant nodes. This metric is useful to calculate the effectiveness of our approach.
- Successful: represents the total number of operations which are fail safe and recovered. This metric is useful to calculate the overall availability of the system.
- Fail: represents the number of operations that were failed to produce the desired results despite the use of our approach.

As shown in Figure 7, initially the recovery of operations grows linearly as more failures are injected in the network with maximum of 99.32% successful operations, however, after reaching a certain threshold, which we achieved at 40%, it starts decreasing, mainly due to shortage of redundant nodes, and reaching to zero% of successful operations when 100% nodes are dis-functional.

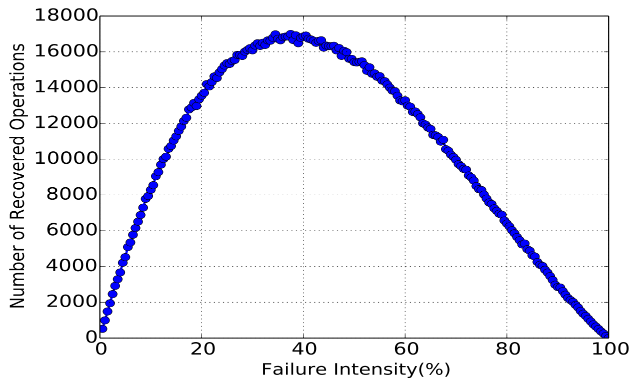

Fig. 7. Recovered operations in random failures.

With this test data, we finally calculated overall availability of the system (only up to the maximum performance) which is 98.57% (percentage of time the system was able to successfully perform the requested operations) since it goes through cycles of successful and un-successful operations during its whole execution time. However, we did not consider failure detection, recovery, and restart times since we did not implement any recovery mechanism.

2) Failure on demand: We calculate the probability of failure on demand (PFD) by calling a particular service at a given failure intensity with maximum of 10% of network failure. For this purpose, we use the same test-bed described above with the same configuration except, we reduce the total number of operations to 200 because we only want to check the system behaviour under on-demand service invocation. For each operation, we invoke a particular service instead of a random operation. Figure 8 shows that 99.55% of operations were successful (hence PFD = 0.45%). We observed that number of recovered operations also grows linearly, like the previous experiment, but in a more zigzag fashion. This is particularly due to the nature of operation performed, i.e., which part of the network is being accessed by the on-demand operation, accessing part of the network with majority of failed nodes means that the operation is likely to be un-successful.

Fig. 8. Recovered operations in on-demand service invocation.

C. Systematic Failures

We consider systematic failure mainly by introducing: *i)* partial network failure, and *ii)* injecting failure to most critical nodes. In the following, we describe each:

1) Partial Network failures: In this category, a sub-network consisting of a set of nodes becomes inaccessible due to the common cause failure. We used the same network layout as described previously and injected partial network failure of size 10% of the whole network right in the middle. We then, performed specified number of operations with: i) all redundancy models (randomly distributed), and ii) using only MooN (3oo5). In both cases, we observed almost constant number of recovered operations (the former with 10% and the latter with slightly increased value of 12.5%). The reason of this slight increase is that the latter contains more redundant nodes with increased fault tolerance capability. With only MooN redundancy, we further evaluated our test-bed with

TABLE II
PERCENTAGE OF DIFFERENT OPERATIONS IN CRITICAL NODES' FAILURE WITH NET PRODUCIBILITY = 95.5%

Node Degree	Fail Safe	Recovered (R)	Recovered (T)	Failed
2	79%	17%	2.5%	1.5%
3	74%	19%	4%	3%
4	70%	20%	5%	5.5%
5	67%	19.5%	5%	8.5%

increasing value of m (i.e., decreasing service accessibility with increased accuracy level) and found that higher the value of m, we are able to save fewer number of operations.

2) Failures in most critical nodes: We define the criticality of a sensor node by its degree of being shared among applications. For example, a node with degree 2 is shared between two different applications. Hence, a greater degree of node will represent its higher criticality.

We run our experiment with 100 applications with random access (can access any number of nodes) to the same network resources as described in the previous section. We first equally divide all the operations among applications and allocate them a fix number of nodes. By this, we calculate the degree of each node. We now repeat the experiment by performing operations with failures injected to nodes with degree from 1 to 5 (maximum) and compute the percentage of different types of operation (shown in Table II) with two additional types of operations defined as follows:

- Recovered through Redundancy (Recovered (R)): operations which face partial failures in primary nodes but yielded desired result due to standby nodes replacing them.
- Recovered through Tuple space (Recovered (T)): are those operations which, despite being failed, yielded near accurate result by fetching an old copy of data from the tuple space.

Table II shows the efficiency of the use of our framework. As the percentage of fail safe operations decreases due to failures injected in nodes with higher degrees, the number of recovered operations increases linearly. However, we see the slight decrease in the number of operations recovered when failure is injected to nodes with degree 5. This is mainly due to the fact that nodes with a higher degree will have more reusability and more applications will experience the same failure. However, the percentage of operations recovered due to the tuple space data is not affected and remains constant because there will be no failures on this level. In the end, we calculate the net producibility of the framework by averaging the number of successful operations which is 95.5%.

VI. DISCUSSION AND CONCLUSIONS

Exposing smart IoT devices as cloud services enhances complexities in building more robust IoT systems at scale. This work targets to deliver reliable IoT services at the cloud level. We propose a framework that assists IoT applications to specify their sensing requirements along with the expected dependability demands in term of components redundancy. The framework is based on a generic model that can include different redundancy patterns and use the virtualization technique to deploy those patterns at runtime in the form of virtual services. A virtual service within the framework also encapsulates the sensor data processing logic (by implementing application-specified virtualization functions) and sensing functionality to invoke IoT services (using our previously built communication framework). In this way, matching functional requirements with the expected level of dependability decreases the service maintenance cost and prevents potential ripple effects from the services dis-continuity later in the actual execution environment. In addition, the use of tuple space as sensor data buffer further adds value to the data dependability while compromising a little on data freshness.

To demonstrate the feasibility of cloud-based IoT services, we implemented a Contiki-based network of real Tmote Sky nodes within the SicsthSense cloud platform. From this real implementation, we observed that our framework has negligible processing overhead as compared to the overall processing and communication delay (using CoAP) for accessing sensor data.

We further performed extensive simulation tests to measure the performance of our framework under different types of failures and reported the relevant dependability metrics. For example, the maximum availability of the system is 98.57%, under 40% of random nodes failure in the network. Similarly, within the same network, the probability of failure on demand is calculated as 0.45% given a maximum of 10% nodes failure. Under the systematic failure category, we observed the net producibility of the system is 95.5% when a partial network of size 10% of the whole network becomes isolated or dis-functional.

Beside above results, our framework is highly tunable in terms of introducing more redundancy patterns and dependability levels. By considering the *dependability by design* principle, cloud-based IoT applications can find a good match of their dependability demands and available resources through the use of our framework. Despite the fact that we do not provide a recovery mechanism for failed operations (that we think is potential future work), the design of such recovery mechanism is easier with our approach since failure handling can be performed by only fixing the related virtual services (once recovered, we only need to re-deploy them). Moreover, this avoids the cascading effect of service failure prorogation.

VII. ACKNOWLEDGEMENTS

We gratefully acknowledge the guidelines and feedback provided by Professor Roman Vitenberg to improve the final version of this paper. We are also thankful to the anonymous reviewers for their careful reading of our paper and their many insightful comments and suggestions.

REFERENCES

[1] C. Boano, J. Brown, Z. He, U. Roedig, and T. Voigt, "Low-power radio communication in industrial outdoor deployments: The impact

of weather conditions and atex-compliance," in *Sensor Applications, Experimentation, and Logistics*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, 2010, vol. 29.

[2] Francoise Sailhan, Thierry Delot, Animesh Pathak, Aymeric Puech, and Matthieu Roy, "Dependable wireless sensor networks," 2009. [Online]. Available: http://cedric.cnam.fr/ sailhanf/publications/gedsip.pdf

[3] C. Jaggle, J. Neidig, T. Grosch, F. Dressler, "Introduction to model-based reliability evaluation of wireless sensor networks," in *2nd IFAC Workshop on Dependable Control of Discrete Systems*, 2009.

[4] S. Distefano, "Evaluating reliability of wsn with sleep/wake-up interfering nodes," *International Journal of Systems Science*, vol. 44, no. 10, pp. 1793–1806, 2013.

[5] I. Silva, R. Leandro, D. Macedo, and L. A. Guedes, "A dependability evaluation tool for the internet of things," *Comput. Electr. Eng.*, vol. 39, no. 7, pp. 2005–2018, Oct. 2013.

[6] S. Distefano, G. Merlino, and A. Puliafito, "Sensing and actuation as a service: A new development for clouds," in *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on*, Aug 2012, pp. 272–275.

[7] Carlo Alberto Boano, Kay Rmer and Thiemo Voigt, "RELYonIT: Dependability for the Internet of Things," *IEEE IoT Newsletter Januray 13, 2015*.

[8] N. Raveendranathan, S. Galzarano, V. Loseu, R. Gravina, R. Giannantonio, M. Sgroi, R. Jafari, and G. Fortino, "From modeling to implementation of virtual sensors in body sensor networks," *Sensors Journal, IEEE*, vol. 12, no. 3, pp. 583–593, March 2012.

[9] D. Macedo, L. Guedes, and I. Silva, "A dependability evaluation for internet of things incorporating redundancy aspects," in *Networking, Sensing and Control (ICNSC), 2014 IEEE 11th International Conference on*, April 2014, pp. 417–422.

[10] James Kempf, Jari Arkko, Neda Beheshti, Kiran Yedavalli, "Thoughts on reliability in the internet of things," in *Interconnecting Smart Objects with the Internet Workshop, Prague*, 2011.

[11] H. Pohls, V. Angelakis, S. Suppan, K. Fischer, G. Oikonomou, E. Tragos, R. Diaz Rodriguez, and T. Mouroutis, "Rerum: Building a reliable iot upon privacy- and security- enabled smart objects," in *Wireless Communications and Networking Conference Workshops (WCNCW), 2014 IEEE*, April 2014, pp. 122–127.

[12] F. J. Oppermann, C. A. Boano, M. A. Zúñiga, and K. Römer, "Automatic protocol configuration for dependable internet of things applications," in *Proceedings of the* 10th *International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp)*. IEEE, Oct. 2015.

[13] L. Venkatesan, S. Shanmugavel and C. Subramaniam, "A survey on modeling and enhancing reliability of wireless sensor network," *Wireless Sensor Network*, vol. 5, no. 3, pp. 41–51, 2013.

[14] Ting-Ge Li and Chih-Cheng Hsu and Cheng-Fu Chou, "On reliable transmission by adaptive network coding in wireless sensor networks," in *IEEE International Conference on Communications, 2009. ICC '09.*, June 2009, pp. 1–5.

[15] M. R. S. Saqaeeyan, "Improved multi-path and multi-speed routing protocol in wireless sensor networks," *IJCNIS*, vol. 4, no. 2, pp. 8–14, 2012.

[16] Braem, B.; Latre, B.; Blondia, C.; Moerman, I.; Demeester, P.,, "Improving reliability in multi-hop body sensor networks," *Sensor Technologies and Applications, International Conference on*, vol. 0, pp. 342–347, 2008.

[17] I. Ishaq, J. Hoebeke, I. Moerman, and P. Demeester, "Internet of things virtual networks: Bringing network virtualization to resource-constrained devices," in *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, Nov 2012, pp. 293–300.

[18] Jeroen Hoebeke, Eli De Poorter, Stefan Bouckaert, Ingrid Moerman, Piet Demeester.

[19] M. Navarro, M. Antonucci, L. Sarakis, and T. Zahariadis, "Vitro architecture: Bringing virtualization to wsn world," in *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, Oct 2011, pp. 831–836.

[20] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[21] A. Armoush, "Design patterns for safety-critical embedded systems," Ph.D. dissertation, Aachen, Techn. Hochsch., Diss.,, 2010.

[22] F. Stajano, N. Hoult, I. Wassell, P. Bennett, C. Middleton, and K. Soga, "Smart bridges, smart tunnels: Transforming wireless sensor networks from research prototypes into robust engineering infrastructure," 2009.

[23] S. Mukhopadhyay, C. Schurgers, D. Panigrahi, and S. Dey, "Model-based techniques for data reliability in wireless sensor networks," *Mobile Computing, IEEE Transactions on*, vol. 8, no. 4, pp. 528–543, April 2009.

[24] R. Tavakkoli-Moghaddam and J. Safari, "A new mathematical model for a redundancy allocation problem with mixing components redundant and choice of redundancy strategies," *Applied Mathematical Sciences*, vol. 1, no. 45, pp. 2221–2230, 2007.

[25] W. Yuan, S. Krishnamurthy, and S. Tripathi, "Improving the reliability of event reports in wireless sensor networks," in *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on*, vol. 1, June 2004, pp. 220–225 Vol.1.

[26] James Brown, Utz Roedig, Carlo Alberto Boano, Kay Romer. and Nicolas Tsiftes, "How temperature affects iot communication," in *the 11th European Conference on Wireless Sensor Networks (EWSN)*. IEEE, 2014.

[27] M. Fairbairn, I. Bate, and J. Stankovic, "Improving the dependability of sensornets," in *Distributed Computing in Sensor Systems (DCOSS), 2013 IEEE International Conference on*, May 2013, pp. 274–282.

[28] Wu, Yafeng and Kapitanova, Krasimira and Li, Jingyuan and Stankovic, John A. and Son, Sang H. and Whitehouse, Kamin, "Run time assurance of application-level requirements in wireless sensor networks," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, ser. IPSN '10. New York, NY, USA: ACM, 2010, pp. 197–208.

[29] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, "Senshare: Transforming sensor networks into multi-application sensing infrastructures," in *Wireless Sensor Networks*, ser. Lecture Notes in Computer Science, G. Picco and W. Heinzelman, Eds. Springer Berlin Heidelberg, 2012, vol. 7158, pp. 65–81.

[30] S. Kabadayi, A. Pridgen, and C. Julien, "Virtual sensors: Abstracting data from physical sensors," in *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, ser. WOWMOM '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 587–592.

[31] N. Raveendranathan, S. Galzarano, V. Loseu, R. Gravina, R. Giannantonio, M. Sgroi, R. Jafari, and G. Fortino, "From modeling to implementation of virtual sensors in body sensor networks," *Sensors Journal, IEEE*, vol. 12, no. 3, pp. 583–593, March 2012.

[32] D. Bruneo, A. Puliafito, and M. Scarpa, "Dependability analysis of wireless sensor networks with active-sleep cycles and redundant nodes," in *Proceedings of the First Workshop on DYnamic Aspects in DEpendability Models for Fault-Tolerant Systems*, ser. DYADEM-FTS '10. New York, NY, USA: ACM, 2010, pp. 25–30.

[33] H. Kwon, T. H. Kim, S. Choi, and B. G. Lee, "A cross-layer strategy for energy-efficient reliable delivery in wireless sensor networks," *Wireless Communications, IEEE Transactions on*, vol. 5, no. 12, pp. 3689–3699, December 2006.

[34] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and beyond: New aggregation techniques for sensor networks," in *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 239–249.

[35] K. Dar, A. Taherkordi, H. Baraki, F. Eliassen, and K. Geihs, "A resource oriented integration architecture for the internet of things: A business process perspective," *Pervasive and Mobile Computing*, vol. 20, pp. 145 – 159, 2015.

[36] S. Haller, A. Serbanati, M. Bauer, and F. Carrez, "A domain model for the internet of things," in *IEEE International Conference on Cyber, Physical and Social Computing*, Aug 2013, pp. 411–417.

[37] Luis Sanchez , Veronica Gutierrez,Jose A Galache, Pablo Sotres, Juan Ramn Santana,Javier Casanueva , Luis Muoz, "Smartsantander: Experimentation and service provision in the smart city," in *Global Wireless Summit*, 2013.